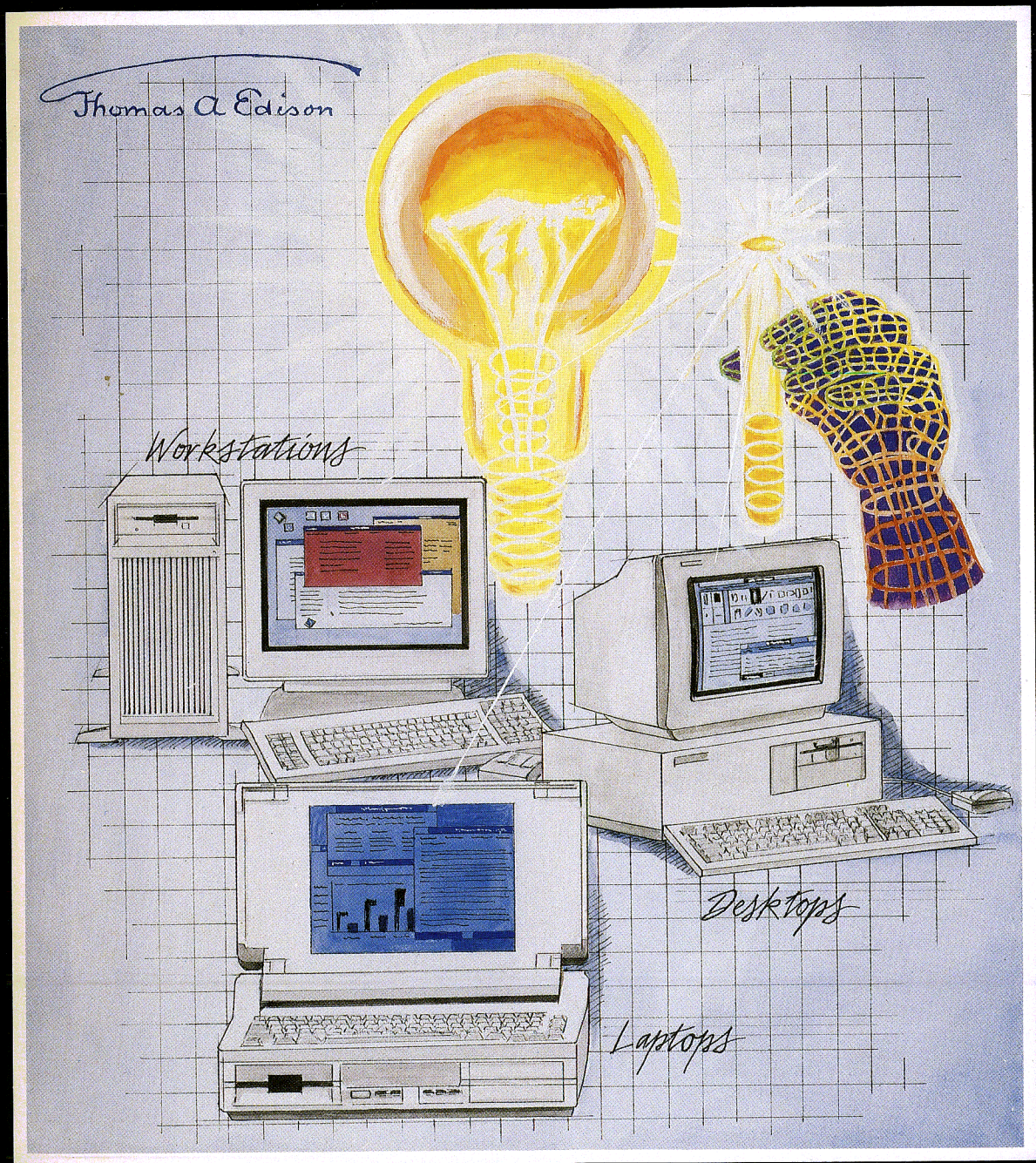
intel®

Microprocessors

Volume II



Benedict Norbert Wong

Order Number: 230843-008

LITERATURE

To order Intel Literature or obtain literature pricing information in the U.S. and Canada call or write Intel Literature Sales. In Europe and other international locations, please contact your ***local*** sales office or distributor.

INTEL LITERATURE SALES
P.O. BOX 7641
Mt. Prospect, IL 60056-7641

In the U.S. and Canada
call toll free
(800) 548-4725

CURRENT HANDBOOKS

Product line handbooks contain data sheets, application notes, article reprints and other design information. All handbooks can be ordered individually, and most are available in a pre-packaged set in the U.S. and Canada.

TITLE	INTEL ORDER NUMBER	ISBN
SET OF THIRTEEN HANDBOOKS (Available in U.S. and Canada)	**231003**	**N/A**
CONTENTS LISTED BELOW FOR INDIVIDUAL ORDERING:		
COMPONENTS QUALITY/RELIABILITY	210997	1-55512-132-2
EMBEDDED APPLICATIONS	270648	1-55512-123-3
8-BIT EMBEDDED CONTROLLERS	270645	1-55512-121-7
16-BIT EMBEDDED CONTROLLERS	270646	1-55512-120-9
16/32-BIT EMBEDDED PROCESSORS	270647	1-55512-122-5
MEMORY PRODUCTS	210830	1-55512-117-9
MICROCOMMUNICATIONS	231658	1-55512-119-5
MICROCOMPUTER PRODUCTS	280407	1-55512-118-7
MICROPROCESSORS	230843	1-55512-115-2
PACKAGING	240800	1-55512-128-4
PERIPHERAL COMPONENTS	296467	1-55512-127-6
PRODUCT GUIDE (Overview of Intel's complete product lines)	210846	1-55512-116-0
PROGRAMMABLE LOGIC	296083	1-55512-124-1
ADDITIONAL LITERATURE: (Not included in handbook set)		
AUTOMOTIVE HANDBOOK	231792	1-55512-125-x
INTERNATIONAL LITERATURE GUIDE (Available in Europe only)	E00029	N/A
CUSTOMER LITERATURE GUIDE	210620	N/A
MILITARY HANDBOOK (2 volume set)	210461	1-55512-126-8
SYSTEMS QUALITY/RELIABILITY	231762	1-55512-046-6

intel®

U.S. and CANADA LITERATURE ORDER FORM

NAME: _____

COMPANY: _____

ADDRESS: _____

CITY: _____ STATE: _____ ZIP: _____

COUNTRY: _____

PHONE NO.: ( ) _____

ORDER NO.	TITLE	QTY.		PRICE	TOTAL
			×		=
			×		=
			×		=
			×		=
			×		=
			×		=
			×		=
			×		=
			×		=
			×		=

Subtotal _____

Must Add Your Local Sales Tax _____

Include postage:
Must add 15% of Subtotal to cover U.S. and Canada postage. (20% all other.)

→ Postage _____

Total _____

Pay by check, money order, or include company purchase order with this form ($100 minimum). We also accept VISA, MasterCard or American Express. Make payment to Intel Literature Sales. Allow 2-4 weeks for delivery.

☐ VISA ☐ MasterCard ☐ American Express Expiration Date _____

Account No. _____

Signature _____

Mail To: Intel Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641

International Customers outside the U.S. and Canada should use the International order form on the next page or contact their local Sales Office or Distributor.

For phone orders in the U.S. and Canada
Call Toll Free: (800) 548-4725

Prices good until 12/31/91.
Source HB

intel®

INTERNATIONAL LITERATURE ORDER FORM

NAME: _____

COMPANY: _____

ADDRESS: _____

CITY: _____ STATE: _____ ZIP: _____

COUNTRY: _____

PHONE NO.: ( ) _____

ORDER NO.	TITLE	QTY.		PRICE	TOTAL
			×		=
			×		=
			×		=
			×		=
			×		=
			×		=
			×		=
			×		=
			×		=
			×		=

Subtotal _____

Must Add Your Local Sales Tax _____

Total _____

PAYMENT

Cheques should be made payable to your ***local*** Intel Sales Office (see inside back cover).

Other forms of payment may be available in your country. Please contact the Literature Coordinator at your ***local*** Intel Sales Office for details.

The completed form should be marked to the attention of the LITERATURE COORDINATOR and returned to your ***local*** Intel Sales Office.



Intel Corporation is a leading supplier of microcomputer components, modules and systems. When Intel invented the microprocessor in 1971, it created the era of the microcomputer. Today, Intel architectures are considered world standards. Whether used in embedded applications such as automobiles, printers and microwave ovens, or as the CPU in personal computers, client servers or supercomputers, Intel delivers leading-edge technology.

MICROPROCESSORS

VOLUME II

1991

About Our Cover:

Thinkers, inventors, and artists throughout history have breathed life into their ideas by converting them into rough working sketches, models, and products. This series of covers shows a few of these creations, along with the applications and products created by Intel customers.

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel products:

287, 376, 386, 387, 486, 4-SITE, Above, ACE51, ACE96, ACE186, ACE196, ACE960, ActionMedia, BITBUS, COMMputer, CREDIT, Data Pipeline, DVI, ETOX, FaxBACK, Genius, i, i$\overset{\wedge}{\text{T}}$, i486, i750, i860, ICE, iCEL, ICEVIEW, iCS, iDBP, iDIS, I^2ICE, iLBX, iMDDX, iMMX, Inboard, Insite, Intel, int$_e$l, Intel386, int$_e$lBOS, Intel Certified, Intelevision, int$_e$ligent Identifier, int$_e$ligent Programming, Intellec, Intellink, iOSP, iPAT, iPDS, iPSC, iRMK, iRMX, iSBC, iSBX, iSDM, iSXM, Library Manager, MAPNET, MCS, Megachassis, MICROMAINFRAME, MULTICHANNEL, MULTIMODULE, MultiSERVER, ONCE, OpenNET, OTP, Pro750, PROMPT, Promware, QUEST, QueX, Quick-Erase, Quick-Pulse Programming, READY LAN, RMX/80, RUPI, Seamless, SLD, SugarCube, SX, ToolTALK, UPI, VAPI, Visual Edge, VLSiCEL, and ZapCode, and the combination of ICE, iCS, iRMX, iSBC, iSBX, iSXM, MCS, or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corporation.

CHMOS and HMOS are patented processes of Intel Corp.

Intel Corporation and Intel's FASTPATH are not affiliated with Kinetics, a division of Excelan, Inc. or its FASTPATH trademark or products.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641

©INTEL CORPORATION 1990

CUSTOMER SUPPORT

INTEL'S COMPLETE SUPPORT SOLUTION WORLDWIDE

Customer Support is Intel's complete support service that provides Intel customers with hardware support, software support, customer training, consulting services and network management services. For detailed information contact your local sales offices.

After a customer purchases any system hardware or software product, service and support become major factors in determining whether that product will continue to meet a customer's expectations. Such support requires an international support organization and a breadth of programs to meet a variety of customer needs. As you might expect, Intel's customer support is extensive. It can start with assistance during your development effort to network management. 100 Intel sales and service offices are located worldwide—in the U.S., Canada, Europe and the Far East. So wherever you're using Intel technology, our professional staff is within close reach.

HARDWARE SUPPORT SERVICES

Intel's hardware maintenance service, starting with complete on-site installation will boost your productivity from the start and keep you running at maximum efficiency. Support for system or board level products can be tailored to match your needs, from complete on-site repair and maintenance support to economical carry-in or mail-in factory service.

Intel can provide support service for not only Intel systems and emulators, but also support for equipment in your development lab or provide service on your product to your end-user/customer.

SOFTWARE SUPPORT SERVICES

Software products are supported by our Technical Information Service (TIPS) that has a special toll free number to provide you with direct, ready information on known, documented problems and deficiencies, as well as work-arounds, patches and other solutions.

Intel's software support consists of two levels of contracts. Standard support includes TIPS (Technical Information Phone Service), updates and subscription service (product-specific troubleshooting guides and; *COMMENTS Magazine*). Basic support consists of updates and the subscription service. Contracts are sold in environments which represent product groupings (e.g., iRMX® environment).

NETWORK SERVICE AND SUPPORT

Today's broad spectrum of powerful networking capabilities are only as good as the customer support provided by the vendor. Intel offers network services and support structured to meet a wide variety of end-user computing needs. From a ground up design of your network's physical and logical design to implementation, installation and network wide maintenance. From software products to turn-key system solutions; Intel offers the customer a complete networked solution. With over 10 years of network experience in both the commercial and Government arena; network products, services and support from Intel provide you the most optimized network offering in the industry.

CONSULTING SERVICES

Intel provides field system engineering consulting services for any phase of your development or application effort. You can use our system engineers in a variety of ways ranging from assistance in using a new product, developing an application, personalizing training and customizing an Intel product to providing technical and management consulting. Systems Engineers are well versed in technical areas such as microcommunications, real-time applications, embedded microcontrollers, and network services. You know your application needs; we know our products. Working together we can help you get a successful product to market in the least possible time.

CUSTOMER TRAINING

Intel offers a wide range of instructional programs covering various aspects of system design and implementation. In just three to ten days a limited number of individuals learn more in a single workshop than in weeks of self-study. For optimum convenience, workshops are scheduled regularly at Training Centers worldwide or we can take our workshops to you for on-site instruction. Covering a wide variety of topics, Intel's major course categories include: architecture and assembly language, programming and operating systems, BITBUS™ and LAN applications.

intel®

DATA SHEET DESIGNATIONS

Intel uses various data sheet markings to designate each phase of the document as it relates to the product. The marking appears in the upper, right-hand corner of the data sheet. The following is the definition of these markings:

Data Sheet Marking	Description
Product Preview	Contains information on products in the design phase of development. Do not finalize a design with this information. Revised information will be published when the product becomes available.
Advanced Information	Contains information on products being sampled or in the initial production phase of development.*
Preliminary	Contains preliminary information on new products in production.*
No Marking	Contains information on products in full production.*

*Specifications within these data sheets are subject to change without notice. Verify with your local Intel sales office that you have the latest data sheet before finalizing a design.



Overview 1

8086 Microprocessor Family 2

80286 Microprocessor Family 3

Development Tools for the 8086, 80186, 80188, and 80286 4

Intel386™ Family 5

i860™ Microprocessor Family 6

i750™ Video Processor Family 7

Development Tools for the 80386 and 80486 8

Table of Contents

Alphanumeric Index xi

CHAPTER 1

Overview

Introduction 1-1

CHAPTER 2

8086 Microprocessor Family

DATA SHEETS

8086 16-Bit HMOS Microprocessor 8086/8086-2/8086-1* 2-1
80C86A 16-Bit CHMOS Microprocessor 2-31
8088 8-Bit HMOS Microprocessor 8088/8088-2 2-60
80C88A 8-Bit CHMOS Microprocessor 2-90
8087 Math Coprocessor 2-122

CHAPTER 3

80286 Microprocessor Family

DATA SHEETS

80C286 High Performance Microprocessor with Memory Management and Protection 3-1
80286 High Performance Microprocessor with Memory Management and Protection 3-60
80287XL/XLT CHMOS III Math CoProcessor 3-116
82C288 Bus Controller for 80286 Processors (82C288-12, 82C288-10, 82C288-8) . 3-148
82C284 Clock Generator and Ready Interface for 80286 Processors (82C284-12, 82C284-10, 82C284-8) 3-169

CHAPTER 4

Development Tools for the 8086, 80186, 80188, and 80286

LANGUAGES AND SOFTWARE DEVELOPMENT TOOLS

8086/80186 Software Development Packages 4-1
iC-86/286 C Compiler 4-8
AEDIT Source Code and Text Editor 4-12
iPAT Performance Analysis Tool 4-14

IN-CIRCUIT EMULATORS

I2ICE In-Circuit Emulation System 4-18
ICE-186 and ICE-188 In-Circuit Emulators 4-22
ICE-186EB and ICE-188EB In-Circuit Emulators 4-25
ICE-286 In-Circuit Emulator 4-32

CHAPTER 5

INTEL386™ Family

DATA SHEETS

i486 Microprocessor 5-1
485Turbocache Module i486 Microprocessor Cache Upgrade 5-177
82485 Second Level Cache Controller for the i486 Microprocessor 5-206
AP-447 A Memory Subsystem for the i486 CPU Including Second Level Cache 5-207
386 DX Microprocessor High Performance 32-Bit CHMOS Microprocessor with Integrated Memory Management 5-287
387 DX Math Coprocessor 5-425
82395DX High Performance 386 Smart Cache 5-466
82385 High Performance 32-Bit Cache Controller 5-547
AP-442 33 MHz 386 System Design Considerations 5-620
386 SL Microprocessor SuperSet 5-731
386 SX Microprocessor 5-864
387 SX Math Coprocessor 5-962

Table of Contents (Continued)

82395SX 386 SX Smart Cache 5-1002
82385SX High Performance Cache Controller 5-1003
82380 High Performance 32-Bit DMA Controller with Integrated System Support Peripherals 5-1080
376 High Performance 32-Bit Embedded Processor 5-1217
82370 Integrated System Peripheral 5-1312

CHAPTER 6
i860™ Microprocessor Family
i860 64-Bit Microprocessor 6-1
AP-434 Using i860 Microprocessor Graphics Instructions for 3-D Rendering 6-81
AP-435 Fast Fourier Transforms on the i860 Microprocessor 6-96

CHAPTER 7
i750™ Video Processor Family
82750PB Pixel Processor 7-1
82750DB Display Processor 7-3

CHAPTER 8
Development Tools for the 80386 and 80486
LANGUAGES AND SOFTWARE DEVELOPMENT TOOLS
Intel386/i486 Family Development Support 8-1
Intel 376 Family Development Support 8-11
ICD-486 In-Circuit Debugger 8-23
IN-CIRCUIT EMULATORS
Intel386 Family of In-Circuit Emulators 8-29
Intel i486 In-Circuit Emulator 8-55

Alphanumeric Index

376 High Performance 32-Bit Embedded Processor 5-1217
386 DX Microprocessor High Performance 32-Bit CHMOS Microprocessor with Integrated Memory Management 5-287
386 SL Microprocessor SuperSet 5-731
386 SX Microprocessor 5-864
387 DX Math Coprocessor 5-425
387 SX Math Coprocessor 5-962
485Turbocache Module i486 Microprocessor Cache Upgrade 5-177
80286 High Performance Microprocessor with Memory Management and Protection 3-60
80287XL/XLT CHMOS III Math CoProcessor 3-116
8086 16-Bit HMOS Microprocessor 8086/8086-2/8086-1* 2-1
8086/80186 Software Development Packages 4-1
8087 Math Coprocessor 2-122
8088 8-Bit HMOS Microprocessor 8088/8088-2 2-60
80C286 High Performance Microprocessor with Memory Management and Protection 3-1
80C86A 16-Bit CHMOS Microprocessor 2-31
80C88A 8-Bit CHMOS Microprocessor 2-90
82370 Integrated System Peripheral 5-1312
82380 High Performance 32-Bit DMA Controller with Integrated System Support Peripherals 5-1080
82385 High Performance 32-Bit Cache Controller 5-547
82385SX High Performance Cache Controller 5-1003
82395DX High Performance 386 Smart Cache 5-466
82395SX 386 SX Smart Cache 5-1002
82485 Second Level Cache Controller for the i486 Microprocessor 5-206
82750DB Display Processor 7-3
82750PB Pixel Processor 7-1
82C284 Clock Generator and Ready Interface for 80286 Processors (82C284-12, 82C284-10, 82C284-8) 3-169
82C288 Bus Controller for 80286 Processors (82C288-12, 82C288-10, 82C288-8) 3-148
AEDIT Source Code and Text Editor 4-12
AP-434 Using i860 Microprocessor Graphics Instructions for 3-D Rendering 6-81
AP-435 Fast Fourier Transforms on the i860 Microprocessor 6-96
AP-442 33 MHz 386 System Design Considerations 5-620
AP-447 A Memory Subsystem for the i486 CPU Including Second Level Cache 5-207
I2ICE In-Circuit Emulation System 4-18
i486 Microprocessor 5-1
i860 64-Bit Microprocessor 6-1
iC-86/286 C Compiler 4-8
ICD-486 In-Circuit Debugger 8-23
ICE-186 and ICE-188 In-Circuit Emulators 4-22
ICE-186EB and ICE-188EB In-Circuit Emulators 4-25
ICE-286 In-Circuit Emulator 4-32
Intel 376 Family Development Support 8-11
Intel386 Family of In-Circuit Emulators 8-29
Intel386/i486 Family Development Support 8-1
Intel i486 In-Circuit Emulator 8-55
iPAT Performance Analysis Tool 4-14

Intel386™ Family

5

i486™ MICROPROCESSOR

- **Binary Compatible with Large Software Base**
  - **— MS-DOS*, OS/2**, Windows**
  - **— UNIX*** System V/386**
  - **— iRMX®, iRMK™ Kernels**
- **High Integration Enables On-Chip**
  - **— 8 Kbyte Code and Data Cache**
  - **— Floating Point Unit**
  - **— Paged, Virtual Memory Management**
- **Easy To Use**
  - **— Built-In Self Test**
  - **— Hardware Debugging Support**
  - **— Intel Software Support**
  - **— Extensive Third Party Software Support**
- **168-Pin Grid Array Package**
- **High Performance Design**
  - **— Frequent Instructions Execute in One Clock**
  - **— 25 MHz and 33 MHz Clock Frequencies**
  - **— 80 and 106 Mbyte/Sec Burst Bus**
  - **— CHMOS IV Process Technology**
  - **— Dynamic Bus Sizing for 8-, 16- and 32-Bit Busses**
- **Complete 32-Bit Architecture**
  - **— Address and Data Busses**
  - **— Registers**
  - **— 8-, 16- and 32-Bit Data Types**
- **Multiprocessor Support**
  - **— Multiprocessor Instructions**
  - **— Cache Consistency Protocols**
  - **— Support for Second Level Cache**

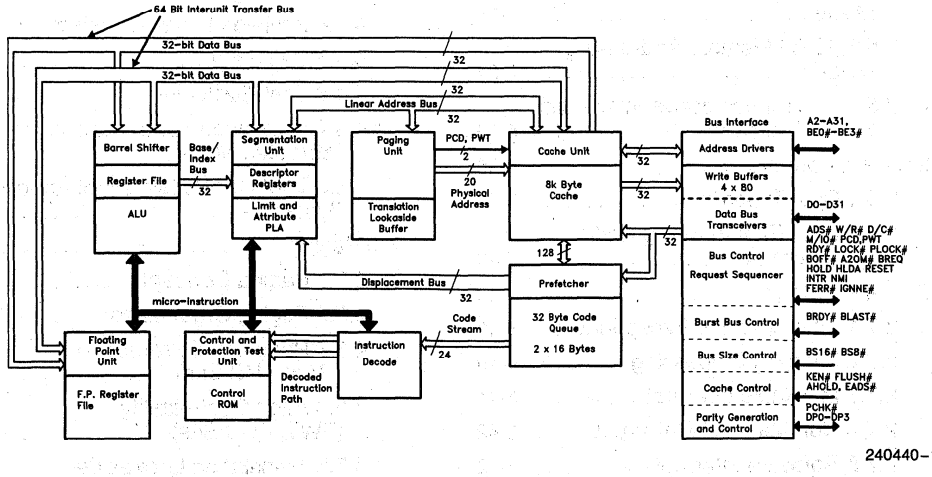
The i486™ CPU offers the highest performance for DOS, OS/2, Windows and UNIX System V/386 applications. It is 100% binary compatible with the 386™ CPU. Over one million transistors integrate cache memory, floating point hardware and memory management on-chip while retaining binary compatibility with previous members of the X86 architectural family. Frequently used instructions execute in one cycle resulting in RISC performance levels. An 8 Kbyte unified code and data cache combined with a 106 Mbyte/Sec burst bus at 33.3 MHz ensure high system throughput even with inexpensive DRAMs.

New features enhance multiprocessing systems. New instructions speed manipulation of memory based semaphores. On-chip hardware ensures cache consistency and provides hooks for multilevel caches.

The built in self test extensively tests on-chip logic, cache memory and the on-chip paging translation cache. Debug features include breakpoint traps on code execution and data accesses.

5

i486™ Microprocessor Pipelined 32-Bit Microarchitecture



240440-1

iRMX, iRMK, 386, 387, 486, i486 are trademarks of Intel Corporation.
*MS-DOS® is a registered trademark of Microsoft Corporation.
**OS/2™ is a trademark of Microsoft Corporation.
***UNIX™ is a trademark of AT&T.

November 1990
Order Number: 240440-003

i486™ MICROPROCESSOR

CONTENTS

PAGE

1.0 TABLE OF CONTENTS 5-2
Pinout 5-6
Brief Pin Descriptions 5-9
Component and Revision ID 5-13

2.0 ARCHITECTURAL OVERVIEW 5-14
2.1 Register Set 5-14
2.1.1 Base Architecture Registers 5-15
2.1.2 System Level Registers 5-19
2.1.3 Floating Point Registers 5-23
2.1.4 Debug and Test Registers 5-30
2.1.5 Register Accessibility 5-30
2.1.6 Compatibility 5-31
2.2 Instruction Set 5-32
2.3 Memory Organization 5-32
2.3.1 Address Spaces 5-32
2.3.2 Segment Register Usage 5-33
2.4 I/O Space 5-33
2.5 Addressing Modes 5-34
2.5.1 Addressing Modes Overview 5-34
2.5.2 Register and Immediate Modes 5-34
2.5.3 32-Bit Memory Addressing Modes 5-34
2.5.4 Differences between 16- and 32-Bit Addresses 5-36
2.6 Data Formats 5-36
2.6.1 Data Types 5-36
2.6.2 Little Endian vs Big Endian Data Formats 5-40
2.7 Interrupts 5-40
2.7.1 Interrupts and Exceptions 5-40
2.7.2 Interrupt Processing 5-40
2.7.3 Maskable Interrupt 5-41
2.7.4 Non-Maskable Interrupt 5-42
2.7.5 Software Interrupts 5-42
2.7.6 Interrupt and Exception Priorities 5-42
2.7.7 Instruction Restart 5-43

CONTENTS

PAGE

2.7.8 Double Fault 5-43
2.7.9 Floating Point Interrupt Vectors 5-43

3.0 REAL MODE ARCHITECTURE 5-44
3.1 Real Mode Introduction 5-44
3.2 Memory Addressing 5-45
3.3 Reserved Locations 5-45
3.4 Interrupts 5-45
3.5 Shutdown and Halt 5-45

4.0 PROTECTED MODE ARCHITECTURE 5-46
4.1 Introduction 5-46
4.2 Addressing Mechanism 5-46
4.3 Segmentation 5-47
4.3.1 Segmentation Introduction 5-47
4.3.2 Terminology 5-47
4.3.3 Descriptor Tables 5-48
4.3.4 Descriptors 5-49
4.4 Protection 5-58
4.4.1 Protection Concepts 5-58
4.4.2 Rules of Privilege 5-59
4.4.3 Privilege Levels 5-59
4.4.4 Privilege Level Transfers 5-60
4.4.5 Call Gates 5-63
4.4.6 Task Switching 5-63
4.4.7 Initialization and Transition to Protected Mode 5-64
4.4.8 Tools for Building Protected Systems 5-65
4.5 Paging 5-65
4.5.1 Paging Concepts 5-65
4.5.2 Paging Organization 5-66
4.5.3 Page Level Protection (R/W, U/S Bits) 5-67
4.5.4 Page Cacheability (PWT, PCD Bits) 5-68
4.5.5 Translation Lookaside Buffer 5-68
4.5.6 Paging Operation 5-69
4.5.7 Operating System Responsibilities 5-70

CONTENTS

PAGE

4.6 Virtual 8086 Environment 5-70
4.6.1 Executing 8086 Programs 5-70
4.6.2 Virtual 8086 Addressing Mechanism 5-70
4.6.3 Paging in Virtual Mode 5-70
4.6.4 Protection and Virtual 8086 Mode to I/O Permission Bitmap 5-71
4.6.5 Interrupt Handling 5-72
4.6.6 Entering and Leaving Virtual 8086 Mode 5-73

5.0 ON-CHIP CACHE 5-75
5.1 Cache Organization 5-75
5.2 Cache Control 5-76
5.3 Cache Line Fills 5-76
5.4 Cache Line Invalidations 5-77
5.5 Cache Replacement 5-77
5.6 Page Cacheability 5-78
5.7 Cache Flushing 5-79
5.8 Caching Translation Lookaside Buffer Entries 5-79

6.0 HARDWARE INTERFACE 5-80
6.1 Introduction 5-80
6.2 Signal Descriptions 5-81
6.2.1 Clock (CLK) 5-81
6.2.2 Address Bus (A31–A2, BE0#–BE3#) 5-81
6.2.3 Data Lines (D31–D0) 5-81
6.2.4 Parity 5-82
Data Parity Input/Outputs (DP0–DP3) 5-82
Parity Status Output (PCHK#) 5-82
6.2.5 Bus Cycle Definition 5-82
M/IO#, D/C#, W/R# Outputs 5-82
Bus Lock Output (LOCK#) 5-82
Pseudo-Lock Output (PLOCK#) 5-82

CONTENTS

PAGE

6.2.6 Bus Control 5-83
Address Status Output (ADS#) 5-83
Non-Burst Ready Input (RDY#) 5-83
6.2.7 Burst Control 5-83
Burst Ready Input (BRDY#) 5-83
Burst Last Output (BLAST#) 5-83
6.2.8 Interrupt Signals 5-84
Reset Input (RESET) 5-84
Maskable Interrupt Request Input (INTR) 5-84
Non-Maskable Interrupt Request Input (NMI) 5-84
6.2.9 Bus Arbitration Signals 5-84
Bus Request Output (BREQ) 5-84
Bus Hold Request Input (HOLD) 5-84
Bus Hold Acknowledge Output (HLDA) 5-84
Backoff Input (BOFF#) 5-85
6.2.10 Cache Invalidation 5-85
Address Hold Request Input (AHOLD) 5-85
External Address Valid Input (EADS#) 5-85
6.2.11 Cache Control 5-85
Cache Enable Input (KEN#) 5-85
Cache Flush Input (FLUSH#) 5-86
6.2.12 Page Cacheability Outputs (PWT, PCD) 5-86
6.2.13 Numeric Error Reporting 5-86
Floating Point Error Output (FERR#) 5-86
Ignore Numeric Error Input (IGNNE#) 5-86
6.2.14 Bus Size Control (BS16#, BS8#) 5-86

CONTENTS

PAGE

6.2.15 Address Bit 20 Mask (A20M#) 5-87
6.3 Write Buffers 5-87
6.3.1 Write Buffers and I/O Cycles 5-88
6.3.2 Write Buffers Implications on Locked Bus Cycles 5-88
6.4 Interrupt and Non-Maskable Interrupt Interface 5-88
6.4.1 Interrupt Logic 5-88
6.4.2 NMI Logic 5-89
6.5 Reset and Initialization 5-89
6.5.1 Pin State during Reset 5-90

7.0 BUS OPERATION 5-92
7.1 Data Transfer Mechanism 5-92
7.1.1 Memory and I/O Spaces 5-92
7.1.2 Memory and I/O Space Organization 5-93
7.1.3 Dynamic Data Bus Sizing 5-94
7.1.4 Interfacing with 8-, 16- and 32-bit Memories 5-95
7.1.5 Dynamic Bus Sizing during Cache Line Fills 5-97
7.1.6 Operand Alignment 5-97
7.2 Bus Functional Description 5-98
7.2.1 Non-Cacheable Non-Burst Single Cycle 5-98
7.2.2 Multiple and Burst Cycle Bus Transfers 5-99
7.2.3 Cacheable Cycles 5-103
7.2.4 Burst Mode Details 5-106
7.2.5 8- and 16-Bit Cycles 5-110
7.2.6 Locked Cycles 5-112
7.2.7 Pseudo-Locked Cycles 5-113
7.2.8 Invalidate Cycles 5-113
7.2.9 Bus Hold 5-117
7.2.10 Interrupt Acknowledge 5-117
7.2.11 Special Bus Cycles 5-119
7.2.12 Bus Cycle Restart 5-120
7.2.13 Bus States 5-121

CONTENTS

PAGE

7.2.14 Floating Point Error Handling 5-122

8.0 TESTABILITY 5-122
8.1 Built-In Self Test (BIST) 5-122
8.2 On-Chip Cache Testing 5-123
8.2.1 Cache Testing Registers TR3, TR4 and TR5 5-123
Cache Data Test Register: TR3 5-124
Cache Status Test Register: TR4 5-124
Cache Control Test Register: TR5 5-124
8.2.2 Cache Testability Write 5-124
8.2.3 Cache Testability Read 5-126
8.2.4 Flush Cache 5-126
8.3 Translation Lookaside Buffer (TLB) Testing 5-126
8.3.1 Translation Lookaside Buffer Organization 5-126
8.3.2 TLB Test Registers: TR6 and TR7 5-127
Command Test Register: TR6 5-128
Data Test Register: TR7 5-128
8.3.3 TLB Write Test 5-129
8.3.4 TLB Lookup Test 5-129
8.4 Tristate Output Test Mode 5-129

9.0 DEBUGGING SUPPORT 5-130
9.1 Breakpoint Instructions 5-130
9.2 Single Step Instructions 5-130
9.3 Debug Registers 5-130
9.3.1 Linear Address Breakpoint Registers 5-130
9.3.2 Debug Control Register 5-130
9.3.3 Debug Status Register 5-133
9.3.4 Use of Resume Flag (RF) in Flag Register 5-133

i486™ MICROPROCESSOR

CONTENTS

PAGE

10.0 INSTRUCTION SET SUMMARY .. 5-133

10.1 486™ Microprocessor Instruction Encoding and Clock Count Summary 5-134

10.2 Instruction Encoding 5-152

10.2.1 Overview 5-152

10.2.2 32-Bit Extensions of the Instruction Set 5-153

10.2.3 Encoding of Integer Instruction Fields 5-154

10.2.4 Encoding of Floating Point Instruction Fields 5-160

11.0 DIFFERENCES WITH THE 386™ MICROPROCESSOR 5-160

CONTENTS

PAGE

12.0 ELECTRICAL DATA 5-161

12.1 Power and Grounding 5-161

12.2 Maximum Ratings 5-161

12.3 D.C. Specifications 5-162

12.4 A.C. Specifications 5-162

12.5 Designing for ICD-486 5-167

13.0 MECHANICAL DATA 5-172

13.1 Package Thermal Specifications 5-173

14.0 SUGGESTED SOURCES FOR i486 ACCESSORIES 5-174

15.0 REVISION HISTORY 5-175

5

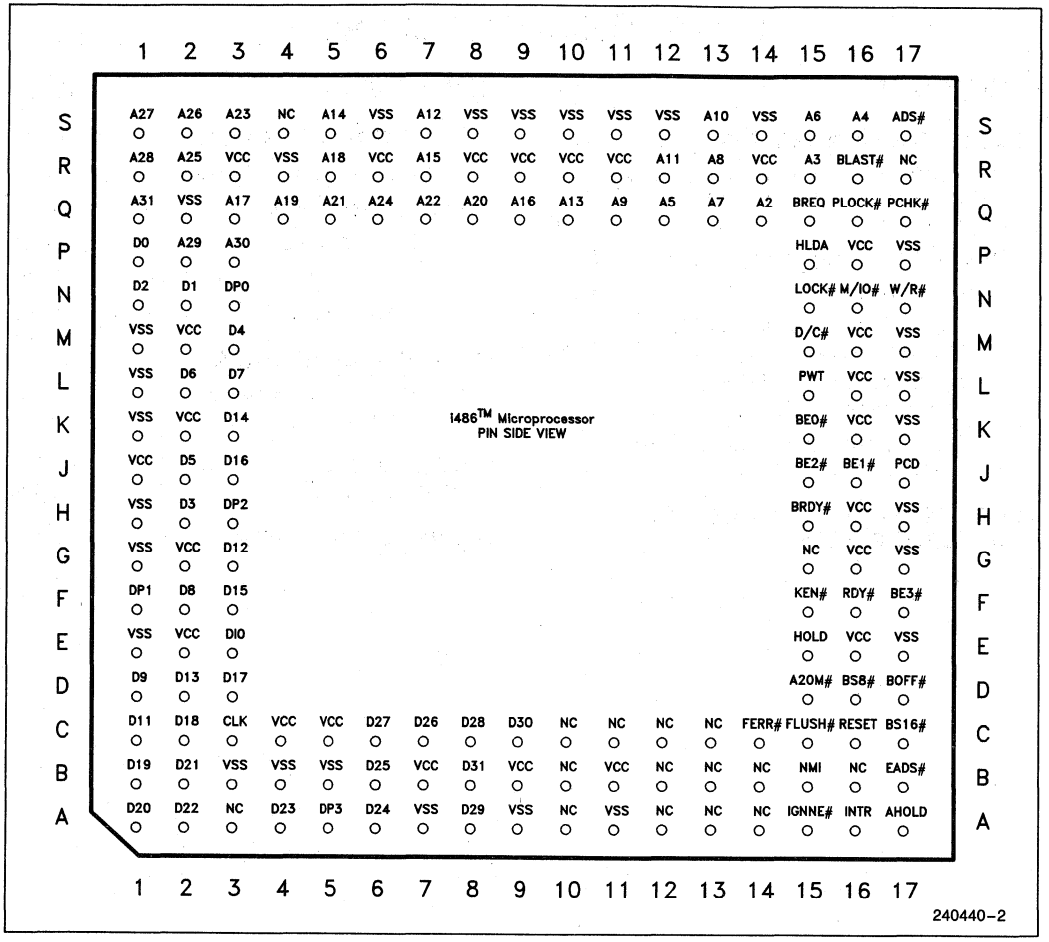

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
S A27 A26 A23 NC A14 VSS A12 VSS VSS VSS VSS VSS A10 VSS A6 A4 ADS#
R A28 A25 VCC VSS A18 VCC A15 VCC VCC VCC VCC A11 A8 VCC A3 BLAST# NC
Q A31 VSS A17 A19 A21 A24 A22 A20 A16 A13 A9 A5 A7 A2 BREQ PLOCK# PCHK#
P D0 A29 A30 HLDA VCC VSS
N D2 D1 DP0 LOCK# M/IO# W/R#
M VSS VCC D4 D/C# VCC VSS
L VSS D6 D7 PWT VCC VSS
K VSS VCC D14 BE0# VCC VSS
J VCC D5 D16 BE2# BE1# PCD
H VSS D3 DP2 BRDY# VCC VSS
G VSS VCC D12 NC VCC VSS
F DP1 D8 D15 KEN# RDY# BE3#
E VSS VCC DIO HOLD VCC VSS
D D9 D13 D17 A20M# BS8# BOFF#
C D11 D18 CLK VCC VCC D27 D26 D28 D30 NC NC NC NC FERR# FLUSH# RESET BS16#
B D19 D21 VSS VSS VSS D25 VCC D31 VCC NC VCC NC NC NC NMI NC EADS#
A D20 D22 NC D23 DP3 D24 VSS D29 VSS NC VSS NC NC NC IGNNE# INTR AHOLD
i486™ Microprocessor
PIN SIDE VIEW
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
240440-2

Figure 1.1

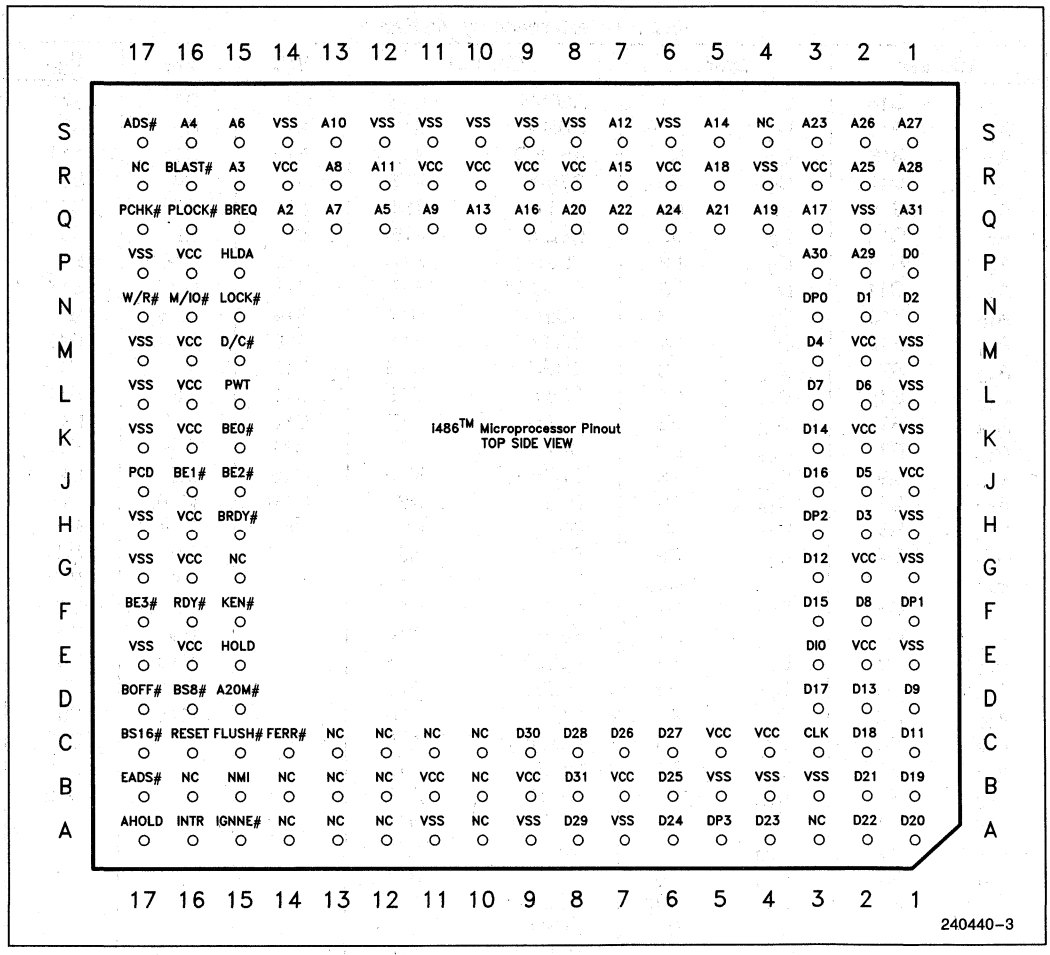

17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
S ADS# A4 A6 VSS A10 VSS VSS VSS VSS VSS A12 VSS A14 NC A23 A26 A27
R NC BLAST# A3 VCC A8 A11 VCC VCC VCC VCC A15 VCC A18 VSS VCC A25 A28
Q PCHK# PLOCK# BREQ A2 A7 A5 A9 A13 A16 A20 A22 A24 A21 A19 A17 VSS A31
P VSS VCC HLDA A30 A29 D0
N W/R# M/IO# LOCK# DP0 D1 D2
M VSS VCC D/C# D4 VCC VSS
L VSS VCC PWT D7 D6 VSS
K VSS VCC BE0# D14 VCC VSS
J PCD BE1# BE2# D16 D5 VCC
H VSS VCC BRDY# DP2 D3 VSS
G VSS VCC NC D12 VCC VSS
F BE3# RDY# KEN# D15 D8 DP1
E VSS VCC HOLD DI0 VCC VSS
D BOFF# BS8# A20M# D17 D13 D9
C BS16# RESET FLUSH# FERR# NC NC NC NC D30 D28 D26 D27 VCC VCC CLK D18 D11
B EADS# NC NMI NC NC NC VCC NC VCC D31 VCC D25 VSS VSS VSS D21 D19
A AHOLD INTR IGNNE# NC NC NC VSS NC VSS D29 VSS D24 DP3 D23 NC D22 D20
17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
i486™ Microprocessor Pinout
TOP SIDE VIEW
240440-3

Figure 1.2

Pin Cross Reference by Pin Name

Address		Data		Control		N/C	V_{CC}	V_{SS}
A_2	Q14	D_0	P1	A20M#	D15	A3	B7	A7
A_3	R15	D_1	N2	ADS#	S17	A10	B9	A9
A_4	S16	D_2	N1	AHOLD	A17	A12	B11	A11
A_5	Q12	D_3	H2	BE0#	K15	A13	C4	B3
A_6	S15	D_4	M3	BE1#	J16	A14	C5	B4
A_7	Q13	D_5	J2	BE2#	J15	B10	E2	B5
A_8	R13	D_6	L2	BE3#	F17	B12	E16	E1
A_9	Q11	D_7	L3	BLAST#	R16	B13	G2	E17
A_{10}	S13	D_8	F2	BOFF#	D17	B14	G16	G1
A_{11}	R12	D_9	D1	BRDY#	H15	B16	H16	G17
A_{12}	S7	D_{10}	E3	BREQ#	Q15	C10	J1	H1
A_{13}	Q10	D_{11}	C1	BS8#	D16	C11	K2	H17
A_{14}	S5	D_{12}	G3	BS16#	C17	C12	K16	K1
A_{15}	R7	D_{13}	D2	CLK	C3	C13	L16	K17
A_{16}	Q9	D_{14}	K3	D/C#	M15	G15	M2	L1
A_{17}	Q3	D_{15}	F3	DP0	N3	R17	M16	L17
A_{18}	R5	D_{16}	J3	DP1	F1	S4	P16	M1
A_{19}	Q4	D_{17}	D3	DP2	H3		R3	M17
A_{20}	Q8	D_{18}	C2	DP3	A5		R6	P17
A_{21}	Q5	D_{19}	B1	EADS#	B17		R8	Q2
A_{22}	Q7	D_{20}	A1	FERR#	C14		R9	R4
A_{23}	S3	D_{21}	B2	FLUSH#	C15		R10	S6
A_{24}	Q6	D_{22}	A2	HLDA	P15		R11	S8
A_{25}	R2	D_{23}	A4	HOLD	E15		R14	S9
A_{26}	S2	D_{24}	A6	IGNNE#	A15			S10
A_{27}	S1	D_{25}	B6	INTR	A16			S11
A_{28}	R1	D_{26}	C7	KEN#	F15			S12
A_{29}	P2	D_{27}	C6	LOCK#	N15			S14
A_{30}	P3	D_{28}	C8	M/IO#	N16			
A_{31}	Q1	D_{29}	A8	NMI	B15			
		D_{30}	C9	PCD	J17			
		D_{31}	B8	PCHK#	Q17			
				PWT	L15			
				PLOCK#	Q16			
				RDY#	F16			
				RESET	C16			
				W/R#	N17			

QUICK PIN REFERENCE

What follows is a brief pin description. For detailed signal descriptions refer to Section 6.

Symbol	Type	Name and Function
CLK	I	*Clock* provides the fundamental timing and the internal operating frequency for the 486 microprocessor. All external timing parameters are specified with respect to the rising edge of CLK.
ADDRESS BUS		
A31–A4 A2–A3	I/O O	A31–A2 are the *address lines* of the microprocessor. A31–A2, together with the byte enables BE0#–BE3#, define the physical area of memory or input/output space accessed. Address lines A31–A4 are used to drive addresses into the microprocessor to perform cache line invalidations. Input signals must meet setup and hold times t_{22} and t_{23}. A31–A2 are not driven during bus or address hold.
BE0–3#	O	The *byte enable* signals indicate active bytes during read and write cycles. During the first cycle of a cache fill, the external system should assume that all byte enables are active. BE3# applies to D24–D31, BE2# applies to D16–D23, BE1# applies to D8–D15 and BE0# applies to D0–D7. BE0#–BE3# are active LOW and are not driven during bus hold.
DATA BUS		
D31–D0	I/O	These are the *data lines* for the 486 microprocessor. Lines D0–D7 define the least significant byte of the data bus while lines D24–D31 define the most significant byte of the data bus. These signals must meet setup and hold times t_{22} and t_{23} for proper operation on reads. These pins are driven during the second and subsequent clocks of write cycles.
DATA PARITY		
DP0–DP3	I/O	There is one *data parity* pin for each byte of the data bus. Data parity is generated on all write data cycles with the same timing as the data driven by the 486 microprocessor. Even parity information must be driven back into the microprocessor on the data parity pins with the same timing as read information to insure that the correct parity check status is indicated by the 486 microprocessor. The signals read on these pins do not affect program execution. Input signals must meet setup and hold times t_{22} and t_{23}. DP0–DP3 should be connected to V_{CC} through a pullup resistor in systems which do not use parity. DP0–DP3 are active HIGH and are driven during the second and subsequent clocks of write cycles.
PCHK#	O	*Parity Status* is driven on the PCHK# pin the clock after ready for read operations. The parity status is for data sampled at the end of the previous clock. A parity error is indicated by PCHK# being LOW. Parity status is only checked for enabled bytes as indicated by the byte enable and bus size signals. PCHK# is valid only in the clock immediately after read data is returned to the microprocessor. At all other times PCHK# is inactive (HIGH). PCHK# is never floated.
BUS CYCLE DEFINITION		
M/IO# D/C# W/R#	O O O	The *memory/input-output, data/control* and *write/read* lines are the primary bus definition signals. These signals are driven valid as the ADS# signal is asserted.

M/IO#	D/C#	W/R#	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Halt/Special Cycle
0	1	0	I/O Read
0	1	1	I/O Write
1	0	0	Code Read
1	0	1	Reserved
1	1	0	Memory Read
1	1	1	Memory Write

The bus definition signals are not driven during bus hold and follow the timing of the address bus. Refer to Section 7.2.11 for a description of the special bus cycles.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
BUS CYCLE DEFINITION (Continued)		
LOCK#	O	The *bus lock* pin indicates that the current bus cycle is locked. The 486 microprocessor will not allow a bus hold when LOCK# is asserted (but address holds are allowed). LOCK# goes active in the first clock of the first locked bus cycle and goes inactive after the last clock of the last locked bus cycle. The last locked cycle ends when ready is returned. LOCK# is active LOW and is not driven during bus hold. Locked read cycles will not be transformed into cache fill cycles if KEN# is returned active.
PLOCK#	O	The *pseudo-lock* pin indicates that the current bus transaction requires more than one bus cycle to complete. Examples of such operations are floating point long reads and writes (64 bits), segment table descriptor reads (64 bits), in addition to cache line fills (128 bits). The 486 microprocessor will drive PLOCK# active until the addresses for the last bus cycle of the transaction have been driven regardless of whether RDY# or BRDY# have been returned. Normally PLOCK# and BLAST# are inverse of each other. However during the first bus cycle of a 64-bit floating point write, both PLOCK# and BLAST# will be asserted. PLOCK# is a function of the BS8#, BS16# and KEN# inputs. PLOCK# should be sampled only in the clock ready is returned. PLOCK# is active LOW and is not driven during bus hold.
BUS CONTROL		
ADS#	O	The *address status* output indicates that a valid bus cycle definition and address are available on the cycle definition lines and address bus. ADS# is driven active in the same clock as the addresses are driven. ADS# is active LOW and is not driven during bus hold.
RDY#	I	The *non-burst ready* input indicates that the current bus cycle is complete. RDY# indicates that the external system has presented valid data on the data pins in response to a read or that the external system has accepted data from the 486 microprocessor in response to a write. RDY# is ignored when the bus is idle and at the end of the first clock of the bus cycle. RDY# is active during address hold. Data can be returned to the processor while AHOLD is active. RDY# is active LOW, and is not provided with an internal pullup resistor. RDY# must satisfy setup and hold times t_{16} and t_{17} for proper chip operation.
BURST CONTROL		
BRDY#	I	The *burst ready input* performs the same function during a burst cycle that RDY# performs during a non-burst cycle. BRDY# indicates that the external system has presented valid data in response to a read or that the external system has accepted data in response to a write. BRDY# is ignored when the bus is idle and at the end of the first clock in a bus cycle. BRDY# is sampled in the second and subsequent clocks of a burst cycle. The data presented on the data bus will be strobed into the microprocessor when BRDY# is sampled active. If RDY# is returned simultaneously with BRDY#, BRDY# is ignored and the burst cycle is prematurely aborted. BRDY# is active LOW and is provided with a small pullup resistor. BRDY# must satisfy the setup and hold times t_{16} and t_{17}.
BLAST#	O	The *burst last* signal indicates that the next time BRDY# is returned the burst bus cycle is complete. BLAST# is active for both burst and non-burst bus cycles. BLAST# is active LOW and is not driven during bus hold.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
INTERRUPTS		
RESET	I	The *reset* input forces the 486 microprocessor to begin execution at a known state. The microprocessor cannot begin execution of instructions until at least 1 ms after V_{CC} and CLK have reached their proper DC and AC specifications. The RESET pin should remain active during this time to insure proper microprocessor operation. RESET is active HIGH. RESET is asynchronous but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
INTR	I	The *maskable interrupt* indicates that an external interrupt has been generated. If the internal interrupt flag is set in EFLAGS, active interrupt processing will be initiated. The 486 microprocessor will generate two locked interrupt acknowledge bus cycles in response to the INTR pin going active. INTR must remain active until the interrupt acknowledges have been performed to assure that the interrupt is recognized. INTR is active HIGH and is not provided with an internal pulldown resistor. INTR is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
NMI	I	The *non-maskable interrupt* request signal indicates that an external non-maskable interrupt has been generated. NMI is rising edge sensitive. NMI must be held LOW for at least four CLK periods before this rising edge. NMI is not provided with an internal pulldown resistor. NMI is asynchronous, but must meet setup and hold times t_{20} and t_{21} for recognition in any specific clock.
BUS ARBITRATION		
BREQ	O	The *internal cycle pending* signal indicates that the 486 microprocessor has internally generated a bus request. BREQ is generated whether or not the 486 microprocessor is driving the bus. BREQ is active HIGH and is never floated.
HOLD	I	The *bus hold request* allows another bus master complete control of the 486 microprocessor bus. In response to HOLD going active the 486 microprocessor will float most of its output and input/output pins. HLDA will be asserted after completing the current bus cycle, burst cycle or sequence of locked cycles. The 486 microprocessor will remain in this state until HOLD is deasserted. HOLD is active high and is not provided with an internal pulldown resistor. HOLD must satisfy setup and hold times t_{18} and t_{19} for proper operation.
HLDA	O	*Hold acknowledge* goes active in response to a hold request presented on the HOLD pin. HLDA indicates that the 486 microprocessor has given the bus to another local bus master. HLDA is driven active in the same clock that the 486 microprocessor floats its bus. HLDA is driven inactive when leaving bus hold. HLDA is active HIGH and remains driven during bus hold.
BOFF#	I	The *backoff* input forces the 486 microprocessor to float its bus in the next clock. The microprocessor will float all pins normally floated during bus hold but HLDA will not be asserted in response to BOFF#. BOFF# has higher priority than RDY# or BRDY#; if both are returned in the same clock, BOFF# takes effect. The microprocessor remains in bus hold until BOFF# is negated. If a bus cycle was in progress when BOFF# was asserted the cycle will be restarted. BOFF# is active LOW and must meet setup and hold times t_{18} and t_{19} for proper operation.
CACHE INVALIDATION		
AHOLD	I	The *address hold* request allows another bus master access to the 486 microprocessor's address bus for a cache invalidation cycle. The 486 microprocessor will stop driving its address bus in the clock following AHOLD going active. Only the address bus will be floated during address hold, the remainder of the bus will remain active. AHOLD is active HIGH and is provided with a small internal pulldown resistor. For proper operation AHOLD must meet setup and hold times t_{18} and t_{19}.

5

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
CACHE INVALIDATION (Continued)		
EADS#	I	This signal indicates that a *valid external address* has been driven onto the 486 microprocessor address pins. This address will be used to perform an internal cache invalidation cycle. EADS# is active LOW and is provided with an internal pullup resistor. EADS# must satisfy setup and hold times t_{12} and t_{13} for proper operation.
CACHE CONTROL		
KEN#	I	The *cache enable* pin is used to determine whether the current cycle is cacheable. When the 486 microprocessor generates a cycle that can be cached and KEN# is active, the cycle will become a cache line fill cycle. Returning KEN# active one clock before ready during the last read in the cache line fill will cause the line to be placed in the on-chip cache. KEN# is active LOW and is provided with a small internal pullup resistor. KEN# must satisfy setup and hold times t_{14} and t_{15} for proper operation.
FLUSH#	I	The *cache flush* input forces the 486 microprocessor to flush its entire internal cache. FLUSH# is active low and need only be asserted for one clock. FLUSH# is asynchronous but setup and hold times t_{20} and t_{21} must be met for recognition in any specific clock. FLUSH# being sampled low in the clock before the falling edge of RESET causes the 486 microprocessor to enter the tri-state test mode.
PAGE CACHEABILITY		
PWT PCD	O O	The *page write-through* and *page cache disable* pins reflect the state of the page attribute bits, PWT and PCD, in the page table entry or page directory entry. If paging is disabled or for cycles that are not paged, PWT and PCD reflect the state of the PWT and PCD bits in control register 3. PWT and PCD have the same timing as the cycle definition pins (M/IO#, D/C# and W/R#). PWT and PCD are active HIGH and are not driven during bus hold. PCD is masked by the cache disable bit (CD) in Control Register 0.
NUMERIC ERROR REPORTING		
FERR#	O	The *floating point error* pin is driven active when a floating point error occurs. FERR# is similar to the ERROR# pin on the 387™ math coprocessor. FERR# is included for compatibility with systems using DOS type floating point error reporting. FERR# will not go active if FP errors are masked in FPU register. FERR# is active LOW, and is not floated during bus hold.
IGNNE#	I	When the *ignore numeric error* pin is asserted the 486 microprocessor will ignore a numeric error and continue executing non-control floating point instructions, but FERR# will still be activated by the i486. When IGNNE# is deasserted the 486 microprocessor will freeze on a non-control floating point instruction, if a previous floating point instruction caused an error. IGNNE# has no effect when the NE bit in control register 0 is set. IGNNE# is active LOW and is provided with a small internal pullup resistor. IGNNE# is asynchronous but setup and hold times t_{20} and t_{21} must be met to insure recognition on any specific clock.
BUS SIZE CONTROL		
BS16# BS8#	I I	The *bus size 16* and *bus size 8* pins (bus sizing pins) cause the 486 microprocessor to run multiple bus cycles to complete a request from devices that cannot provide or accept 32 bits of data in a single cycle. The bus sizing pins are sampled every clock. The state of these pins in the clock before ready is used by the 486 microprocessor to determine the bus size. These signals are active LOW and are provided with internal pullup resistors. These inputs must satisfy setup and hold times t_{14} and t_{15} for proper operation.

QUICK PIN REFERENCE (Continued)

Symbol	Type	Name and Function
ADDRESS MASK		
A20M#	I	When the *address bit 20 mask* pin is asserted, the 486 microprocessor masks physical address bit 20 (A20) before performing a lookup to the internal cache or driving a memory cycle on the bus. A20M# emulates the address wraparound at one Mbyte which occurs on the 8086. A20M# is active LOW and should be asserted only when the processor is in real mode. This pin is asynchronous but should meet setup and hold times t_{20} and t_{21} for recognition in any specific clock. For proper operation, A20M# should be sampled high at the falling edge of RESET.

Table 1.1. Output Pins

Name	Active Level	When Floated
BREQ	HIGH	
HLDA	HIGH	
BE0#-BE3#	LOW	Bus Hold
PWT, PCD	HIGH	Bus Hold
W/R#, D/C#, M/IO#	HIGH	Bus Hold
LOCK#	LOW	Bus Hold
PLOCK#	LOW	Bus Hold
ADS#	LOW	Bus Hold
BLAST#	LOW	Bus Hold
PCHK#	LOW	
FERR#	LOW	
A2-A3	HIGH	Bus, Address Hold

Table 1.2. Input Pins

Name	Active Level	Synchronous/ Asynchronous
CLK		
RESET	HIGH	Asynchronous
HOLD	HIGH	Synchronous
AHOLD	HIGH	Synchronous
EADS#	LOW	Synchronous
BOFF#	LOW	Synchronous
FLUSH#	LOW	Asynchronous
A20M#	LOW	Asynchronous
BS16#, BS8#	LOW	Synchronous
KEN#	LOW	Synchronous
RDY#	LOW	Synchronous
BRDY#	LOW	Synchronous
INTR	HIGH	Asynchronous
NMI	HIGH	Asynchronous
IGNNE#	LOW	Asynchronous

Table 1.3. Input/Output Pins

Name	Active Level	When Floated
D0-D31	HIGH	Bus Hold
DP0-DP3	HIGH	Bus Hold
A4-A31	HIGH	Bus, Address Hold

Table 1.4 Component and Revision ID

i486™ CPU Stepping Name	Component ID	Revision ID
B3	04	01
B4	04	01
B5	04	01
B6	04	01
C0	04	02

5

2.0 ARCHITECTURAL OVERVIEW

The 486 microprocessor is a 32-bit architecture with on-chip memory management, floating point and cache memory units.

The 486 microprocessor contains all the features of the 386™ microprocessor with enhancements to increase performance. The instruction set includes the complete 386 microprocessor instruction set along with extensions to serve new applications. The on-chip memory management unit (MMU) is completely compatible with the 386 microprocessor MMU. The 486 microprocessor brings the 387™ math coprocessor on-chip. All software written for the 386 microprocessor, 387 math coprocessor and previous members of the 86/87 architectural family will run on the 486 microprocessor without any modifications.

Several enhancements have been added to the 486 microprocessor to increase performance. On-chip cache memory allows frequently used data and code to be stored on-chip reducing accesses to the external bus. RISC design techniques have been used to reduce instruction cycle times. A burst bus feature enables fast cache fills. All of these features, combined, lead to performance greater than twice that of a 386 microprocessor.

The memory management unit (MMU) consists of a segmentation unit and a paging unit. Segmentation allows management of the logical address space by providing easy data and code relocatibility and efficient sharing of global resources. The paging mechanism operates beneath segmentation and is transparent to the segmentation process. Paging is optional and can be disabled by system software. Each segment can be divided into one or more 4 Kbyte segments. To implement a virtual memory system, the 486 microprocessor supports full restartability for all page and segment faults.

Memory is organized into one or more variable length segments, each up to four gigabytes (2^{32} bytes) in size. A segment can have attributes associated with it which include its location, size, type (i.e., stack, code or data), and protection characteristics. Each task on a 486 microprocessor can have a maximum of 16,381 segments, each up to four gigabytes in size. Thus each task has a maximum of 64 terabytes (trillion bytes) of virtual memory.

The segmentation unit provides four-levels of protection for isolating and protecting applications and the operating system from each other. The hardware enforced protection allows the design of systems with a high degree of integrity.

The 486 microprocessor has two modes of operation: Real Address Mode (Real Mode) and Protected Mode Virtual Address Mode (Protected Mode). In Real Mode the 486 microprocessor operates as a very fast 8086. Real Mode is required primarily to set up the processor for Protected Mode operation. Protected Mode provides access to the sophisticated memory management paging and privilege capabilities of the processor.

Within Protected Mode, software can perform a task switch to enter into tasks designated as Virtual 8086 Mode tasks. Each virtual 8086 task behaves with 8086 semantics, allowing 8086 software (an application program or an entire operating system) to execute.

The on-chip floating point unit operates in parallel with the arithmetic and logic unit and provides arithmetic instructions for a variety of numeric data types. It executes numerous built-in transcendental functions (e.g., tangent, sine, cosine, and log functions). The floating point unit fully conforms to the ANSI/IEEE standard 754-1985 for floating point arithmetic.

The on-chip cache is 8 Kbytes in size. It is 4-way set associative and follows a write-through policy. The on-chip cache includes features to provide flexibility in external memory system design. Individual pages can be designated as cacheable or non-cacheable by software or hardware. The cache can also be enabled and disabled by software or hardware.

Finally the 486 microprocessor has features to facilitate high performance hardware designs. The 1X clock eases high frequency board level designs. The burst bus feature enables fast cache fills. These features are described beginning in Section 6.

2.1 Register Set

The 486 microprocessor register set includes all the registers contained in the 386 microprocessor and the 387 math coprocessor. The register set can be split into the following categories:

- Base Architecture Registers
  - General Purpose Registers
  - Instruction Pointer
  - Flags Register
  - Segment Registers
- Systems Level Registers
  - Control Registers
  - System Address Registers

Floating Point Registers
- Data Registers
- Tag Word
- Status Word
- Instruction and Data Pointers
- Control Word

Debug and Test Registers

The base architecture and floating point registers are accessible by the applications program. The system level registers are only accessible at privilege level 0 and are used by the systems level program. The debug and test registers are also only accessible at privilege level 0.

2.1.1 BASE ARCHITECTURE REGISTERS

Figure 2.1 shows the 486 microprocessor base architecture registers. The contents of these registers are task-specific and are automatically loaded with a new context upon a task switch operation.

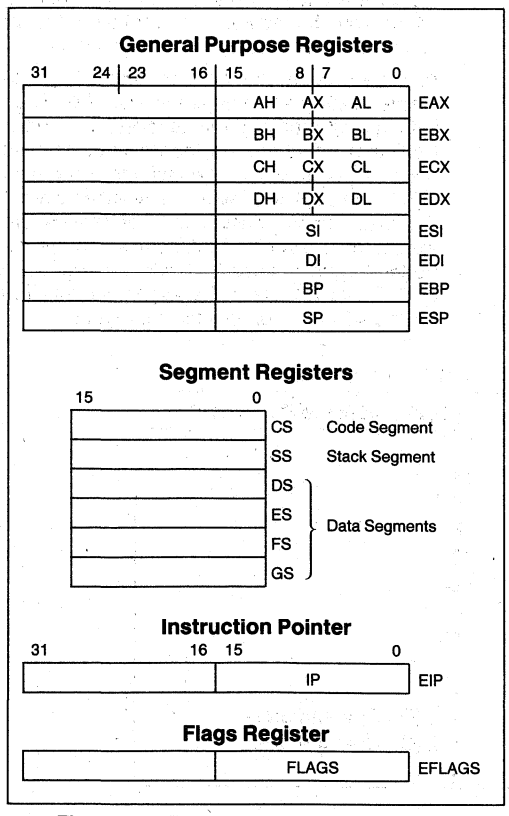


Figure 2.1. Base Architecture Registers

The base architecture includes six directly accessible descriptors, each specifying a segment up to 4 Gbytes in size. The descriptors are indicated by the selector values placed in the 486 microprocessor segment registers. Various selector values can be loaded as a program executes.

The selectors are also task-specific, so the segment registers are automatically loaded with new context upon a task switch operation.

2.1.1.1 General Purpose Registers

The eight 32-bit general purpose registers are shown in Figure 2.1. These registers hold data or address quantities. The general purpose registers can support data operands of 1, 8, 16 and 32 bits, and bit fields of 1 to 32 bits. Address operands of 16 and 32 bits are supported. The 32-bit registers are named EAX, EBX, ECX, EDX, ESI, EDI, EBP and ESP.

The least significant 16 bits of the general purpose registers can be accessed separately by using the 16-bit names of the registers AX, BX, CX, DX, SI, DI, BP and SP. The upper 16 bits of the register are not changed when the lower 16 bits are accessed separately.

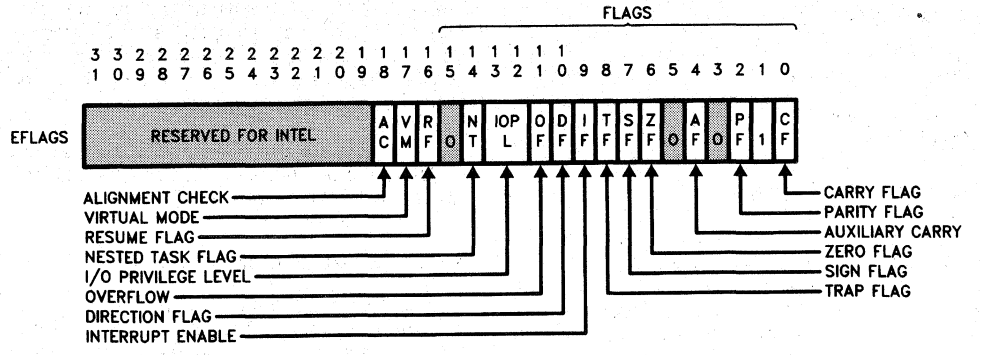
Finally 8-bit operations can individually access the lowest byte (bits 0–7) and the higher byte (bits 8–15) of the general purpose registers AX, BX, CX and DX. The lowest bytes are named AL, BL, CL and DL respectively. The higher bytes are named AH, BH, CH and DH respectively. The individual byte accessibility offers additional flexibility for data operations but is not used for effective address calculation.

2.1.1.2 Instruction Pointer

The instruction pointer, shown in Figure 2.1, is a 32-bit register named EIP. EIP holds the offset of the next instruction to be executed. The offset is always relative to the base of the code segment (CS). The lower 16 bits (bits 0–15) of the EIP contain the 16-bit instruction pointer named IP, which is used for 16-bit addressing.

2.1.1.3 Flags Register

The flags register is a 32-bit register named EFLAGS. The defined bits and bit fields within EFLAGS control certain operations and indicate status of the 486 microprocessor. The lower 16 bits (bit 0–15) of EFLAGS contain the 16-bit register named FLAGS, which is most useful when executing 8086 and 80286 code. EFLAGS is shown in Figure 2.2.



NOTE:
0 indicates Intel Reserved: do not define; see Section 2.1.6.

Figure 2.2. Flags Register

EFLAGS bits 1, 3, 5, 15 and 19–31 are "undefined". When these bits are stored during interrupt processing or with a PUSHF instruction (push flags onto stack), a one is stored in bit 1 and zeros in bits 3, 5, 15 and 19–31.

The EFLAGS register in the 486 microprocessor contains a new bit not previously defined. The new bit, AC, is defined in the upper 16 bits of the register and it enables faults on accesses to misaligned data.

AC (Alignment Check, bit 18)

The AC bit enables the generation of faults if a memory reference is to a misaligned address. Alignment faults are enabled when AC is set to 1. A mis-aligned address is a word access to an odd address, a dword access to an address that is not on a dword boundary, or an 8-byte reference to an address that is not on a 64-bit word boundary. See Section 7.1.6 for more information on operand alignment.

Alignment faults are only generated by programs running at privilege level 3. The AC bit setting is ignored at privilege levels 0, 1 and 2. Note that references to the descriptor tables (for selector loads), or the task state segment (TSS), are implicitly level 0 references even if the instructions causing the references are executed at level 3. Alignment faults are reported through interrupt 17, with an error code of 0. Table 2.1 gives the alignment required for the 486 microprocessor data types.

Table 2.1. Data Type Alignment Requirements

Memory Access	Alignment (Byte Boundary)
Word	2
Dword	4
Single Precision Real	4
Double Precision Real	8
Extended Precision Real	8
Selector	2
48-Bit Segmented Pointer	4
32-Bit Flat Pointer	4
32-Bit Segmented Pointer	2
48-Bit "Pseudo-Descriptor"	4
FSTENV/FLDENV Save Area	4/2 (On Operand Size)
FSAVE/FRSTOR Save Area	4/2 (On Operand Size)
Bit String	4

IMPLEMENTATION NOTE:
Several instructions on the 486 microprocessor generate misaligned references, even if their memory address is aligned. For example, on the 486 microprocessor, the SGDT/SIDT (store global/interrupt descriptor table) instruction reads/writes two bytes, and then reads/writes four bytes from a "pseudo-descriptor" at the given address. The 486 microprocessor will generate misaligned references unless the address is on a 2 mod 4 boundary. The FSAVE and FRSTOR instructions (floating point save and restore state) will generate misaligned references for one-half of the register save/restore cycles. The 486 microprocessor will not cause any AC faults if the effective address given in the instruction has the proper alignment.

VM (Virtual 8086 Mode, bit 17)

The VM bit provides Virtual 8086 Mode within Protected Mode. If set while the 486 Microprocessor is in Protected Mode, the 486 Microprocessor will switch to Virtual 8086 operation, handling segment loads as the 8086 does, but generating exception 13 faults on privileged opcodes. The VM bit can be set only in Protected Mode, by the IRET instruction (if current privilege level = 0) and by task switches at any privilege level. The VM bit is unaffected by POPF. PUSHF always pushes a 0 in this bit, even if executing in Virtual 8086 Mode. The EFLAGS image pushed during interrupt processing or saved during task switches will contain a 1 in this bit if the interrupted code was executing as a Virtual 8086 Task.

RF (Resume Flag, bit 16)

The RF flag is used in conjunction with the debug register breakpoints. It is checked at instruction boundaries before breakpoint processing. When RF is set, it causes any debug fault to be ignored on the next instruction. RF is then automatically reset at the successful completion of every instruction (no faults are signalled) except the IRET instruction, the POPF instruction, (and JMP, CALL, and INT instructions causing a task switch). These instructions set RF to the value specified by the memory image. For example, at the end of the breakpoint service routine, the IRET instruction can pop an EFLAG image having the RF bit set and resume the program's execution at the breakpoint address without generating another breakpoint fault on the same location.

NT (Nested Task, bit 14)

This flag applies to Protected Mode. NT is set to indicate that the execution of this task is nested within another task. If set, it indicates that the current nested task's Task State Segment (TSS) has a valid back link to the previous task's TSS. This bit is set or reset by control transfers to other tasks. The value of NT in EFLAGS is tested by the IRET instruction to determine whether to do an inter-task return or an intra-task return. A POPF or an IRET instruction **will** affect the setting of this bit according to the image popped, at any privilege level.

IOPL (Input/Output Privilege Level, bits 12-13)

This two-bit field applies to Protected Mode. IOPL indicates the numerically maximum CPL (current privilege level) value permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O Permission Bitmap. It also indicates the maximum CPL value allowing alteration of the IF (INTR Enable Flag) bit when new values are popped into the EFLAG register. POPF and IRET instruction can alter the IOPL field when executed at CPL = 0. Task switches can always alter the IOPL field, when the new flag image is loaded from the incoming task's TSS.

OF (Overflow Flag, bit 11)

OF is set if the operation resulted in a signed overflow. Signed overflow occurs when the operation resulted in carry/borrow **into** the sign bit (high-order bit) of the result but did not result in a carry/borrow **out of** the high-order bit, or vice-versa. For 8-, 16-, 32-bit operations, OF is set according to overflow at bit 7, 15, 31, respectively.

DF (Direction Flag, bit 10)

DF defines whether ESI and/or EDI registers postdecrement or postincrement during the string instructions. Postincrement occurs if DF is reset. Postdecrement occurs if DF is set.

IF (INTR Enable Flag, bit 9)

The IF flag, when set, allows recognition of external interrupts signalled on the INTR pin. When IF is reset, external interrupts signalled on the INTR are not recognized. IOPL indicates the maximum CPL value allowing alteration of the IF bit when new values are popped into EFLAGS or FLAGS.

TF (Trap Enable Flag, bit 8)

TF controls the generation of exception 1 trap when single-stepping through code. When TF is set, the 486 Microprocessor generates an exception 1 trap after the next instruction is executed. When TF is reset, exception 1 traps occur only as a function of the breakpoint addresses loaded into debug registers DR0–DR3.

SF (Sign Flag, bit 7)

SF is set if the high-order bit of the result is set, it is reset otherwise. For 8-, 16-, 32-bit operations, SF reflects the state of bit 7, 15, 31 respectively.

ZF (Zero Flag, bit 6)

ZF is set if all bits of the result are 0. Otherwise it is reset.

AF (Auxiliary Carry Flag, bit 4)

The Auxiliary Flag is used to simplify the addition and subtraction of packed BCD quantities. AF is set if the operation resulted in a carry out of bit 3 (addition) or a borrow into bit 3 (subtraction). Otherwise AF is reset. AF is affected by carry out of, or borrow into bit 3 only, regardless of overall operand length: 8, 16 or 32 bits.

PF (Parity Flags, bit 2)

PF is set if the low-order eight bits of the operation contains an even number of "1's" (even parity). PF is reset if the low-order eight bits have odd parity. PF is a function of only the low-order eight bits, regardless of operand size.

CF (Carry Flag, bit 0)

CF is set if the operation resulted in a carry out of (addition), or a borrow into (subtraction) the high-order bit. Otherwise CF is reset. For 8-, 16- or 32-bit operations, CF is set according to carry/borrow at bit 7, 15 or 31, respectively.

NOTE:
In these descriptions, "set" means "set to 1," and "reset" means "reset to 0."

2.1.1.4 Segment Registers

Six 16-bit segment registers hold segment selector values identifying the currently addressable memory segments. In protected mode, each segment may range in size from one byte up to the entire linear and physical address space of the machine, 4 Gbytes (2^{32} bytes). In real address mode, the maximum segment size is fixed at 64 Kbytes (2^{16} bytes).

The six addressable segments are defined by the segment registers CS, SS, DS, ES, FS and GS. The selector in CS indicates the current code segment; the selector in SS indicates the current stack segment; the selectors in DS, ES, FS and GS indicate the current data segments.

2.1.1.5 Segment Descriptor Cache Registers

The segment descriptor cache registers are not programmer visible, yet it is very useful to understand their content. A programmer invisible descriptor cache register is associated with each programmer-visible segment register, as shown by Figure 2.3. Each descriptor cache register holds a 32-bit base address, a 32-bit segment limit, and the other necessary segment attributes.

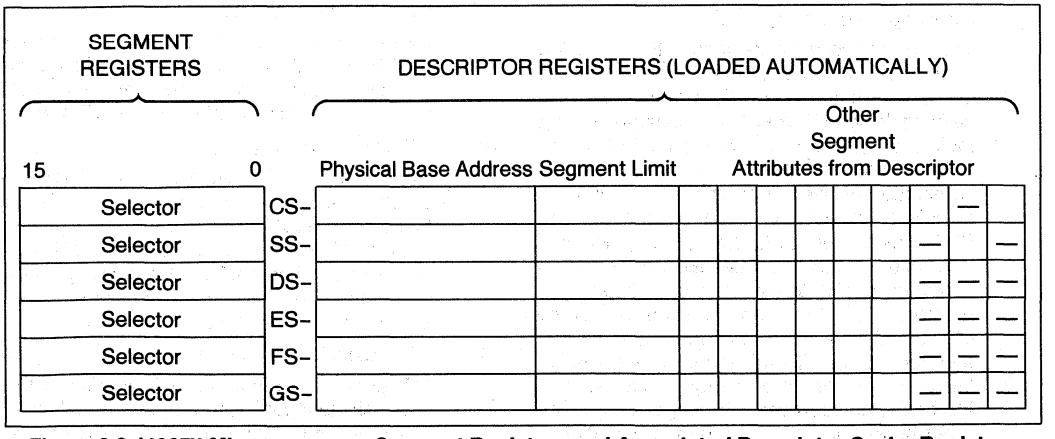


Figure 2.3. i486™ Microprocessor Segment Registers and Associated Descriptor Cache Registers

When a selector value is loaded into a segment register, the associated descriptor cache register is automatically updated with the correct information. In Real Address Mode, only the base address is updated directly (by shifting the selector value four bits to the left), since the segment maximum limit and attributes are fixed in Real Mode. In Protected Mode, the base address, the limit, and the attributes are all updated per the contents of the segment descriptor indexed by the selector.

Whenever a memory reference occurs, the segment descriptor cache register associated with the segment being used is automatically involved with the memory reference. The 32-bit segment base address becomes a component of the linear address calculation, the 32-bit limit is used for the limit-check operation, and the attributes are checked against the type of memory reference requested.

2.1.2 SYSTEM LEVEL REGISTERS

The system level registers, Figure 2.4, control operation of the on-chip cache, the on-chip floating point unit (FPU) and the segmentation and paging mechanisms. These registers are only accessible to programs running at privilege level 0, the highest privilege level.

The system level registers include three control registers and four segmentation base registers. The three control registers are CR0, CR2 and CR3. CR1 is reserved for future Intel processors. The four segmentation base registers are the Global Descriptor Table Register (GDTR), the Interrupt Descriptor Table Register (IDTR), the Local Descriptor Table Register (LDTR) and the Task State Segment Register (TR).

2.1.2.1 Control Registers

Control Register 0 (CR0)

CR0, shown in Figure 2.5, contains 10 bits for control and status purposes. Five of the bits defined in the 486 microprocessor's CR0 are newly defined. The new bits are CD, NW, AM, WP and NE. The function of the bits in CR0 can be categorized as follows:

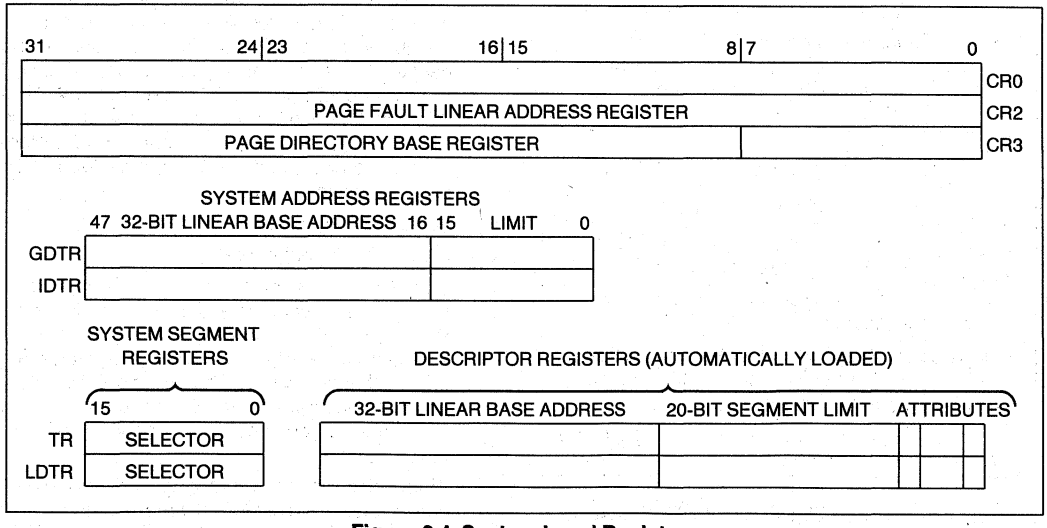


Figure 2.4. System Level Registers

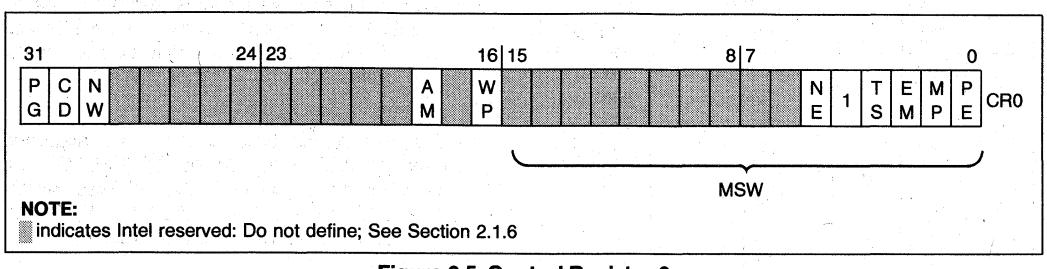


Figure 2.5. Control Register 0

486 Microprocessor Operating Modes: PG, PE (Table 2.2)

On-Chip Cache Control Modes: CD, NW (Table 2.3)

On-Floating Point Unit Control: TS, EM, MP, NE (Table 2.4)

Alignment Check Control: AM

Supervisor Write Protect: WP

Table 2.2. Processor Operating Modes

PG	PE	Mode
0	0	REAL Mode. Exact 8086 semantics, with 32-bit extensions available with prefixes.
0	1	Protected Mode. Exact 80286 semantics, plus 32-bit extensions through both prefixes and "default" prefix setting associated with code segment descriptors. Also, a sub-mode is defined to support a virtual 8086 within the context of the extended 80286 protection model.
1	0	UNDEFINED. Loading CR0 with this combination of PG and PE bits will raise a GP fault with error code 0.
1	1	Paged Protected Mode. All the facilities of Protected mode, with paging enabled underneath segmentation.

Table 2.3. On-Chip Cache Control Modes

CD	NW	Operating Mode
1	1	Cache fills disabled, write-through and invalidates disabled.
1	0	Cache fills disabled, write-through and invalidates enabled.
0	1	INVALID. If CR0 is loaded with this configuration of bits, a GP fault with error code is raised.
0	0	Cache fills enabled, write-through and invalidates enabled.

Table 2.4. On-Chip Floating Point Unit Control

CR0 BIT			Instruction Type	
EM	TS	MP	Floating-Point	Wait
0	0	0	Execute	Execute
0	0	1	Execute	Execute
0	1	0	Trap 7	Execute
0	1	1	Trap 7	Trap 7
1	0	0	Trap 7	Execute
1	0	1	Trap 7	Execute
1	1	0	Trap 7	Execute
1	1	1	Trap 7	Trap 7

The low-order 16 bits of CR0 are also known as the Machine Status Word (MSW), for compatibility with the 80286 protected mode. LMSW and SMSW (load and store MSW) instructions are taken as special aliases of the load and store CR0 operations, where only the low-order 16 bits of CR0 are involved. The LMSW and SMSW instructions in the 486 microprocessor work in an identical fashion to the LMSW and SMSW instructions in the 80286 (i.e., they only operate on the low-order 16 bits of CR0 and ignores the new bits). New 486 microprocessor operating systems should use the MOV CR0, Reg instruction.

The defined CR0 bits are described below.

PG (Paging Enable, bit 31)

The PG bit is used to indicate whether paging is enabled (PG=1) or disabled (PG=0). See Table 2.2.

CD (Cache Disable, bit 30)

The CD bit is used to enable the on-chip cache. When CD=1, the cache will not be filled on cache misses. When CD=0, cache fills may be performed on misses. See Table 2.3.

The state of the CD bit, the cache enable input pin (KEN#), and the relevant page cache disable (PCD) bit determine if a line read in response to a cache miss will be installed in the cache. A line is installed in the cache only if CD=0 and KEN# and PCD are both zero. The relevant PCD bit comes from either the page table entry, page directory entry or control register 3. Refer to Section 5.6 for more details on page cacheability.

CD is set to one after RESET.

NW (Not Write-Through, bit 29)

The NW bit enables on-chip cache write-throughs and write-invalidate cycles (NW=0). When NW=0, all writes, including cache hits, are sent out to the pins. Invalidate cycles are enabled when NW=0. During an invalidate cycle a line will be removed from the cache if the invalidate address hits in the cache. See Table 2.3.

When NW=1, write-throughs and write-invalidate cycles are disabled. A write will not be sent to the pins if the write hits in the cache. With NW=1 the only write cycles that reach the external bus are cache misses. Write hits with NW=1 will never update main memory. Invalidate cycles are ignored when NW=1.

AM (Alignment Mask, bit 18)

The AM bit controls whether the alignment check (AC) bit in the flag register (EFLAGS) can allow an alignment fault. AM=0 disables the AC bit. AM=1 enables the AC bit. AM=0 is the 386 microprocessor compatible mode.

386 microprocessor software may load incorrect data into the AC bit in the EFLAGS register. Setting AM=0 will prevent AC faults from occurring before the 486 microprocessor has created the AC interrupt service routine.

WP (Write Protect, bit 16)

WP protects read-only pages from supervisor write access. The 386 microprocessor allows a read-only page to be written from privilege levels 0–2. The 486 microprocessor is compatible with the 386 microprocessor when WP=0. WP=1 forces a fault on a write to a read-only page from any privilege level. Operating systems with Copy-on-Write features can be supported with the WP bit. Refer to Section 4.5.3 for further details on use of the WP bit.

NE (Numerics Exception, bit 5)

The NE bit controls whether unmasked floating point exceptions (UFPE) are handled through interrupt vector 16 (NE=1) or through an external interrupt (NE=0). NE=0 (default at reset) supports the DOS operating system error reporting scheme from the 8087, 80287 and 387 math coprocessor. In DOS systems, math coprocessor errors are reported via external interrupt vector 13. DOS uses interrupt vector 16 for an operating system call. Refer to Sections 6.2.13 and 7.2.14 for more information on floating point error reporting.

For any UFPE the floating point error output pin (FERR#) will be driven active.

For NE=0, the 486 microprocessor works in conjunction with the ignore numeric error input (IGNNE#) and the FERR# output pins. When a UFPE occurs and the IGNNE# input is inactive, the 486 microprocessor freezes immediately before executing the next floating point instruction. An external interrupt controller will supply an interrupt vector when FERR# is driven active. The UFPE is ignored if IGNNE# is active and floating point execution continues.

NOTE:

The freeze does not take place if the next instruction is one of the control instructions FNCLEX, FNINIT, FNSAVE, FNSTENV, FNSTCW, FNSTSW, FNSTSW AX, FNENI, FNDISI and FNSETPM. The freeze does occur if the next instruction is WAIT.

For NE=1, any UFPE will result in a software interrupt 16, immediately before executing the next non-control floating point or WAIT instruction. The ignore numeric error input (IGNNE#) signal will be ignored.

TS (Task Switched, bit 3)

The TS bit is set whenever a task switch operation is performed. Execution of a floating point instruction with TS=1 will cause a device not available (DNA) fault (trap vector 7). If TS=1 and MP=1 (monitor coprocessor in CR0) a WAIT instruction will cause a DNA fault. See Table 2.4.

EM (Emulate Coprocessor, bit 2)

The EM bit determines whether floating point instructions are trapped (EM=1) or executed. If EM=1, all floating point instructions will cause fault 7.

NOTE:

WAIT instructions are not affected by the state of EM. See Table 2.4.

MP (Monitor Coprocessor, bit 1)

The MP bit is used in conjunction with the TS bit to determine if WAIT instructions should trap. If MP=1 and TS=1, WAIT instructions cause fault 7. Refer to Table 2.4. The TS bit is set to 1 on task switches by the 486 microprocessor. Floating point instructions are not affected by the state of the MP bit. It is recommended that the MP bit be set to one for the normal operation of the 486 microprocessor.

PE (Protection Enable, bit 0)

The PE bit enables the segment based protection mechanism. If PE=1 protection is enabled. When PE=0 the 486 microprocessor operates in REAL mode, with segment based protection disabled, and addresses formed as in an 8086. Refer to Table 2.2.

All new CR0 bits added to the 386 and 486 microprocessors, except for ET and NE, are upward compatible with the 80286 because they are in register bits not defined in the 80286. For strict compatibility with the 80286, the load machine status word (LMSW) instruction is defined to not change the ET or NE bits.

Control Register 1 (CR1)

CR1 is reserved for use in future Intel microprocessors.

Control Register 2 (CR2)

CR2, shown in Figure 2.6, holds the 32-bit linear address that caused the last page fault detected. The error code pushed onto the page fault handler's stack when it is invoked provides additional status information on this page fault.

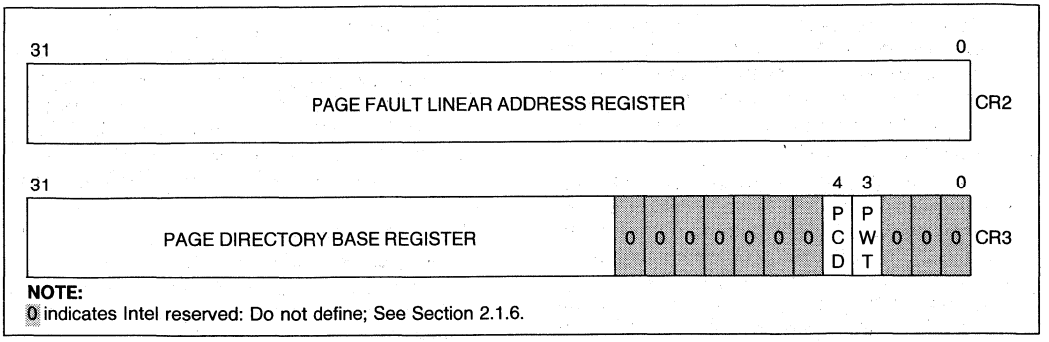


Figure 2.6. Control Registers 2 and 3

Control Register 3 (CR3)

CR3, shown in Figure 2.6, contains the physical base address of the page directory table. The 486 microprocessor page directory is always page aligned (4 Kbyte-aligned). This alignment is enforced by only storing bits 20–31 in CR3.

In the 486 microprocessor CR3 contains two new bits, page write-through (PWT) (bit 3) and page cache disable (PCD) (bit 4). The page table entry (PTE) and page directory entry (PDE) also contain PWT and PCD bits. PWT and PCD control page cacheability. When a page is accessed in external memory, the state of PWT and PCD are driven out on the PWT and PCD pins. The source of PWT and PCD can be CR3, the PTE or the PDE. PWT and PCD are sourced from CR3 when the PDE is being updated. When paging is disabled (PG = 0 in CR0), PCD and PWT are assumed to be 0, regardless of their state in CR3.

A task switch through a task state segment (TSS) which changes the values in CR3, or an explicit load into CR3 with any value, will invalidate all cached page table entries in the translation lookaside buffer (TLB).

The page directory base address in CR3 is a physical address. The page directory can be paged out while its associated task is suspended, but the operating system must ensure that the page directory is resident in physical memory before the task is dispatched. The entry in the TSS for CR3 has a physical address, with no provision for a present bit. This means that the page directory for a task must be resident in physical memory. The CR3 image in a TSS must point to this area, before the task can be dispatched through its TSS.

2.1.2.2 System Address Registers

Four special registers are defined to reference the tables or segments supported by the 80286, 386 and 486 microprocessor protection model. These tables or segments are:

GDT (Global Descriptor Table)
IDT (Interrupt Descriptor Table)
LDT (Local Descriptor Table)
TSS (Task State Segment)

The addresses of these tables and segments are stored in special registers, the System Address and System Segment Registers, illustrated in Figure 2.4. These registers are named GDTR, IDTR, LDTR and TR respectively. Section 4, Protected Mode Architecture, describes the use of these registers.

System Address Registers: GDTR and IDTR

The GDTR and IDTR hold the 32-bit linear base address and 16-bit limit of the GDT and IDT, respectively.

Since the GDT and IDT segments are global to all tasks in the system, the GDT and IDT are defined by 32-bit linear addresses (subject to page translation if paging is enabled) and 16-bit limit values.

System Segment Registers: LDTR and TR

The LDTR and TR hold the 16-bit selector for the LDT descriptor and the TSS descriptor, respectively.

Since the LDT and TSS segments are task specific segments, the LDT and TSS are defined by selector values stored in the system segment registers.

NOTE:
A programmer-invisible segment descriptor register is associated with each system segment register.

2.1.3 FLOATING POINT REGISTERS

Figure 2.7 shows the floating point register set. The on-chip FPU contains eight data registers, a tag word, a control register, a status register, an instruction pointer and a data pointer.

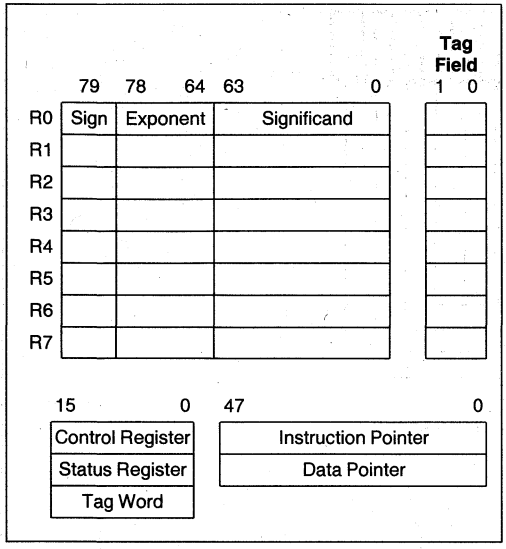


Figure 2.7. Floating Point Registers

The operation of the 486 microprocessor's on-chip floating point unit is exactly the same as the 387 math coprocessor. Software written for the 387 math coprocessor will run on the on-chip floating point unit (FPU) without any modifications.

2.1.3.1 Data Registers

Floating point computations use the 486 microprocessor's FPU data registers. These eight 80-bit registers provide the equivalent capacity of twenty 32-bit registers. Each of the eight data registers is divided into "fields" corresponding to the FPU's extended-precision data type.

The FPU's register set can be accessed either as a stack, with instructions operating on the top one or two stack elements, or as a fixed register set, with instructions operating on explicitly designated registers. The TOP field in the status word identifies the current top-of-stack register. A "push" operation decrements TOP by one and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments TOP by one. Like other 486 microprocessor stacks in memory, the FPU register stack grows "down" toward lower-addressed registers.

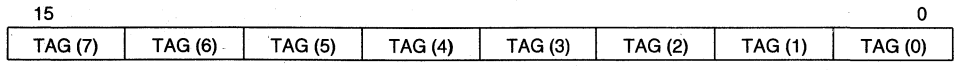
Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the TOP of the stack. These instructions implicitly address the register at which TOP points. Other instructions allow the programmer to explicitly specify which register to use. This explicit register addressing is also relative to TOP.

2.1.3.2 Tag Word

The tag word marks the content of each numeric data register, as shown in Figure 2.8. Each two-bit tag represents one of the eight data registers. The principal function of the tag word is to optimize the FPUs performance and stack handling by making it possible to distinguish between empty and nonempty register locations. It also enables exception handlers to check the contents of a stack location without the need to perform complex decoding of the actual data.

2.1.3.3 Status Word

The 16-bit status word reflects the overall state of the FPU. The status word is shown in Figure 2.9 and is located in the status register.



NOTE:
The index i of tag(i) is **not** top-relative. A program typically uses the "top" field of Status Word to determine which tag(i) field refers to logical top of stack.
TAG VALUES:
00 = Valid
01 = Zero
10 = QNaN, SNaN, Infinity, Denormal and Unsupported Formats
11 = Empty

Figure 2.8. FPU Tag Word

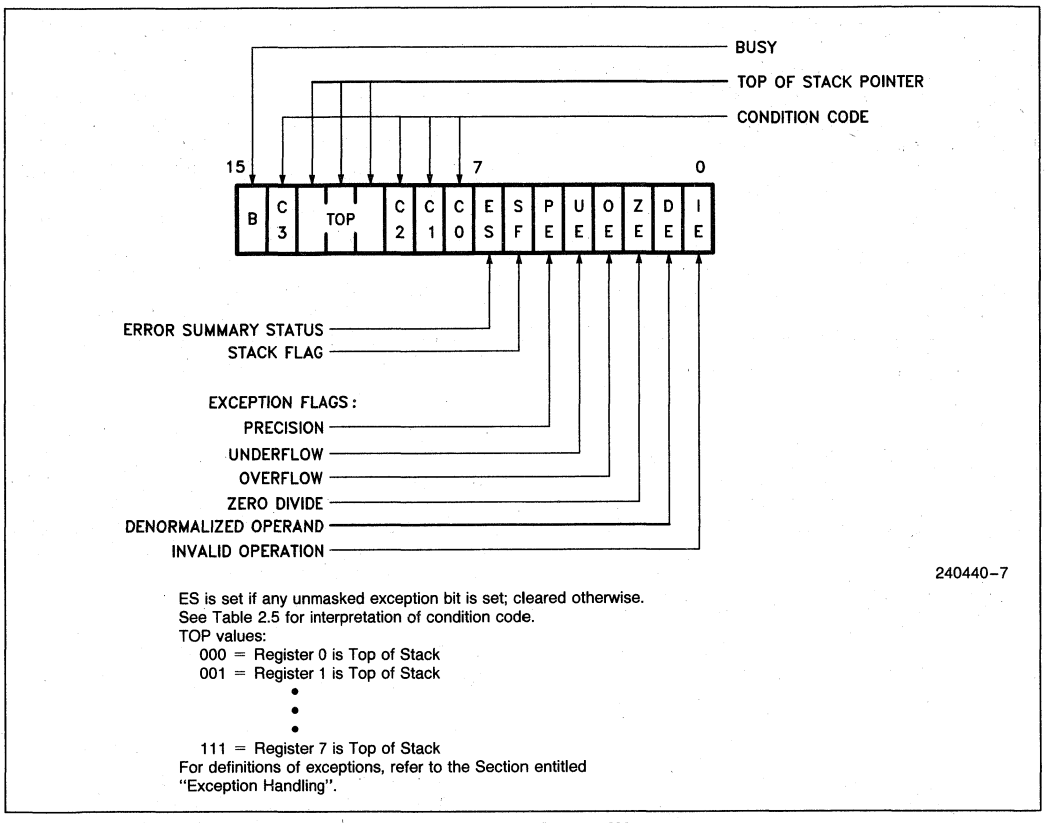


Figure 2.9. FPU Status Word

The B bit (Busy, bit 15) is included for 8087 compatibility. The B bit reflects the contents of the ES bit (bit 7 of the status word).

Bits 13–11 (TOP) point to the FPU register that is the current top-of-stack.

The four numeric condition code bits, C0–C3, are similar to the flags in EFLAGS. Instructions that perform arithmetic operations update C0–C3 to reflect the outcome. The effects of these instructions on the condition codes are summarized in Tables 2.5 through 2.8.

Table 2.5. FPU Condition Code Interpretation

Instruction	C0 (S)	C3 (Z)	C1 (A)	C2 (C)
FPREM, FPREM1 (see Table 2.3)	Three least significant bits of quotient: Q2	Q0	Q1 or O/U#	Reduction: 0 = complete, 1 = incomplete
FCOM, FCOMP, FCOMPP, FTST, FUCOM, FUCOMP, FUCOMPP, FICOM, FICOMP	Result of comparison (see Table 2.7)		Zero or O/U#	Operand is not comparable (Table 2.7)
FXAM	Operand class (see Table 2.8)		Sign or O/U#	Operand class (Table 2.8)
FCHS, FABS, FXCH, FINCTOP, FDECTOP, Constant loads, FXTRACT, FLD, FILD, FBLD, FSTP (ext real)	UNDEFINED		Zero or O/U#	UNDEFINED
FIST, FBSTP, FRNDINT, FST, FSTP, FADD, FMUL, FDIV, FDIVR, FSUB, FSUBR, FSCALE, FSQRT, FPATAN, F2XM1, FYL2X, FYL2XP1	UNDEFINED		Roundup or O/U#	UNDEFINED
FPTAN, FSIN FCOS, FSINCOS	UNDEFINED		Roundup or O/U#, undefined if C2 = 1	Reduction: 0 = complete, 1 = incomplete
FLDENV, FRSTOR	Each bit loaded from memory			
FINIT	Clears these bits			
FLDCW, FSTENV, FSTCW, FSTSW, FCLEX, FSAVE	UNDEFINED			

O/U#	When both IE and SF bits of status word are set, indicating a stack exception, this bit distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).
Reduction	If FPREM or FPREM1 produces a remainder that is less than the modulus, reduction is complete. When reduction is incomplete the value at the top of the stack is a partial remainder, which can be used as input to further reduction. For FPTAN, FSIN, FCOS, and FSINCOS, the reduction bit is set if the operand at the top of the stack is too large. In this case the original operand remains at the top of the stack.
Roundup	When the PE bit of the status word is set, this bit indicates whether the last rounding in the instruction was upward.
UNDEFINED	Do not rely on finding any specific value in these bits.

Table 2.6. Condition Code Interpretation after FPREM and FPREM1 Instructions

Condition Code				Interpretation after FPREM and FPREM1	
C2	C3	C1	C0		
1	X	X	X	Incomplete Reduction: further interaction required for complete reduction	
	Q1	Q0	Q2	Q MOD8	
0	0	0	0	0	Complete Reduction: C0, C3, C1 contain three least significant bits of quotient
	0	1	0	1	
	1	0	0	2	
	1	1	0	3	
	0	0	1	4	
	0	1	1	5	
	1	0	1	6	
	1	1	1	7	

Table 2.7. Condition Code Resulting from Comparison

Order	C3	C2	C0
TOP > Operand	0	0	0
TOP < Operand	0	0	1
TOP = Operand	1	0	0
Unordered	1	1	1

Table 2.8. Condition Code Defining Operand Class

C3	C2	C1	C0	Value at TOP
0	0	0	0	+ Unsupported
0	0	0	1	+ NaN
0	0	1	0	− Unsupported
0	0	1	1	− NaN
0	1	0	0	+ Normal
0	1	0	1	+ Infinity
0	1	1	0	− Normal
0	1	1	1	− Infinity
1	0	0	0	+ 0
1	0	0	1	+ Empty
1	0	1	0	− 0
1	0	1	1	− Empty
1	1	0	0	+ Denormal
1	1	1	0	− Denormal

Bit 7 is the error summary (ES) status bit. The ES bit is set if any unmasked exception bit (bits 0–5 in the status word) is set; ES is clear otherwise. The FERR# (floating point error) signal is asserted when ES is set.

Bit 6 is the stack flag (SF). This bit is used to distinguish invalid operations due to stack overflow or underflow. When SF is set, bit 9 (C1) distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).

Table 2.9 shows the six exception flags in bits 0–5 of the status word. Bits 0–5 are set to indicate that the FPU has detected an exception while executing an instruction.

The six exception flags in the status word can be individually masked by mask bits in the FPU control word. Table 2.9 lists the exception conditions, and their causes in order of precedence. Table 2.9 also shows the action taken by the FPU if the corresponding exception flag is masked.

An exception that is not masked by the control word will cause three things to happen: the corresponding exception flag in the status word will be set, the ES bit in the status word will be set and the FERR# output signal will be asserted. When the 486 microprocessor attempts to execute another floating point or WAIT instruction, exception 16 occurs or an external interrupt happens if the NE = 1 in control register 0. The exception condition must be resolved via an interrupt service routine. The FPU saves the address of the floating point instruction that caused the exception and the address of any memory operand required by that instruction in the instruction and data pointers (see Section 2.1.3.4).

Note that when a new value is loaded into the status word by the FLDENV (load environment) or FRSTOR (restore state) instruction, the value of ES (bit 7) and its reflection in the B bit (bit 15) are not derived from the values loaded from memory. The values of ES and B are dependent upon the values of the exception flags in the status word and their corresponding masks in the control word. If ES is set in such a case, the FERR# output of the 486 microprocessor is activated immediately.

2.1.3.4 Instruction and Data Pointers

Because the FPU operates in parallel with the ALU (in the 486 microprocessor the arithmetic and logic unit (ALU) consists of the base architecture registers), any errors detected by the FPU may be reported after the ALU has executed the floating point instruction that caused it. To allow identification of the failing numeric instruction, the 486 microprocessor contains two pointer registers that supply the address of the failing numeric instruction and the address of its numeric memory operand (if appropriate).

5

Table 2.9. FPU Exceptions

Exception	Cause	Default Action (if exception is masked)
Invalid Operation	Operation on a signaling NaN, unsupported format, indeterminate form (0 * ∞, 0/0, (+ ∞) + (− ∞), etc.), or stack overflow/underflow (SF is also set).	Result is a quiet NaN, integer indefinite, or BCD indefinite
Denormalized Operand	At least one of the operands is denormalized, i.e., it has the smallest exponent but a nonzero significand.	Normal processing continues
Zero Divisor	The divisor is zero while the dividend is a noninfinite, nonzero number.	Result is ∞
Overflow	The result is too large in magnitude to fit in the specified format.	Result is largest finite value or ∞
Underflow	The true result is nonzero but too small to be represented in the specified format, and, if underflow exception is masked, denormalization causes loss of accuracy.	Result is denormalized or zero
Inexact Result (Precision)	The true result is not exactly representable in the specified format (e.g., 1/3); the result is rounded according to the rounding mode.	Normal processing continues

The instruction and data pointers are provided for user-written error handlers. These registers are accessed by the FLDENV (load environment), FSTENV (store environment), FSAVE (save state) and FRSTOR (restore state) instructions. Whenever the 486 microprocessor decodes a new floating point instruction, it saves the instruction (including any prefixes that may be present), the address of the operand (if present) and the opcode.

The instruction and data pointers appear in one of four formats depending on the operating mode of the 486 microprocessor (protected mode or real-address mode) and depending on the operand-size attribute in effect (32-bit operand or 16-bit operand). When the 486 microprocessor is in the virtual-86 mode, the real address mode formats are used. The four formats are shown in Figures 2.10–2.13. The floating point instructions FLDENV, FSTENV, FSAVE and FRSTOR are used to transfer these values to and from memory. Note that the value of the data pointer is undefined if the prior floating point instruction did not have a memory operand.

NOTE:
The operand size attribute is the D bit in a segment descriptor.

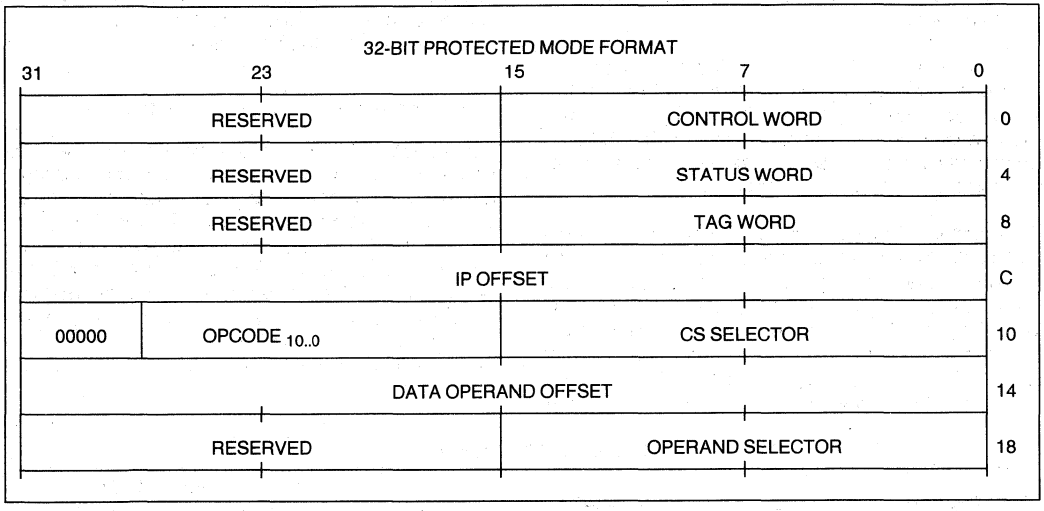


Figure 2.10. Protected Mode FPU Instruction and Data Pointer Image in Memory, 32-Bit Format

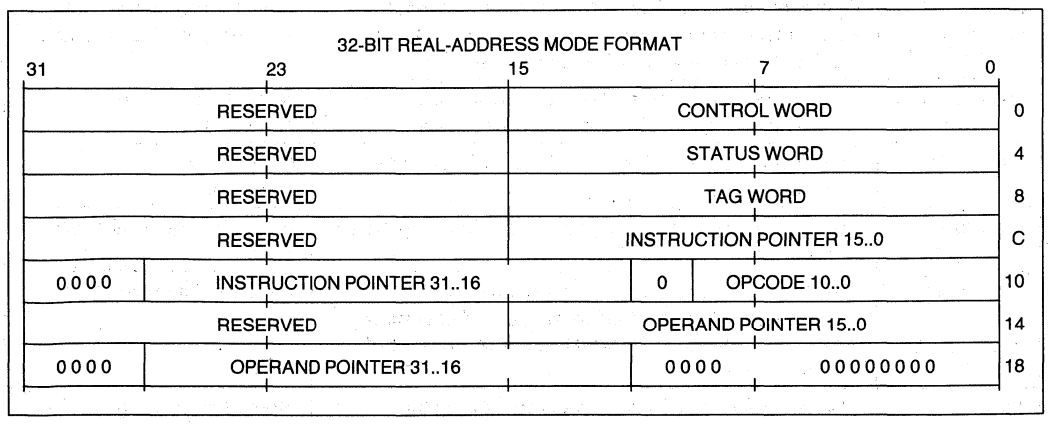


Figure 2.11. Real Mode FPU Instruction and Data Pointer Image in Memory, 32-Bit Format

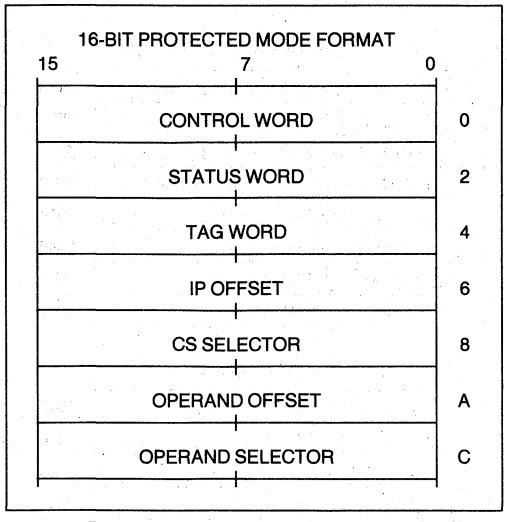


Figure 2.12. Protected Mode FPU Instruction and Data Pointer Image in Memory, 16-Bit Format

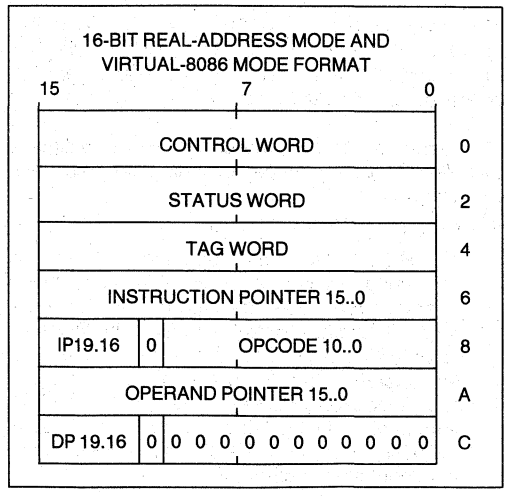


Figure 2.13. Real Mode FPU Instruction and Data Pointer Image in Memory, 16-Bit Format

2.1.3.5 FPU Control Word

The FPU provides several processing options that are selected by loading a control word from memory into the control register. Figure 2.14 shows the format and encoding of fields in the control word.

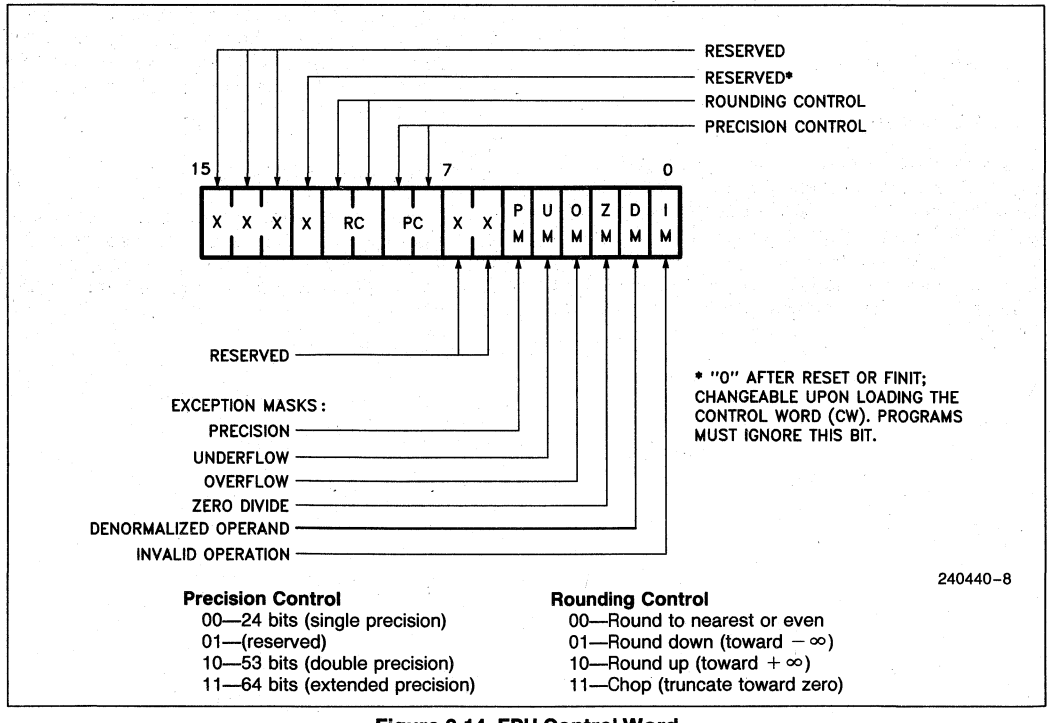


Figure 2.14. FPU Control Word

The low-order byte of the FPU control word configures the FPU error and exception masking. Bits 0–5 of the control word contain individual masks for each of the six exceptions that the FPU recognizes.

The high-order byte of the control word configures the FPU operating mode, including precision and rounding.

RC (Rounding Control, bits 10–11)

The RC bits provide for directed rounding and true chop, as well as the unbiased round to nearest even mode specified in the IEEE standard. Rounding control affects only those instructions that perform rounding at the end of the operation (and thus can generate a precision exception); namely, FST, FSTP, FIST, all arithmetic instructions (except FPREM, FPREM1, FXTRACT, FABS and FCHS), and all transcendental instructions.

PC (Precision Control, bits 8–9)

The PC bits can be used to set the FPU internal operating precision of the significand at less than the default of 64 bits (extended precision). This can be useful in providing compatibility with early generation arithmetic processors of smaller precision. PC affects only the instructions ADD, SUB, DIV, MUL, and SQRT. For all other instructions, either the precision is determined by the opcode or extended precision is used.

2.1.4 DEBUG AND TEST REGISTERS

2.1.4.1 Debug Registers

The six programmer accessible debug registers, Figure 2.15, provide on-chip support for debugging. Debug registers DR0–3 specify the four linear breakpoints. The Debug control register DR7, is used to set the breakpoints and the Debug Status Register, DR6, displays the current state of the breakpoints. The use of the Debug registers is described in Section 9.

Debug Registers

LINEAR BREAKPOINT ADDRESS 0	DR0
LINEAR BREAKPOINT ADDRESS 1	DR1
LINEAR BREAKPOINT ADDRESS 2	DR2
LINEAR BREAKPOINT ADDRESS 3	DR3
Intel Reserved Do Not Define	DR4
Intel Reserved Do Not Define	DR5
BREAKPOINT STATUS	DR6
BREAKPOINT CONTROL	DR7

Test Registers

CACHE TEST DATA	TR3
CACHE TEST STATUS	TR4
CACHE TEST CONTROL	TR5
TLB TEST CONTROL	TR6
TLB TEST STATUS	TR7

TLB = Translation Lookaside Buffer

Figure 2.15

2.1.4.2 Test Registers

The 486 microprocessor contains five test registers. The test registers are shown in Figure 2.15. TR6 and TR7 are used to control the testing of the translation lookaside buffer. TR3, TR4 and TR5 are used for testing the on-chip cache. The use of the test registers is discussed in Section 8.

2.1.5 REGISTER ACCESSIBILITY

There are a few differences regarding the accessibility of the registers in Real and Protected Mode. Table 2.10 summarizes these differences. See Section 4, Protected Mode Architecture, for further details.

Table 2.10. Register Usage

Register	Use in Real Mode		Use in Protected Mode		Use in Virtual 8086 Mode	
	Load	Store	Load	Store	Load	Store
General Registers	Yes	Yes	Yes	Yes	Yes	Yes
Segment Register	Yes	Yes	Yes	Yes	Yes	Yes
Flag Register	Yes	Yes	Yes	Yes	IOPL	IOPL*
Control Registers	Yes	Yes	PL = 0	PL = 0	No	Yes
GDTR	Yes	Yes	PL = 0	Yes	No	Yes
IDTR	Yes	Yes	PL = 0	Yes	No	Yes
LDTR	No	No	PL = 0	Yes	No	No
TR	No	No	PL = 0	Yes	No	No
FPU Data Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Control Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Status Registers	Yes	Yes	Yes	Yes	Yes	Yes
FPU Instruction Pointer	Yes	Yes	Yes	Yes	Yes	Yes
FPU Data Pointer	Yes	Yes	Yes	Yes	Yes	Yes
Debug Registers	Yes	Yes	PL = 0	PL = 0	No	No
Test Registers	Yes	Yes	PL = 0	PL = 0	No	No

NOTES:
PL = 0: The registers can be accessed only when the current privilege level is zero.
*IOPL: The PUSHF and POPF instructions are made I/O Privilege Level sensitive in Virtual 86 Mode.

5

2.1.6 COMPATIBILITY

VERY IMPORTANT NOTE:
COMPATIBILITY WITH FUTURE PROCESSORS

In the preceding register descriptions, note certain 486 Microprocessor register bits are Intel reserved. When reserved bits are called out, treat them as fully undefined. This is essential for your software compatibility with future processors! Follow the guidelines below:

1) **Do not depend on the states of any undefined bits when testing the values of defined register bits. Mask them out when testing.**
2) **Do not depend on the states of any undefined bits when storing them to memory or another register.**
3) **Do not depend on the ability to retain information written into any undefined bits.**
4) **When loading registers always load the undefined bits as zeros.**
5) **However, registers which have been previously stored may be reloaded without masking.**

Depending upon the values of undefined register bits will make your software dependent upon the unspecified 486 Microprocessor handling of these bits. Depending on undefined values risks making your software incompatible with future processors that define usages for the 486 Microprocessor-undefined bits. AVOID ANY SOFTWARE DEPENDENCE UPON THE STATE OF UNDEFINED 486 MICROPROCESSOR REGISTER BITS.

2.2 Instruction Set

The 486 microprocessor instruction set can be divided into 11 categories of operations:

Data Transfer
Arithmetic
Shift/Rotate
String Manipulation
Bit Manipulation
Control Transfer
High Level Language Support
Operating System Support
Processor Control
Floating Point
Floating Point Control

The 486 microprocessor instructions are listed in Section 10. Note that all floating point unit instruction mnemonics begin with an F.

All 486 microprocessor instructions operate on either 0, 1, 2 or 3 operands; where an operand resides in a register, in the instruction itself or in memory. Most zero operand instructions (e.g., CLI, STI) take only one byte. One operand instructions generally are two bytes long. The average instruction is 3.2 bytes long. Since the 486 microprocessor has a 32-byte instruction queue, an average of 10 instructions will be prefetched. The use of two operands permits the following types of common instructions:

Register to Register
Memory to Register
Memory to Memory
Immediate to Register
Register to Memory
Immediate to Memory

The operands can be either 8, 16, or 32 bits long. As a general rule, when executing code written for the 486 or 386 microprocessors (32-bit code), operands are 8 or 32 bits; when executing existing 80286 or 8086 code (16-bit code), operands are 8 or 16 bits. Prefixes can be added to all instructions which override the default length of the operands (i.e., use 32-bit operands for 16-bit code, or 16-bit operands for 32-bit code).

2.3 Memory Organization

Introduction

Memory on the 486 Microprocessor is divided up into 8-bit quantities (bytes), 16-bit quantities (words), and 32-bit quantities (dwords). Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address, the high order byte at the high address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address, the high-order byte at the highest address. The address of a word or dword is the byte address of the low-order byte.

In addition to these basic data types, the 486 Microprocessor supports two larger units of memory: pages and segments. Memory can be divided up into one or more variable length segments, which can be swapped to disk or shared between programs. Memory can also be organized into one or more 4 Kbyte pages. Finally, both segmentation and paging can be combined, gaining the advantages of both systems. The 486 Microprocessor supports both pages and segments in order to provide maximum flexibility to the system designer. Segmentation and paging are complementary. Segmentation is useful for organizing memory in logical modules, and as such is a tool for the application programmer, while pages are useful for the system programmer for managing the physical memory of a system.

2.3.1 ADDRESS SPACES

The 486 Microprocessor has three distinct address spaces: **logical, linear,** and **physical**. A **logical** address (also known as a **virtual** address) consists of a selector and an offset. A selector is the contents of a segment register. An offset is formed by summing all of the addressing components (BASE, INDEX, DISPLACEMENT) discussed in Section 2.5.3 **Memory Addressing Modes** into an effective address. Since each task on the 486 Microprocessor has a maximum of 16K ($2^{14}-1$) selectors, and offsets can be 4 gigabytes, (2^{32} bits) this gives a total of 2^{46} bits or 64 terabytes of **logical** address space per task. The programmer sees this virtual address space.

The segmentation unit translates the **logical** address space into a 32-bit **linear** address space. If the paging unit is not enabled then the 32-bit **linear** address corresponds to the **physical** address. The paging unit translates the **linear** address space into the **physical** address space. The **physical address** is what appears on the address pins.

The primary difference between Real Mode and Protected Mode is how the segmentation unit performs the translation of the **logical** address into the **linear** address. In Real Mode, the segmentation unit shifts the selector left four bits and adds the result to the offset to form the **linear** address. While in Protected Mode every selector has a **linear** base address associated with it. The **linear base** address is stored in one of two operating system tables (i.e., the Local Descriptor Table or Global Descriptor Table). The selector's **linear base** address is added to the offset to form the final **linear** address.

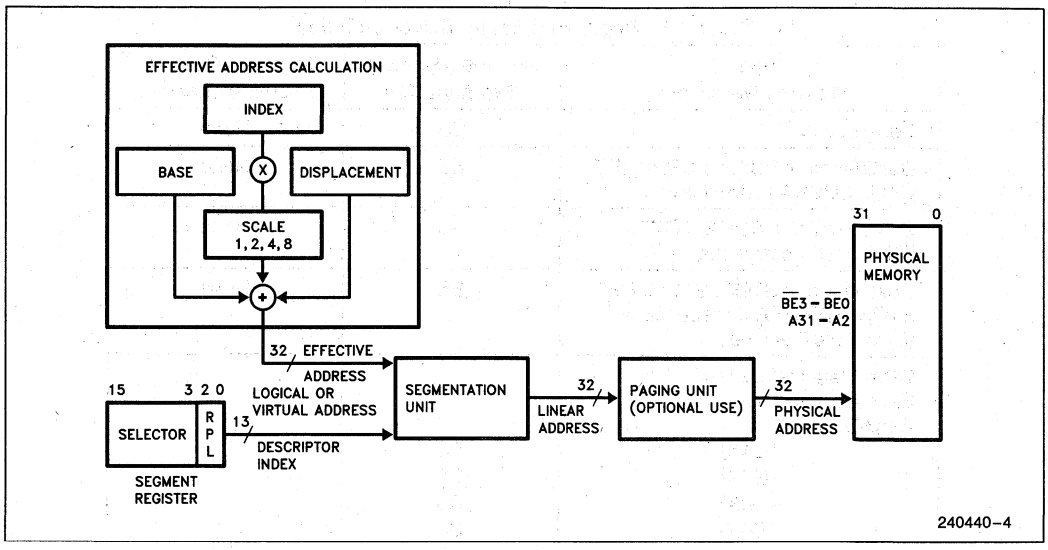


Figure 2.16. Address Translation

Figure 2.16 shows the relationship between the various address spaces.

2.3.2 SEGMENT REGISTER USAGE

The main data structure used to organize memory is the segment. On the 486 Microprocessor, segments are variable sized blocks of linear addresses which have certain attributes associated with them. There are two main types of segments: code and data, the segments are of variable size and can be as small as 1 byte or as large as 4 gigabytes (2^{32} bytes).

In order to provide compact instruction encoding, and increase processor performance, instructions do not need to explicitly specify which segment register is used. A default segment register is automatically chosen according to the rules of Table 2.11 (Segment Register Selection Rules). In general, data references use the selector contained in the DS register; Stack references use the SS register and Instruction fetches use the CS register. The contents of the Instruction Pointer provide the offset. Special segment override prefixes allow the explicit use of a given segment register, and override the implicit rules listed in Table 2.11. The override prefixes also allow the use of the ES, FS and GS segment registers.

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all 6 segments could have the base address set to zero and create a system with a four gigabyte linear address space. This creates a system where the virtual address space is the same as the linear address space. Further details of segmentation are discussed in Section 4.1.

2.4 I/O Space

The 486 Microprocessor has two distinct physical address spaces: Memory and I/O. Generally, peripherals are placed in I/O space although the 486 Microprocessor also supports memory-mapped peripherals. The I/O space consists of 64 Kbytes, it can be divided into 64K 8-bit ports, 32K 16-bit ports, or 16K 32-bit ports, or any combination of ports which add up to less than 64 Kbytes. The 64K I/O address space refers to physical memory rather than linear address since I/O instructions do not go through the segmentation or paging hardware. The M/IO# pin acts as an additional address line thus allowing the system designer to easily determine which address space the processor is accessing.

The I/O ports are accessed via the IN and OUT I/O instructions, with the port address supplied as an immediate 8-bit constant in the instruction or in the DX register. All 8- and 16-bit port addresses are zero extended on the upper address lines. The I/O instructions cause the M/IO# pin to be driven low.

I/O port addresses 00F8H through 00FFH are reserved for use by Intel.

Table 2.11. Segment Register Selection Rules

Type of Memory Reference	Implied (Default) Segment Use	Segment Override Prefixes Possible
Code Fetch	CS	None
Destination of PUSH, PUSHF, INT, CALL, PUSHA Instructions	SS	None
Source of POP, POPA, POPF, IRET, RET instructions	SS	None
Destination of STOS, MOVS, REP STOS, REP MOVS Instructions (DI is Base Register)	ES	None
Other Data References, with Effective Address Using Base Register of:		
[EAX]	DS	All
[EBX]	DS	
[ECX]	DS	
[EDX]	DS	
[ESI]	DS	
[EDI]	DS	
[EBP]	SS	
[ESP]	SS	

2.5 Addressing Modes

2.5.1 ADDRESSING MODES OVERVIEW

The 486 Microprocessor provides a total of 11 addressing modes for instructions to specify operands. The addressing modes are optimized to allow the efficient execution of high level languages such as C and FORTRAN, and they cover the vast majority of data references needed by high-level languages.

2.5.2 REGISTER AND IMMEDIATE MODES

Two of the addressing modes provide for instructions that operate on register or immediate operands:

Register Operand Mode: The operand is located in one of the 8-, 16- or 32-bit general registers.

Immediate Operand Mode: The operand is included in the instruction as part of the opcode.

2.5.3 32-BIT MEMORY ADDRESSING MODES

The remaining 9 modes provide a mechanism for specifying the effective address of an operand. The linear address consists of two components: the segment base address and an effective address. The effective address is calculated by using combinations of the following four address elements:

DISPLACEMENT: An 8-, or 32-bit immediate value, following the instruction.

BASE: The contents of any general purpose register. The base registers are generally used by compilers to point to the start of the local variable area.

INDEX: The contents of any general purpose register except for ESP. The index registers are used to access the elements of an array, or a string of characters.

SCALE: The index register's value can be multiplied by a scale factor, either 1, 2, 4 or 8. Scaled index

mode is especially useful for accessing arrays or structures.

Combinations of these 4 components make up the 9 additional addressing modes. There is no performance penalty for using any of these addressing combinations, since the effective address calculation is pipelined with the execution of other instructions. The one exception is the simultaneous use of Base and Index components which requires one additional clock.

As shown in Figure 2.17, the effective address (EA) of an operand is calculated according to the following formula.

EA = Base Reg + (Index Reg * Scaling) + Displacement

Direct Mode: The operand's offset is contained as part of the instruction as an 8-, 16- or 32-bit displacement.
EXAMPLE: INC Word PTR [500]

Register Indirect Mode: A BASE register contains the address of the operand.
EXAMPLE: MOV [ECX], EDX

Based Mode: A BASE register's contents is added to a DISPLACEMENT to form the operand's offset.
EXAMPLE: MOV ECX, [EAX + 24]

Index Mode: An INDEX register's contents is added to a DISPLACEMENT to form the operand's offset.
EXAMPLE: ADD EAX, TABLE[ESI]

Scaled Index Mode: An INDEX register's contents is multiplied by a scaling factor which is added to a DISPLACEMENT to form the operand's offset.
EXAMPLE: IMUL EBX, TABLE[ESI*4],7

Based Index Mode: The contents of a BASE register is added to the contents of an INDEX register to form the effective address of an operand.
EXAMPLE: MOV EAX, [ESI] [EBX]

Based Scaled Index Mode: The contents of an INDEX register is multiplied by a SCALING factor and the result is added to the contents of a BASE register to obtain the operand's offset.
EXAMPLE: MOV ECX, [EDX*8] [EAX]

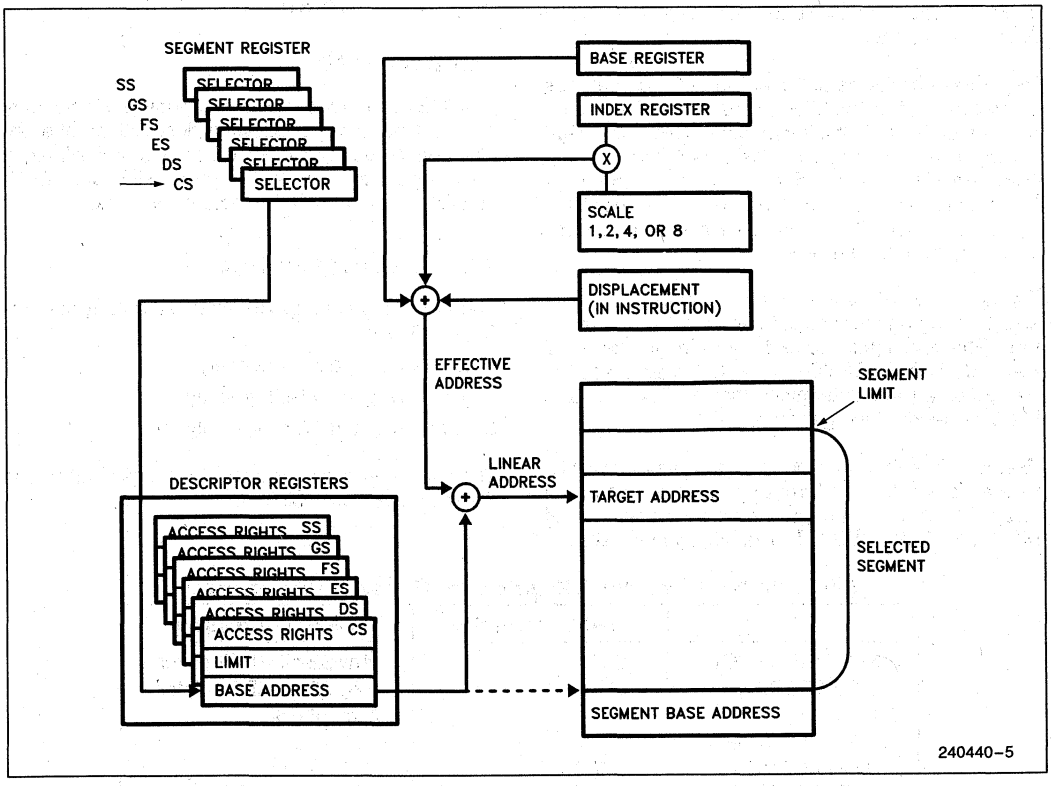


Figure 2.17. Addressing Mode Calculations

Based Index Mode with Displacement: The contents of an INDEX Register and a BASE register's contents and a DISPLACEMENT are all summed together to form the operand offset.
EXAMPLE: ADD EDX, [ESI] [EBP + 00FFFFF0H]

Based Scaled Index Mode with Displacement: The contents of an INDEX register are multiplied by a SCALING factor, the result is added to the contents of a BASE register and a DISPLACEMENT to form the operand's offset.
EXAMPLE: MOV EAX, LOCALTABLE[EDI*4] [EBP + 80]

2.5.4 DIFFERENCES BETWEEN 16- AND 32-BIT ADDRESSES

In order to provide software compatibility with the 80286 and the 8086, the 486 Microprocessor can execute 16-bit instructions in Real and Protected Modes. The processor determines the size of the instructions it is executing by examining the D bit in the CS segment Descriptor. If the D bit is 0 then all operand lengths and effective addresses are assumed to be 16 bits long. If the D bit is 1 then the default length for operands and addresses is 32 bits. In Real Mode the default size for operands and addresses is 16-bits.

Regardless of the default precision of the operands or addresses, the 486 Microprocessor is able to execute either 16- or 32-bit instructions. This is specified via the use of override prefixes. Two prefixes, the **Operand Size Prefix** and the **Address Length Prefix**, override the value of the D bit on an individual instruction basis. These prefixes are automatically added by Intel assemblers.

Example: The processor is executing in Real Mode and the programmer needs to access the EAX registers. The assembler code for this might be MOV EAX, 32-bit MEMORYOP, ASM486 Macro Assembler automatically determines that an Operand Size Prefix is needed and generates it.

Example: The D bit is 0, and the programmer wishes to use Scaled Index addressing mode to access an array. The Address Length Prefix allows the use of MOV DX, TABLE[ESI*2]. The assembler uses an Address Length Prefix since, with D = 0, the default addressing mode is 16-bits.

Example: The D bit is 1, and the program wants to store a 16-bit quantity. The Operand Length Prefix is used to specify only a 16-bit value; MOV MEM16, DX.

The OPERAND LENGTH and Address Length Prefixes can be applied separately or in combination to any instruction. The Address Length Prefix does not allow addresses over 64 Kbytes to be accessed in Real Mode. A memory address which exceeds FFFFH will result in a General Protection Fault. An Address Length Prefix only allows the use of the additional 486 Microprocessor addressing modes.

When executing 32-bit code, the 486 Microprocessor uses either 8-, or 32-bit displacements, and any register can be used as base or index registers. When executing 16-bit code, the displacements are either 8, or 16 bits, and the base and index register conform to the 80286 model. Table 2.12 illustrates the differences.

2.6 Data Formats

2.6.1 DATA TYPES

The 486 microprocessor can support a wide-variety of data types. In the following descriptions, the on-chip floating point unit (FPU) consists of the floating point registers. The central processing unit (CPU) consists of the base architecture registers.

2.6.1.1 Unsigned Data Types

The FPU does not support unsigned data types. Refer to Table 2.13.

Byte: Unsigned 8-bit quantity

Word: Unsigned 16-bit quantity

Dword: Unsigned 32-bit quantity

The least significant bit (LSB) in a byte is bit 0, and the most significant bit is 7.

Table 2.12. BASE and INDEX Registers for 16- and 32-Bit Addresses

	16-Bit Addressing	32-Bit Addressing
BASE REGISTER	BX,BP	Any 32-bit GP Register
INDEX REGISTER	SI,DI	Any 32-bit GP Register Except ESP
SCALE FACTOR	none	1, 2, 4, 8
DISPLACEMENT	0, 8, 16 bits	0, 8, 32 bits

2.6.1.2 Signed Data Types

All signed data types assume 2's complement notation. The signed data types contain two fields, a sign bit and a magnitude. The sign bit is the most significant bit (MSB). The number is negative if the sign bit is 1. If the sign bit is 0, the number is positive. The magnitude field consists of the remaining bits in the number. Refer to Table 2.13.

8-bit Integer: Signed 8-bit quantity

16-bit Integer: Signed 16-bit quantity

32-bit Integer: Signed 32-bit quantity

64-bit Integer: Signed 64-bit quantity

The FPU only supports 16-, 32- and 64-bit integers. The CPU only supports 8-, 16- and 32-bit integers.

2.6.1.3 Floating Point Data Types

Floating point data type in the 486 microprocessor contain three fields, sign, significand and exponent. The sign field is one bit and is the MSB of the floating point number. The number is negative if the sign bit is 1. If the sign bit is 0, the number is positive. The significand gives the significant bits of the number. The exponent field contains the power of 2 needed to scale the significand. Refer to Table 2.13.

Only the FPU supports floating point data types.

Single Precision Real: 23-bit significand and 8-bit exponent. 32 bits total.

Double Precision Real: 52-bit significand and 11-bit exponent. 64 bits total.

Extended Precision Real: 64-bit significand and 15-bit exponent. 80 bits total.

2.6.1.4 BCD Data Types

The 486 microprocessor supports packed and unpacked binary coded decimal (BCD) data types. A packed BCD data type contains two digits per byte, the lower digit is in bits 0–3 and the upper digit in bits 4–7. An unpacked BCD data type contains 1 digit per byte stored in bits 0–3.

The CPU supports 8-bit packed and unpacked BCD data types. The FPU only supports 80-bit packed BCD data types. Refer to Table 2.13.

2.6.1.5 String Data Types

A string data type is a contiguous sequence of bits, bytes, words or dwords. A string may contain between 1 byte and 4 Gbytes. Refer to Table 2.14.

String data types are only supported by the CPU.

Byte String: Contiguous sequence of bytes.

Word String: Contiguous sequence of words.

Dword String: Contiguous sequence of dwords.

Bit String: A set of contiguous bits. In the 486 microprocessor bit strings can be up to 4 gigabits long.

2.6.1.6 ASCII Data Types

The 486 microprocessor supports ASCII (American Standard Code for Information Interchange) strings and can perform arithmetic operations (such as addition and division) on ASCIi data. The CPU can only operate on ASCII data. Refer to Table 2.14.

Table 2.13. i486™ Microprocessor Data Types

Data Format	Supported by Base Registers	Supported by FPU	Range	Precision
Byte	X		0–255	8 bits
Word	X		0–64K	16 bits
Dword	X		0–4G	32 bits
8-Bit Integer	X		10^2	8 bits
16-Bit Integer	X	X	10^4	16 bits
32-Bit Integer	X	X	10^9	32 bits
64-Bit Integer		X	10^{19}	64 bits
8-Bit Unpacked BCD	X		0–9	1 Digit
8-Bit Packed BCD	X		0–9	2 Digits
80-Bit Packed BCD		X	$\pm 10^{\pm 18}$	18 Digits
Single Precision Real		X	$\pm 10^{\pm 38}$	24 Bits
Double Precision Real		X	$\pm 10^{\pm 308}$	53 Bits
Extended Precision Real		X	$\pm 10^{\pm 4932}$	64 Bits

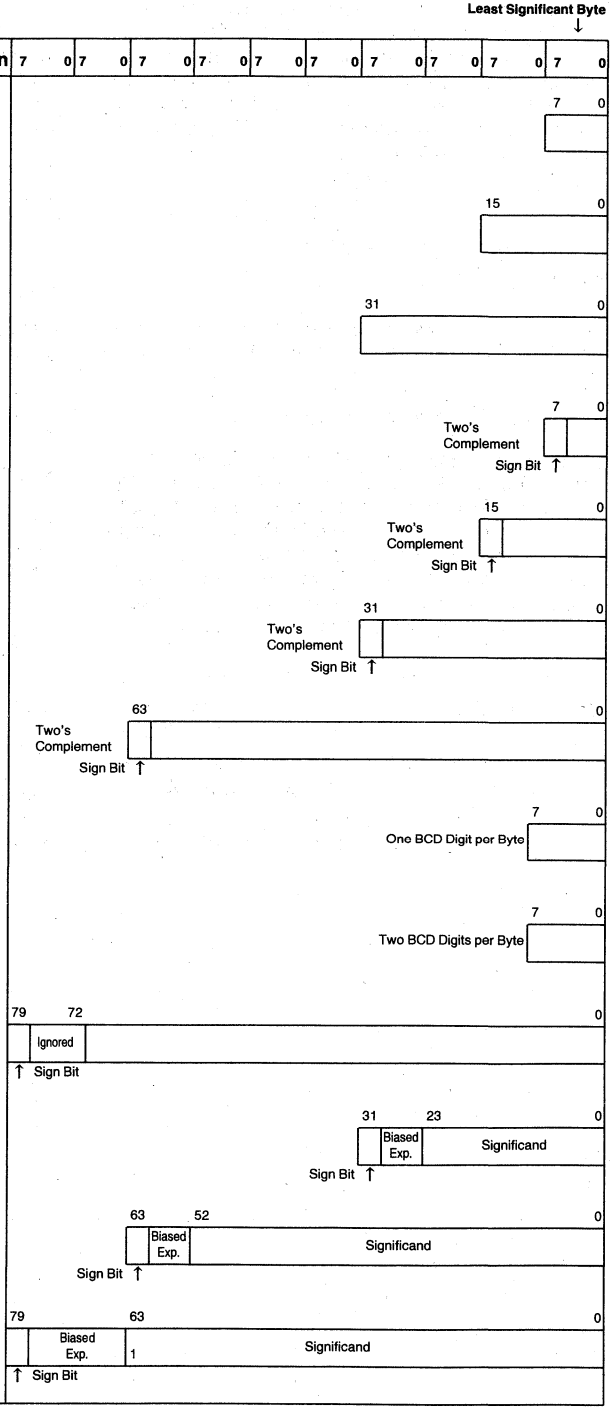
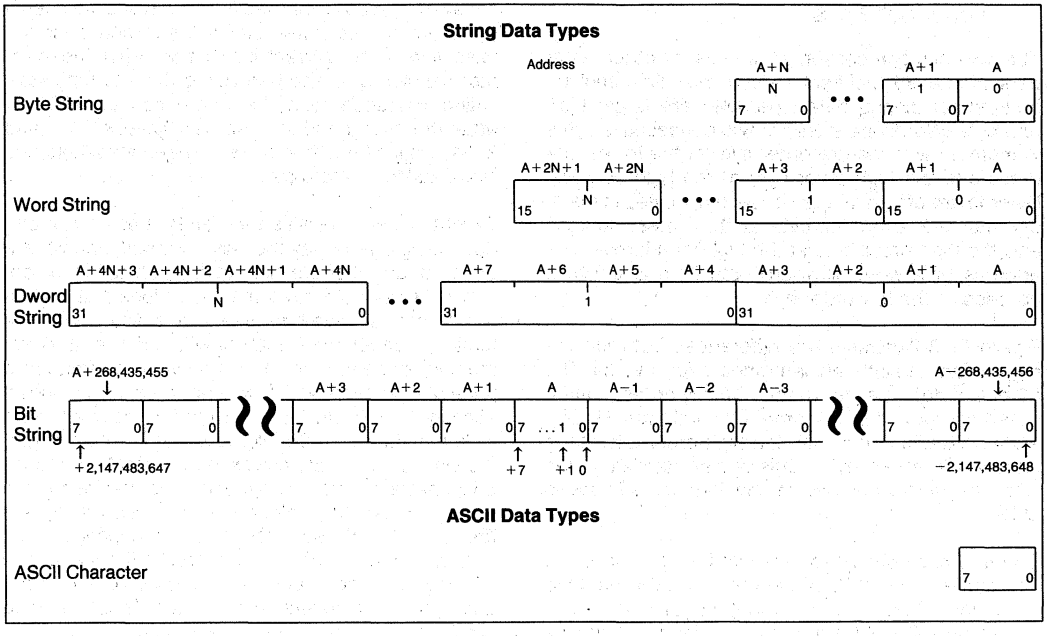


Table 2.14. String and ASCII Data Types



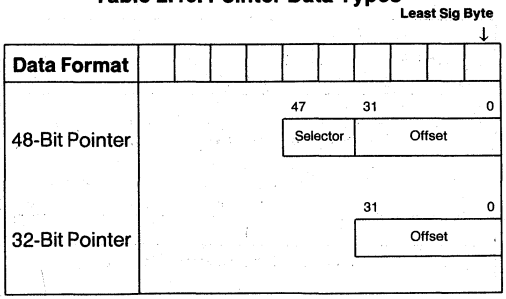
2.6.1.7 Pointer Data Types

A pointer data type contains a value that gives the address of a piece of data. The 486 microprocessor supports two types of pointers. Refer to Table 2.15.

48-bit Pointer: 16-bit selector and 32-bit offset

32-bit Pointer: 32-bit offset

Table 2.15. Pointer Data Types



2.6.2 LITTLE ENDIAN vs BIG ENDIAN DATA FORMATS

The 486 microprocessor, as well as all other members of the 86 architecture use the "little-endian" method for storing data types that are larger than one byte. Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address and the high order byte at the high address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address and the high order byte at the highest address. The address of a word or dword data item is the byte address of the low-order byte.

Figure 2.18 illustrates the differences between the big-endian and little-endian formats for dwords. The 32 bits of data are shown with the low order bit numbered bit 0 and the high order bit numbered 32. Big-endian data is stored with the high-order bits at the lowest addressed byte. Little-endian data is stored with the high-order bits in the highest addressed byte.

The 486 microprocessor has two instructions which can convert 16- or 32-bit data between the two byte orderings. BSWAP (byte swap) handles four byte values and XCHG (exchange) handles two byte values.

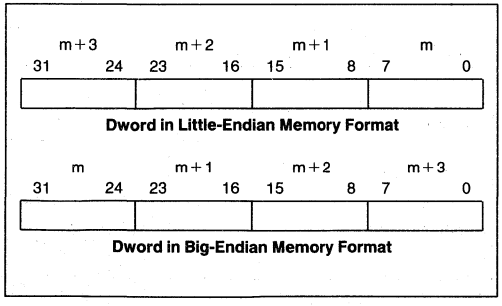


Figure 2.18. Big vs Little Endian Memory Format

2.7 Interrupts

2.7.1 INTERRUPTS AND EXCEPTIONS

Interrupts and exceptions alter the normal program flow, in order to handle external events, to report errors or exceptional conditions. The difference between interrupts and exceptions is that interrupts are used to handle asynchronous external events while exceptions handle instruction faults. Although a program can generate a software interrupt via an INT N instruction, the processor treats software interrupts as exceptions.

Hardware interrupts occur as the result of an external event and are classified into two types: maskable or non-maskable. Interrupts are serviced after the execution of the current instruction. After the interrupt handler is finished servicing the interrupt, execution proceeds with the instruction immediately **after** the interrupted instruction. Sections 2.7.3 and 2.7.4 discuss the differences between Maskable and Non-Maskable interrupts.

Exceptions are classified as faults, traps, or aborts depending on the way they are reported, and whether or not restart of the instruction causing the exception is supported. **Faults** are exceptions that are detected and serviced **before** the execution of the faulting instruction. A fault would occur in a virtual memory system, when the processor referenced a page or a segment which was not present. The operating system would fetch the page or segment from disk, and then the 486 Microprocessor would restart the instruction. **Traps** are exceptions that are reported immediately **after** the execution of the instruction which caused the problem. User defined interrupts are examples of traps. **Aborts** are exceptions which do not permit the precise location of the instruction causing the exception to be determined. Aborts are used to report severe errors, such as a hardware error, or illegal values in system tables.

Thus, when an interrupt service routine has been completed, execution proceeds from the instruction immediately following the interrupted instruction. On the other hand, the return address from an exception fault routine will always point at the instruction causing the exception and include any leading instruction prefixes. Table 2.16 summarizes the possible interrupts for the 486 Microprocessor and shows where the return address points.

The 486 Microprocessor has the ability to handle up to 256 different interrupts/exceptions. In order to service the interrupts, a table with up to 256 interrupt vectors must be defined. The interrupt vectors are simply pointers to the appropriate interrupt service routine. In Real Mode (see Section 3.1), the vectors are 4 byte quantities, a Code Segment plus a 16-bit offset; in Protected Mode, the interrupt vectors are 8 byte quantities, which are put in an Interrupt Descriptor Table (see Section 4.3.3.4). Of the 256 possible interrupts, 32 are reserved for use by Intel, the remaining 224 are free to be used by the system designer.

2.7.2 INTERRUPT PROCESSING

When an interrupt occurs the following actions happen. First, the current program address and the Flags are saved on the stack to allow resumption of the interrupted program. Next, an 8-bit vector is supplied to the 486 Microprocessor which identifies the

appropriate entry in the interrupt table. The table contains the starting address of the interrupt service routine. Then, the user supplied interrupt service routine is executed. Finally, when an IRET instruction is executed the old processor state is restored and program execution resumes at the appropriate instruction.

The 8-bit interrupt vector is supplied to the 486 Microprocessor in several different ways: exceptions supply the interrupt vector internally; software INT instructions contain or imply the vector; maskable hardware interrupts supply the 8-bit vector via the interrupt acknowledge bus sequence. Non-Maskable hardware interrupts are assigned to interrupt vector 2.

2.7.3 MASKABLE INTERRUPT

Maskable interrupts are the most common way used by the 486 Microprocessor to respond to asynchronous external hardware events. A hardware interrupt occurs when the INTR is pulled high and the Interrupt Flag bit (IF) is enabled. The processor only responds to interrupts between instructions, (REPeat String instructions, have an "interrupt window", between memory moves, which allows interrupts during long string moves). When an interrupt occurs the processor reads an 8-bit vector supplied by the hardware which identifies the source of the interrupt, (one of 224 user defined interrupts). The exact nature of the interrupt sequence is discussed in Section 7.2.10.

Table 2.16. Interrupt Vector Assignments

Function	Interrupt Number	Instruction Which Can Cause Exception	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	YES	FAULT
Debug Exception	1	Any Instruction	YES	TRAP*
NMI Interrupt	2	INT 2 or NMI	NO	NMI
One Byte Interrupt	3	INT	NO	TRAP
Interrupt on Overflow	4	INTO	NO	TRAP
Array Bounds Check	5	BOUND	YES	FAULT
Invalid OP-Code	6	Any Illegal Instruction	YES	FAULT
Device Not Available	7	ESC, WAIT	YES	FAULT
Double Fault	8	Any Instruction That Can Generate an Exception		ABORT
Intel Reserved	9			
Invalid TSS	10	JMP, CALL, IRET, INT	YES	FAULT
Segment Not Present	11	Segment Register Instructions	YES	FAULT
Stack Fault	12	Stack References	YES	FAULT
General Protection Fault	13	Any Memory Reference	YES	FAULT
Page Fault	14	Any Memory Access or Code Fetch	YES	FAULT
Intel Reserved	15			
Floating Point Error	16	Floating Point, WAIT	YES	FAULT
Alignment Check Interrupt	17	Unaligned Memory Access	YES	FAULT
Intel Reserved	18–31			
Two Byte Interrupt	0–255	INT n	NO	TRAP

*Some debug exceptions may report both traps on the previous instruction, and faults on the next instruction.

The IF bit in the EFLAG registers is reset when an interrupt is being serviced. This effectively disables servicing additional interrupts during an interrupt service routine. However, the IF may be set explicitly by the interrupt handler, to allow the nesting of interrupts. When an IRET instruction is executed the original state of the IF is restored.

2.7.4 NON-MASKABLE INTERRUPT

Non-maskable interrupts provide a method of servicing very high priority interrupts. A common example of the use of a non-maskable interrupt (NMI) would be to activate a power failure routine. When the NMI input is pulled high it causes an interrupt with an internally supplied vector value of 2. Unlike a normal hardware interrupt, no interrupt acknowledgment sequence is performed for an NMI.

While executing the NMI servicing procedure, the 486 Microprocessor will not service further NMI requests until an interrupt return (IRET) instruction is executed or the processor is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. The IF bit is cleared at the beginning of an NMI interrupt to inhibit further INTR interrupts.

2.7.5 SOFTWARE INTERRUPTS

A third type of interrupt/exception for the 486 Microprocessor is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the nth vector in the interrupt table.

A special case of the two byte software interrupt INT n is the one byte INT 3, or breakpoint interrupt. By inserting this one byte instruction in a program, the user can set breakpoints in his program as a debugging tool.

A final type of software interrupt is the single step interrupt. It is discussed in Section 9.2.

2.7.6 INTERRUPT AND EXCEPTION PRIORITIES

Interrupts are externally-generated events. Maskable Interrupts (on the INTR input) and Non-Maskable Interrupts (on the NMI input) are recognized at instruction boundaries. When NMI and maskable INTR are **both** recognized at the **same** instruction boundary, the 486 Microprocessor invokes the NMI service routine first. If, after the NMI service routine has been invoked, maskable interrupts are still enabled, then the 486 Microprocessor will invoke the appropriate interrupt service routine.

Table 2.17a. i486™ Microprocessor Priority for Invoking Service Routines in Case of Simultaneous External Interrupts

1. NMI
2. INTR

Exceptions are internally-generated events. Exceptions are detected by the 486 Microprocessor if, in the course of executing an instruction, the 486 Microprocessor detects a problematic condition. The 486 Microprocessor then immediately invokes the appropriate exception service routine. The state of the 486 Microprocessor is such that the instruction causing the exception can be restarted. If the exception service routine has taken care of the problematic condition, the instruction will execute without causing the same exception.

It is possible for a single instruction to generate several exceptions (for example, transferring a single operand could generate two page faults if the operand location spans two "not present" pages). However, only one exception is generated upon each attempt to execute the instruction. Each exception service routine should correct its corresponding exception, and restart the instruction. In this manner, exceptions are serviced until the instruction executes successfully.

As the 486 Microprocessor executes instructions, it follows a consistent cycle in checking for exceptions, as shown in Table 2.17b. This cycle is repeated as each instruction is executed, and occurs in parallel with instruction decoding and execution.

Table 2.17b. Sequence of Exception Checking

Consider the case of the 486 Microprocessor having just completed an instruction. It then performs the following checks before reaching the point where the next instruction is completed:

1. Check for Exception 1 Traps from the instruction just completed (single-step via Trap Flag, or Data Breakpoints set in the Debug Registers).
2. Check for Exception 1 Faults in the next instruction (Instruction Execution Breakpoint set in the Debug Registers for the next instruction).
3. Check for external NMI and INTR.
4. Check for Segmentation Faults that prevented fetching the entire next instruction (exceptions 11 or 13).
5. Check for Page Faults that prevented fetching the entire next instruction (exception 14).
6. Check for Faults decoding the next instruction (exception 6 if illegal opcode; exception 6 if in Real Mode or in Virtual 8086 Mode and attempting to execute an instruction for Protected Mode only (see Section 4.6.4); or exception 13 if instruction is longer than 15 bytes, or privilege violation in Protected Mode (i.e., not at IOPL or at CPL=0).
7. If WAIT opcode, check if TS=1 and MP=1 (exception 7 if both are 1).
8. If opcode for Floating Point Unit, check if EM=1 or TS=1 (exception 7 if either are 1).
9. If opcode for Floating Point Unit (FPU), check FPU error status (exception 16 if error status is asserted).
10. Check in the following order for each memory reference required by the instruction:
    a. Check for Segmentation Faults that prevent transferring the entire memory quantity (exceptions 11, 12, 13).
    b. Check for Page Faults that prevent transferring the entire memory quantity (exception 14).

NOTE:
The order stated supports the concept of the paging mechanism being "underneath" the segmentation mechanism. Therefore, for any given code or data reference in memory, segmentation exceptions are generated before paging exceptions are generated.

2.7.7 INSTRUCTION RESTART

The 486 Microprocessor fully supports restarting all instructions after faults. If an exception is detected in the instruction to be executed (exception categories 4 through 10 in Table 2.17b), the 486 Microprocessor invokes the appropriate exception service routine. The 486 Microprocessor is in a state that permits restart of the instruction, for all cases but those in Table 2.17c. Note that all such cases are easily avoided by proper design of the operating system.

Table 2.17c. Conditions Preventing Instruction Restart

An instruction causes a task switch to a task whose Task State Segment is **partially** "not present". (An entirely "not present" TSS is restartable.) Partially present TSS's can be avoided either by keeping the TSS's of such tasks present in memory, or by aligning TSS segments to reside entirely within a single 4K page (for TSS segments of 4 Kbytes or less).

NOTE:
These conditions are avoided by using the operating system designs mentioned in this table.

2.7.8 DOUBLE FAULT

A Double Fault (exception 8) results when the processor attempts to invoke an exception service routine for the segment exceptions (10, 11, 12 or 13), but in the process of doing so, detects an exception other than a Page Fault (exception 14).

A Double Fault (exception 8) will also be generated when the processor attempts to invoke the Page Fault (exception 14) service routine, and detects an exception other than a second Page Fault. In any functional system, the entire Page Fault service routine must remain "present" in memory.

When a Double Fault occurs, the 486 Microprocessor invokes the exception service routine for exception 8.

2.7.9 FLOATING POINT INTERRUPT VECTORS

Several interrupt vectors of the 486 microprocessor are used to report exceptional conditions while executing numeric programs in either real or protected mode. Table 2.18 shows these interrupts and their causes.

Table 2.18. Interrupt Vectors Used by FPU

Interrupt Number	Cause of Interrupt
7	A Floating Point instruction was encountered when EM or TS of the 486™ processor control register zero (CR0) was set. EM = 1 indicates that software emulation of the instruction is required. When TS is set, either a Floating Point or WAIT instruction causes interrupt 7. This indicates that the current FPU context may not belong to the current task.
13	The first word or doubleword of a numeric operand is not entirely within the limit of its segment. The return address pushed onto the stack of the exception handler points at the Floating Point instruction that caused the exception, including any prefixes. The FPU has not executed this instruction; the instruction pointer and data pointer register refer to a previous, correctly executed instruction.
16	The previous numerics instruction caused an unmasked exception. The address of the faulty instruction and the address of its operand are stored in the instruction pointer and data pointer registers. Only Floating Point and WAIT instructions can cause this interrupt. The 486™ processor return address pushed onto the stack of the exception handler points to a WAIT or Floating Point instruction (including prefixes). This instruction can be restarted after clearing the exception condition in the FPU. The FNINIT, FNCLEX, FNSTSW, FNSTENV, and FNSAVE instructions cannot cause this interrupt.

3.0 REAL MODE ARCHITECTURE

3.1 Real Mode Introduction

When the processor is reset or powered up it is initialized in Real Mode. Real Mode has the same base architecture as the 8086, but allows access to the 32-bit register set of the 486 Microprocessor. The addressing mechanism, memory size, interrupt handling, are all identical to the Real Mode on the 80286.

All of the 486 Microprocessor instructions are available in Real Mode (except those instructions listed in Section 4.6.4). The default operand size in Real Mode is 16 bits, just like the 8086. In order to use the 32-bit registers and addressing modes, override prefixes must be used. In addition, the segment size on the 486 Microprocessor in Real Mode is 64 Kbytes so 32-bit effective addresses must have a value less the 0000FFFFH. The primary purpose of Real Mode is to set up the processor for Protected Mode Operation.

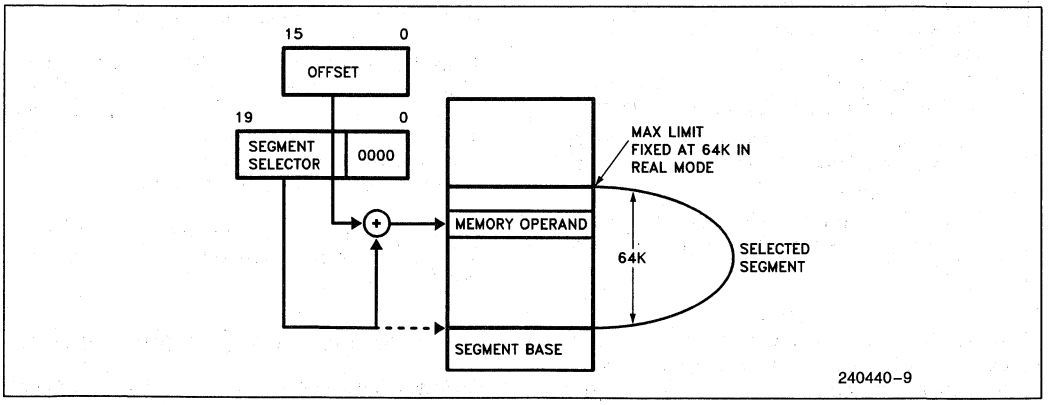


Figure 3.1. Real Address Mode Addressing

The LOCK prefix on the 486 Microprocessor, even in Real Mode, is more restrictive than on the 80286. This is due to the addition of paging on the 486 Microprocessor in Protected Mode and Virtual 8086 Mode. Paging makes it impossible to guarantee that repeated string instructions can be LOCKed. The 486 Microprocessor can't require that all pages holding the string be physically present in memory. Hence, a Page Fault (exception 14) might have to be taken during the repeated string instruction. Therefore the LOCK prefix can't be supported during repeated string instructions.

These are the only instruction forms where the LOCK prefix is legal on the 486 Microprocessor:

Opcode	Operands (Dest, Source)
BIT Test and SET/RESET/COMPLEMENT	Mem, Reg/immed
XCHG	Reg, Mem
XCHG	Mem, Reg
ADD, OR, ADC, SBB, AND, SUB, XOR	Mem, Reg/immed
NOT, NEG, INC, DEC	Mem
CMPXCHG, XADD	Mem, Reg

An exception 6 will be generated if a LOCK prefix is placed before any instruction form or opcode not listed above. The LOCK prefix allows indivisible read/modify/write operations on memory operands using the instructions above. For example, even the ADD Reg, Mem is not LOCKable, because the Mem operand is not the destination (and therefore no memory read/modify/operation is being performed).

Since, on the 486 Microprocessor, repeated string instructions are not LOCKable, it is not possible to LOCK the bus for a long period of time. Therefore, the LOCK prefix is not IOPL-sensitive on the 486 Microprocessor. The LOCK prefix can be used at any privilege level, but only on the instruction forms listed above.

3.2 Memory Addressing

In Real Mode the maximum memory size is limited to 1 megabyte. Thus, only address lines A2–A19 are active. (Exception, after RESET address lines A20–A31 are high during CS-relative memory cycles until an intersegment jump or call is executed (see Section 6.5)).

Since paging is not allowed in Real Mode the linear addresses are the same as physical addresses. Physical addresses are formed in Real Mode by adding the contents of the appropriate segment register which is shifted left by four bits to an effective address. This addition results in a physical address from 00000000H to 0010FFEFH. This is compatible with 80286 Real Mode. Since segment registers are shifted left by 4 bits, Real Mode segments always start on 16 byte boundaries.

All segments in Real Mode are exactly 64 Kbytes long, and may be read, written, or executed. The 486 Microprocessor will generate an exception 13 if a data operand or instruction fetch occurs past the end of a segment (i.e., if an operand has an offset greater than FFFFH, for example a word with a low byte at FFFFH and the high byte at 0000H).

Segments may be overlapped in Real Mode. Thus, if a particular segment does not use all 64 Kbytes another segment can be overlayed on top of the unused portion of the previous segment. This allows the programmer to minimize the amount of physical memory needed for a program.

3.3 Reserved Locations

There are two fixed areas in memory which are reserved in Real address mode: system initialization area and the interrupt table area. Locations 00000H through 003FFH are reserved for interrupt vectors. Each one of the 256 possible interrupts has a 4-byte jump vector reserved for it. Locations FFFFFFF0H through FFFFFFFFH are reserved for system initialization.

3.4 Interrupts

Many of the exceptions shown in Table 2.16 and discussed in Section 2.7 are not applicable to Real Mode operation, in particular exceptions 10, 11, 14, 17, will not happen in Real Mode. Other exceptions have slightly different meanings in Real Mode; Table 3.1 identifies these exceptions.

3.5 Shutdown and Halt

The HLT instruction stops program execution and prevents the processor from using the local bus until restarted. Either NMI, INTR with interrupts enabled (IF = 1), or RESET will force the 486 Microprocessor out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

As in the case in protected mode, the shutdown will occur when a severe error is detected that prevents further processing. In Real Mode, shutdown can occur under two conditions:

An interrupt or an exception occur (exceptions 8 or 13) and the interrupt vector is larger than the Interrupt Descriptor Table (i.e., there is not an interrupt handler for the interrupt).

Table 3.1. Exceptions with Different Meanings in Real Mode (see Table 2.16)

Function	Interrupt Number	Related Instructions	Return Address Location
Interrupt table limit too small	8	INT Vector is not within table limit	Before Instruction
CS, DS, ES, FS, GS Segment overrun exception	13	Word memory reference beyond offset = FFFFH. An attempt to execute past the end of CS segment.	Before Instruction
SS Segment overrun exception	12	Stack Reference beyond offset = FFFFH	Before Instruction

A CALL, INT or PUSH instruction attempts to wrap around the stack segment when SP is not even (i.e., pushing a value on the stack when SP = 0001 resulting in a stack segment greater than FFFFH).

An NMI input can bring the processor out of shutdown if the Interrupt Descriptor Table limit is large enough to contain the NMI interrupt vector (at least 0017H) and the stack has enough room to contain the vector and flag information (i.e., SP is greater than 0005H). If these conditions are not met, the i486 CPU is unable to execute the NMI and executes another shutdown cycle. In this case, the processor remains in the shutdown and can only exit via the RESET input.

4.0 PROTECTED MODE ARCHITECTURE

4.1 Introduction

The complete capabilities of the 486 Microprocessor are unlocked when the processor operates in Protected Virtual Address Mode (Protected Mode). Protected Mode vastly increases the linear address space to four gigabytes (2^{32} bytes) and allows the running of virtual memory programs of almost unlimited size (64 terabytes or 2^{46} bytes). In addition Protected Mode allows the 486 Microprocessor to run all of the existing 8086, 80286 and 386 microprocessor software, while providing a sophisticated memory management and a hardware-assisted protection mechanism. Protected Mode allows the use of additional instructions especially optimized for supporting multitasking operating systems. The base architecture of the 486 Microprocessor remains the same, the registers, instructions, and addressing modes described in the previous sections are retained. The main difference between Protected Mode, and Real Mode from a programmer's view is the increased address space, and a different addressing mechanism.

4.2 Addressing Mechanism

Like Real Mode, Protected Mode uses two components to form the logical address, a 16-bit selector is used to determine the linear base address of a segment, the base address is added to a 32-bit effective address to form a 32-bit linear address. The linear address is then either used as the 32-bit physical address, or if paging is enabled the paging mechanism maps the 32-bit linear address into a 32-bit physical address.

The difference between the two modes lies in calculating the base address. In Protected Mode the selector is used to specify an index into an operating system defined table (see Figure 4.1). The table contains the 32-bit base address of a given segment. The physical address is formed by adding the base address obtained from the table to the offset.

Paging provides an additional memory management mechanism which operates only in Protected Mode. Paging provides a means of managing the very large segments of the 486 Microprocessor. As such, paging operates beneath segmentation. The paging mechanism translates the protected linear address which comes from the segmentation unit into a physical address. Figure 4.2 shows the complete 486 Microprocessor addressing mechanism with paging enabled.

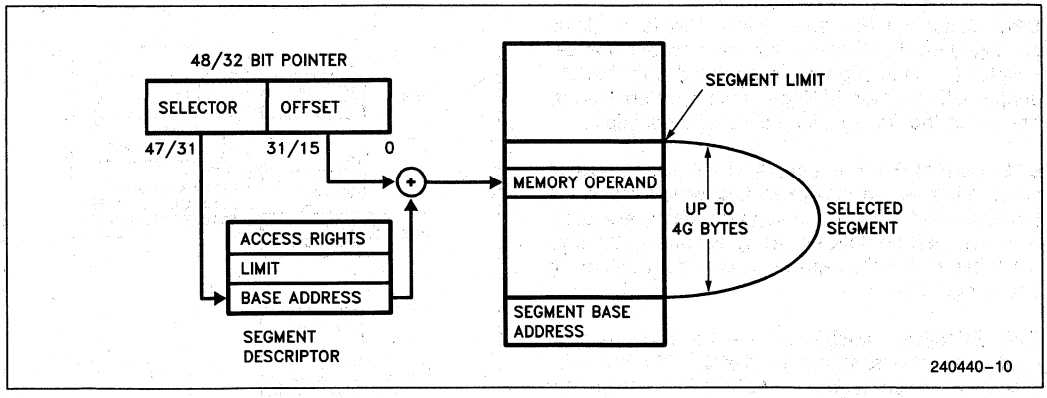


Figure 4.1. Protected Mode Addressing

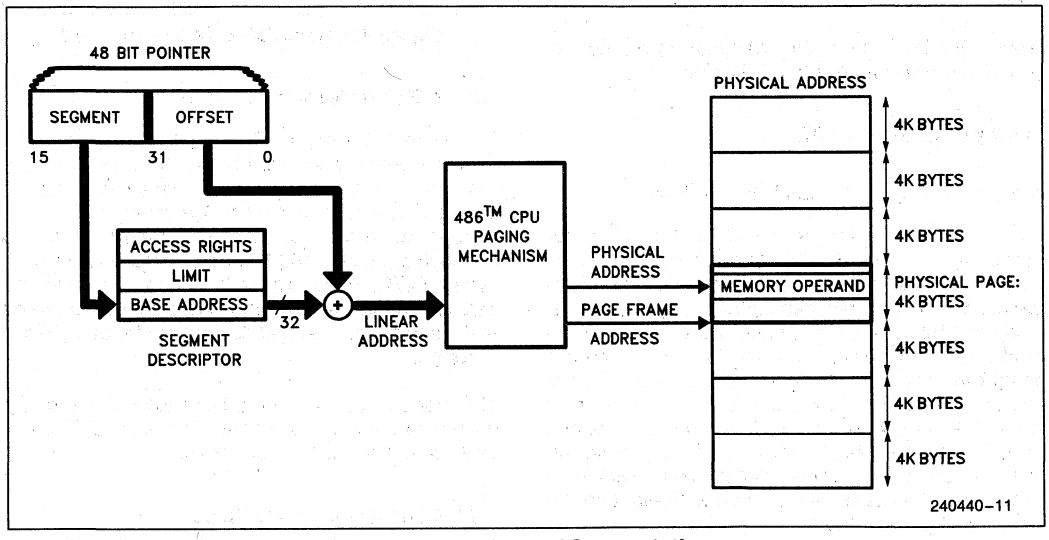


Figure 4.2. Paging and Segmentation

4.3 Segmentation

4.3.1 SEGMENTATION INTRODUCTION

Segmentation is one method of memory management. Segmentation provides the basis for protection. Segments are used to encapsulate regions of memory which have common attributes. For example, all of the code of a given program could be contained in a segment, or an operating system table may reside in a segment. All information about a segment is stored in an 8 byte data structure called a descriptor. All of the descriptors in a system are contained in tables recognized by hardware.

4.3.2 TERMINOLOGY

The following terms are used throughout the discussion of descriptors, privilege levels and protection:

PL: Privilege Level—One of the four hierarchical privilege levels. Level 0 is the most privileged level and level 3 is the least privileged. More privileged levels are numerically smaller than less privileged levels.

RPL: Requestor Privilege Level—The privilege level of the original supplier of the selector. RPL is determined by the **least two** significant bits of a selector.

DPL: Descriptor Privilege Level—This is the least privileged level at which a task may access that descriptor (and the segment associated with that descriptor). Descriptor Privilege Level is determined by bits 6:5 in the Access Right Byte of a descriptor.

CPL: Current Privilege Level—The privilege level at which a task is currently executing, which equals the privilege level of the code segment being executed. CPL can also be determined by examining the lowest 2 bits of the CS register, except for conforming code segments.

EPL: Effective Privilege Level—The effective privilege level is the least privileged of the RPL and DPL. Since smaller privilege level **values** indicate greater privilege, EPL is the numerical maximum of RPL and DPL.

Task: One instance of the execution of a program. Tasks are also referred to as processes.

4.3.3 DESCRIPTOR TABLES

4.3.3.1 Descriptor Tables Introduction

The descriptor tables define all of the segments which are used in an 486 Microprocessor system. There are three types of tables on the 486 Microprocessor which hold descriptors: the Global Descriptor Table, Local Descriptor Table, and the Interrupt Descriptor Table. All of the tables are variable length memory arrays. They can range in size between 8 bytes and 64 Kbytes. Each table can hold up to 8192 8-byte descriptors. The upper 13 bits of a selector are used as an index into the descriptor table. The tables have registers associated with them which hold the 32-bit linear base address, and the 16-bit limit of each table.

Each of the tables has a register associated with it, the GDTR, LDTR, and the IDTR (see Figure 4.3). The LGDT, LLDT, and LIDT instructions, load the base and limit of the Global, Local, and Interrupt Descriptor Tables, respectively, into the appropriate register. The SGDT, SLDT, and SIDT store the base and limit values. These tables are manipulated by the operating system. Therefore, the load descriptor table instructions are privileged instructions.

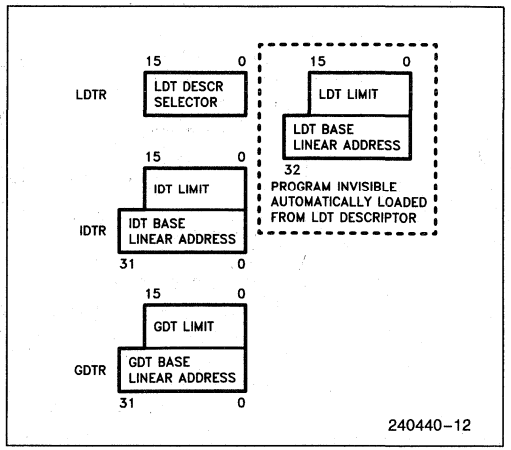


Figure 4.3. Descriptor Table Registers

4.3.3.2 Global Descriptor Table

The Global Descriptor Table (GDT) contains descriptors which are possibly available to all of the tasks in a system. The GDT can contain any type of segment descriptor except for descriptors which are used for servicing interrupts (i.e., interrupt and trap descriptors). Every 486 Microprocessor system contains a GDT. Generally the GDT contains code and data segments used by the operating systems and task state segments, and descriptors for the LDTs in a system.

The first slot of the Global Descriptor Table corresponds to the null selector and is not used. The null selector defines a null pointer value.

4.3.3.3 Local Descriptor Table

LDTs contain descriptors which are associated with a given task. Generally, operating systems are designed so that each task has a separate LDT. The LDT may contain only code, data, stack, task gate, and call gate descriptors. LDTs provide a mechanism for isolating a given task's code and data segments from the rest of the operating system, while the GDT contains descriptors for segments which are common to all tasks. A segment cannot be accessed by a task if its segment descriptor does not exist in either the current LDT or the GDT. This pro-

vides both isolation and protection for a task's segments, while still allowing global data to be shared among tasks.

Unlike the 6 byte GDT or IDT registers which contain a base address and limit, the visible portion of the LDT register contains only a 16-bit selector. This selector refers to a Local Descriptor Table descriptor in the GDT.

4.3.3.4 Interrupt Descriptor Table

The third table needed for 486 Microprocessor systems is the Interrupt Descriptor Table. (See Figure 4.4.) The IDT contains the descriptors which point to the location of up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates, and trap gates. The IDT should be at least 256 bytes in size in order to hold the descriptors for the 32 Intel Reserved Interrupts. Every interrupt used by a system must have an entry in the IDT. The IDT entries are referenced via INT instructions, external interrupt vectors, and exceptions. (See Section 2.7 **Interrupts).**

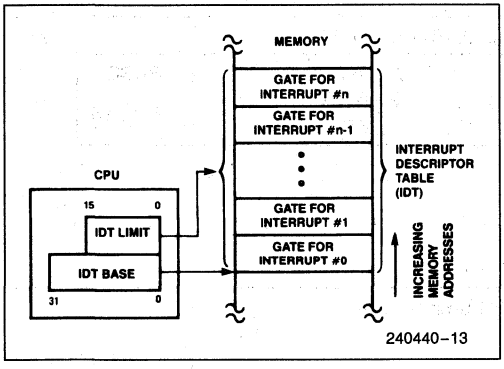


Figure 4.4. Interrupt Descriptor Table Register Use

4.3.4 DESCRIPTORS

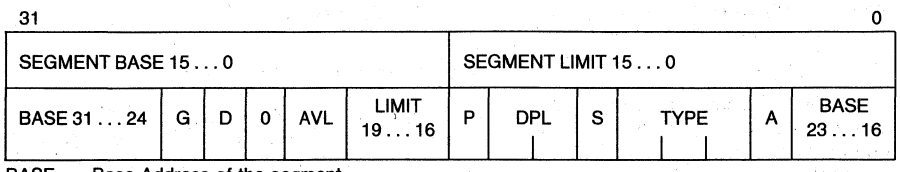
4.3.4.1 Descriptor Attribute Bits

The object to which the segment selector points to is called a descriptor. Descriptors are eight byte quantities which contain attributes about a given region of linear address space (i.e., a segment). These attributes include the 32-bit base linear address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, the default size of the operands (16-bit or 32-bit), and the type of segment. All of the attribute information about a segment is contained in 12 bits in the segment descriptor. Figure 4.5 shows the general format of a descriptor. All segments on the 486 Microprocessor have three attribute fields in common: the **P** bit, the **DPL** bit, and the **S** bit. The Present **P** bit is 1 if the segment is loaded in physical memory, if P=0 then any attempt to access this segment causes a not present exception (exception 11). The Descriptor Privilege Level **DPL** is a two-bit field which specifies the protection level 0–3 associated with a segment.

The 486 Microprocessor has two main categories of segments: system segments and non-system segments (for code and data). The segment **S** bit in the segment descriptor determines if a given segment is a system segment or a code or data segment. If the S bit is 1 then the segment is either a code or data segment, if it is 0 then the segment is a system segment.

4.3.4.2 i486™ CPU Code, Data Descriptors (S=1)

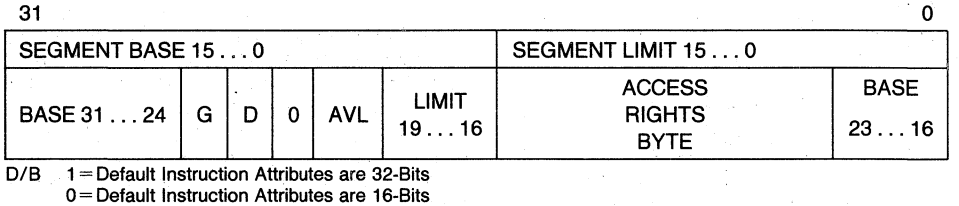
Figure 4.6 shows the general format of a code and data descriptor and Table 4.1 illustrates how the bits in the Access Rights Byte are interpreted.



BASE	Base Address of the segment
LIMIT	The length of the segment
P	Present Bit 1 = Present 0 = Not Present
DPL	Descriptor Privilege Level 0–3
S	Segment Descriptor 0 = System Descriptor 1 = Code or Data Segment Descriptor
TYPE	Type of Segment
A	Accessed Bit
G	Granularity Bit 1 = Segment length is page granular 0 = Segment length is byte granular
D	Default Operation Size (recognized in code segment descriptors only) 1 = 32-bit segment 0 = 16-bit segment
0	Bit must be zero (0) for compatibility with future processors
AVL	Available field for user or OS

NOTE:
In a maximum-size segment (i.e., a segment with G = 1 and segment limit 19...0 = FFFFFH), the lowest 12 bits of the segment base should be zero (i.e., segment base 11...000 = 000H).

Figure 4.5. Segment Descriptors



D/B	1 = Default Instruction Attributes are 32-Bits 0 = Default Instruction Attributes are 16-Bits
AVL	Available field for user or OS
G	Granularity Bit 1 = Segment length is page granular 0 = Segment length is byte granular
0	Bit must be zero (0) for compatibility with future processors

Figure 4.6. Segment Descriptors

Table 4.1. Access Rights Byte Definition for Code and Data Descriptions

	Bit Position	Name	Function	
	7	Present (P)	P = 1 Segment is mapped into physical memory. P = 0 No mapping to physical memory exits, base and limit are not used.	
	6–5	Descriptor Privilege Level (DPL)	Segment privilege attribute used in privilege tests.	
	4	Segment Descriptor (S)	S = 1 Code or Data (includes stacks) segment descriptor. S = 0 System Segment Descriptor or Gate Descriptor.	
Type Field Definition	3	Executable (E)	E = 0 Descriptor type is data segment:	If Data Segment (S = 1, E = 0)
	2	Expansion Direction (ED)	ED = 0 Expand up segment, offsets must be ≤ limit. ED = 1 Expand down segment, offsets must be > limit.	
	1	Writeable (W)	W = 0 Data segment may not be written into. W = 1 Data segment may be written into.	
	3	Executable (E)	E = 1 Descriptor type is code segment:	If Code Segment (S = 1, E = 1)
	2	Conforming (C)	C = 1 Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.	
	1	Readable (R)	R = 0 Code segment may not be read. R = 1 Code segment may be read.	
	0	Accessed (A)	A = 0 Segment has not been accessed. A = 1 Segment selector has been loaded into segment register or used by selector test instructions.	

Code and data segments have several descriptor fields in common. The accessed **A** bit is set whenever the processor accesses a descriptor. The **A** bit is used by operating systems to keep usage statistics on a given segment. The **G** bit, or granularity bit, specifies if a segment length is byte-granular or page-granular. 486 Microprocessor segments can be one megabyte long with byte granularity (G=0) or four gigabytes with page granularity (G=1), (i.e., 2^{20} pages each page is 4 Kbytes in length). The granularity is totally unrelated to paging. A 486 Microprocessor system can consist of segments with byte granularity, and page granularity, whether or not paging is enabled.

The executable **E** bit tells if a segment is a code or data segment. A code segment (E=1, S=1) may be execute-only or execute/read as determined by the Read **R** bit. Code segments are execute only if R=0, and execute/read if R=1. Code segments may never be written into.

NOTE:
Code segments may be modified via aliases. Aliases are writeable data segments which occupy the same range of linear address space as the code segment.

The **D** bit indicates the default length for operands and effective addresses. If D=1 then 32-bit operands and 32-bit addressing modes are assumed. If D=0 then 16-bit operands and 16-bit addressing modes are assumed. Therefore all existing 80286 code segments will execute on the 486 Microprocessor assuming the D bit is set 0.

Another attribute of code segments is determined by the conforming **C** bit. Conforming segments, C=1, can be executed and shared by programs at different privilege levels. (See Section 4.4 **Protection**.)

Segments identified as data segments (E=0, S=1) are used for two types of 486 Microprocessor segments: stack and data segments. The expansion direction **(ED)** bit specifies if a segment expands downward (stack) or upward (data). If a segment is a stack segment all offsets must be greater than the segment limit. On a data segment all offsets must be less than or equal to the limit. In other words, stack segments start at the base linear address plus the maximum segment limit and grow down to the base linear address plus the limit. On the other hand, data segments start at the base linear address and expand to the base linear address plus limit.

The write **W** bit controls the ability to write into a segment. Data segments are read-only if W = 0. The stack segment must have W = 1.

The **B** bit controls the size of the stack pointer register. If B = 1, then PUSHes, POPs, and CALLs all use the 32-bit ESP register for stack references and assume an upper limit of FFFFFFFFH. If B = 0, stack instructions all use the 16-bit SP register and assume an upper limit of FFFFH.

4.3.4.3 System Descriptor Formats

System segments describe information about operating system tables, tasks, and gates. Figure 4.7 shows the general format of system segment descriptors, and the various types of system segments. 486 Microprocessor system descriptors contain a 32-bit base linear address and a 20-bit segment limit. 80286 system descriptors have a 24-bit base address and a 16-bit segment limit. 80286 system descriptors are identified by the upper 16 bits being all zero.

4.3.4.4 LDT Descriptors (S = 0, TYPE = 2)

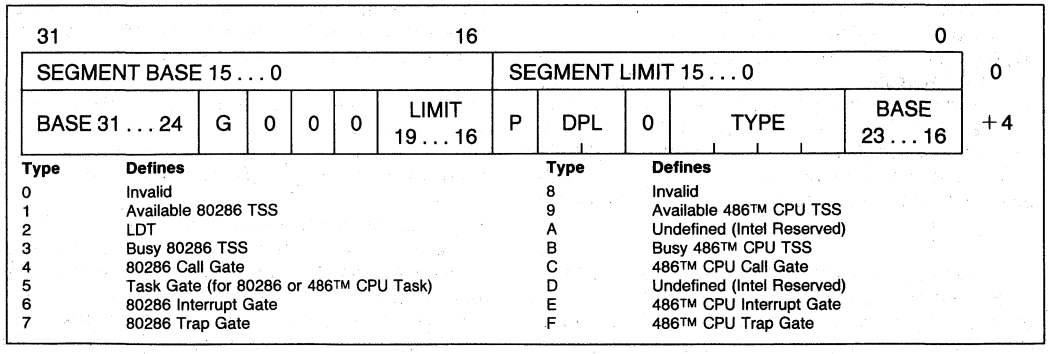
LDT descriptors (S = 0, TYPE = 2) contain information about Local Descriptor Tables. LDTs contain a table of segment descriptors, unique to a particular task. Since the instruction to load the LDTR is only available at privilege level 0, the DPL field is ignored. LDT descriptors are only allowed in the Global Descriptor Table (GDT).

4.3.4.5 TSS Descriptors (S = 0, TYPE = 1, 3, 9, B)

A Task State Segment (TSS) descriptor contains information about the location, size, and privilege level of a Task State Segment (TSS). A TSS in turn is a special fixed format segment which contains all the state information for a task and a linkage field to permit nesting tasks. The TYPE field is used to indicate whether the task is currently BUSY (i.e., on a chain of active tasks) or the TSS is available. The TYPE field also indicates if the segment contains a 80286 or a 486 Microprocessor TSS. The Task Register (TR) contains the selector which points to the current Task State Segment.

4.3.4.6 Gate Descriptors (S = 0, TYPE = 4-7, C, F)

Gates are used to control access to entry points within the target code segment. The various types of gate descriptors are **call** gates, **task** gates, **interrupt** gates, and **trap** gates. Gates provide a level of indirection between the source and destination of the control transfer. This indirection allows the processor to automatically perform protection checks. It also allows system designers to control entry points to the operating system. Call gates are used to change privilege levels (see Section 4.4 **Protection**), task gates are used to perform a task switch, and interrupt and trap gates are used to specify interrupt service routines.



Type	Defines	Type	Defines
0	Invalid	8	Invalid
1	Available 80286 TSS	9	Available 486™ CPU TSS
2	LDT	A	Undefined (Intel Reserved)
3	Busy 80286 TSS	B	Busy 486™ CPU TSS
4	80286 Call Gate	C	486™ CPU Call Gate
5	Task Gate (for 80286 or 486™ CPU Task)	D	Undefined (Intel Reserved)
6	80286 Interrupt Gate	E	486™ CPU Interrupt Gate
7	80286 Trap Gate	F	486™ CPU Trap Gate

Figure 4.7. System Segment Descriptors

Figure 4.8 shows the format of the four types of gate descriptors. Call gates are primarily used to transfer program control to a more privileged level. The call gate descriptor consists of three fields: the access byte, a long pointer (selector and offset) which points to the start of a routine and a word count which specifies how many parameters are to be copied from the caller's stack to the stack of the called routine. The word count field is only used by call gates when there is a change in the privilege level, other types of gates ignore the word count field.

Interrupt and trap gates use the destination selector and destination offset fields of the gate descriptor as a pointer to the start of the interrupt or trap handler routines. The difference between interrupt gates and trap gates is that the interrupt gate disables interrupts (resets the IF bit) while the trap gate does not.

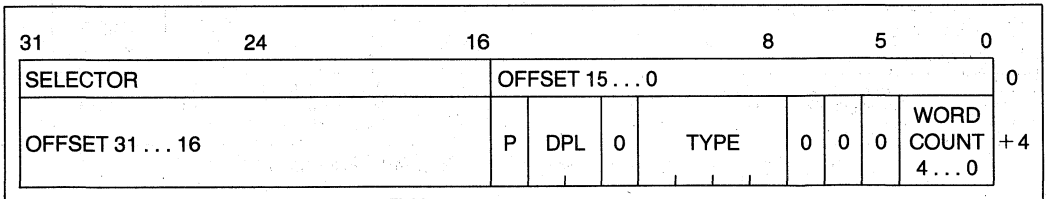
Task gates are used to switch tasks. Task gates may only refer to a task state segment (see Section 4.4.6 **Task Switching**) therefore only the destination selector portion of a task gate descriptor is used, and the destination offset is ignored.

Exception 13 is generated when a destination selector does not refer to a correct descriptor type, i.e., a code segment for an interrupt, trap or call gate, a TSS for a task gate.

The access byte format is the same for all gate descriptors. P = 1 indicates that the gate contents are valid. P = 0 indicates the contents are not valid and causes exception 11 if referenced. DPL is the descriptor privilege level and specifies when this descriptor may be used by a task (see Section 4.4 **Protection**). The S field, bit 4 of the access rights byte, must be 0 to indicate a system control descriptor. The type field specifies the descriptor type as indicated in Figure 4.8.

4.3.4.7 Differences Between i486™ Microprocessor and 80286 Descriptors

In order to provide operating system compatibility between the 80286 and 486 Microprocessor, the 486 Microprocessor supports all of the 80286 segment descriptors. Figure 4.9 shows the general format of an 80286 system segment descriptor. The only differences between 80286 and 486 Microprocessor descriptor formats are that the values of the type fields, and the limit and base address fields have been expanded for the 486 Microprocessor. The 80286 system segment descriptors contained a 24-bit base address and 16-bit limit, while the 486 Microprocessor system segment descriptors have a 32-bit base address, a 20-bit limit field, and a granularity bit.



Gate Descriptor Fields

Name	Value	Description
Type	4	80286 call gate
	5	Task gate (for 80286 or 486™ CPU task)
	6	80286 interrupt gate
	7	80286 trap gate
	C	486™ CPU call gate
	E	486™ CPU interrupt gate
	F	486™ CPU trap gate
P	0	Descriptor contents are not valid
	1	Descriptor contents are valid

DPL—least privileged level at which a task may access the gate. WORD COUNT 0–31—the number of parameters to copy from caller's stack to the called procedure's stack. The parameters are 32-bit quantities for 486™ CPU gates, and 16-bit quantities for 80286 gates.

DESTINATION SELECTOR	16-bit selector	Selector to the target code segment or Selector to the target task state segment for task gate
DESTINATION OFFSET	offset 16-bit 80286 32-bit 486™ CPU	Entry point within the target code segment

Figure 4.8. Gate Descriptor Formats

By supporting 80286 system segments the 486 Microprocessor is able to execute 80286 application programs on a 486 Microprocessor operating system. This is possible because the processor automatically understands which descriptors are 80286-style descriptors and which descriptors are 486 Microprocessor-style descriptors. In particular, if the upper word of a descriptor is zero, then that descriptor is a 80286-style descriptor.

The only other differences between 80286-style descriptors and 486 Microprocessor descriptors is the interpretation of the word count field of call gates and the B bit. The word count field specifies the number of 16-bit quantities to copy for 80286 call gates and 32-bit quantities for 486 Microprocessor call gates. The B bit controls the size of PUSHes when using a call gate; if B=0 PUSHes are 16 bits, if B=1 PUSHes are 32 bits.

4.3.4.8 Selector Fields

A selector in Protected Mode has three fields: Local or Global Descriptor Table Indicator (TI), Descriptor Entry Index (Index), and Requestor (the selector's) Privilege Level (RPL) as shown in Figure 4.10. The TI bits select one of two memory-based tables of descriptors (the Global Descriptor Table or the Local Descriptor Table). The Index selects one of 8K descriptors in the appropriate descriptor table. The RPL bits allow high speed testing of the selector's privilege attributes.

4.3.4.9 Segment Descriptor Cache

In addition to the selector value, every segment register has a segment descriptor cache register associated with it. Whenever a segment register's contents are changed, the 8-byte descriptor associated with that selector is automatically loaded (cached) on the chip. Once loaded, all references to that segment use the cached descriptor information instead of reaccessing the descriptor. The contents of the descriptor cache are not visible to the programmer. Since descriptor caches only change when a segment register is changed, programs which modify the descriptor tables must reload the appropriate segment registers after changing a descriptor's value.

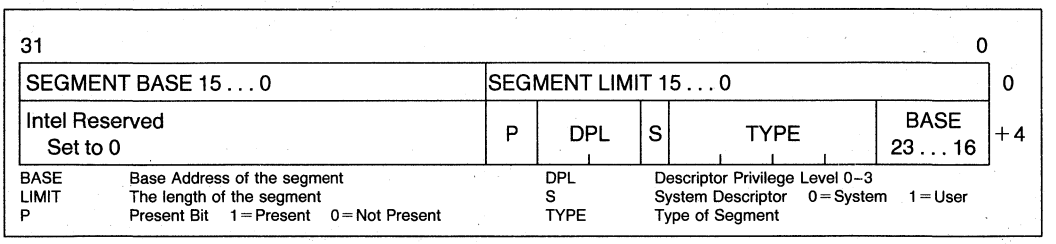


Figure 4.9. 80286 Code and Data Segment Descriptors

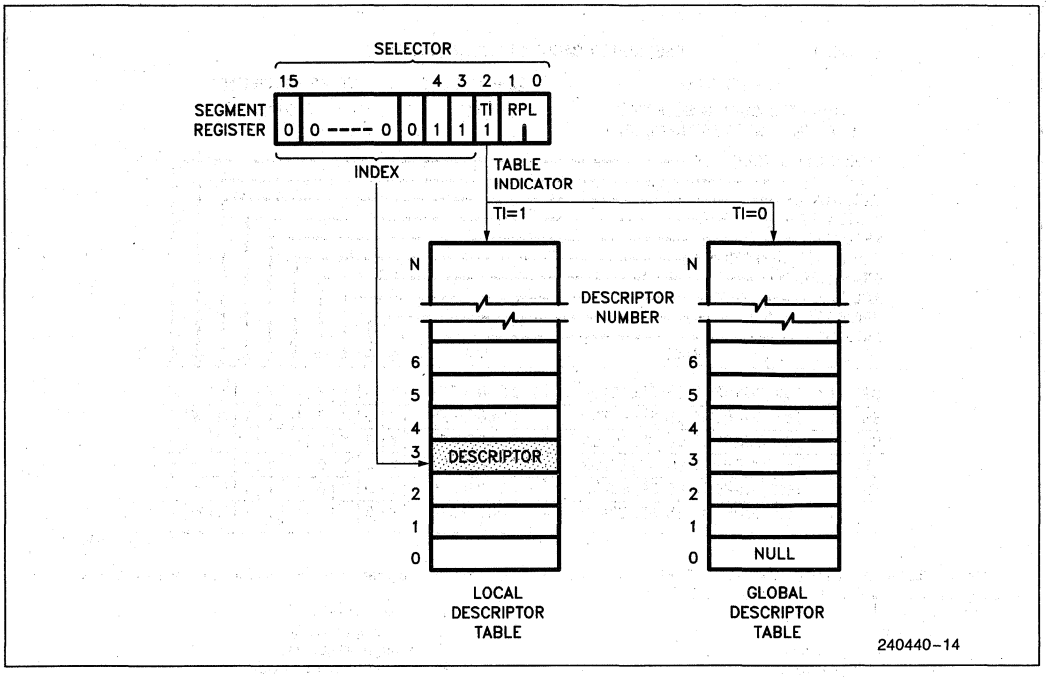
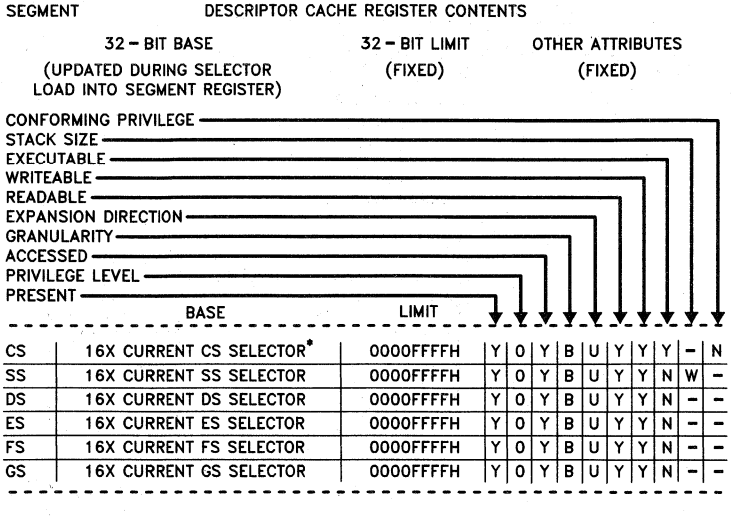


Figure 4.10. Example Descriptor Selection

4.3.4.10 Segment Descriptor Register Settings

The contents of the segment descriptor cache vary depending on the mode the 486 Microprocessor is operating in. When operating in Real Address Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.11. For compatibility with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. In Real Address Mode, the internal "privilege level" is always fixed to the highest level, level 0, so I/O and other privileged opcodes may be executed.

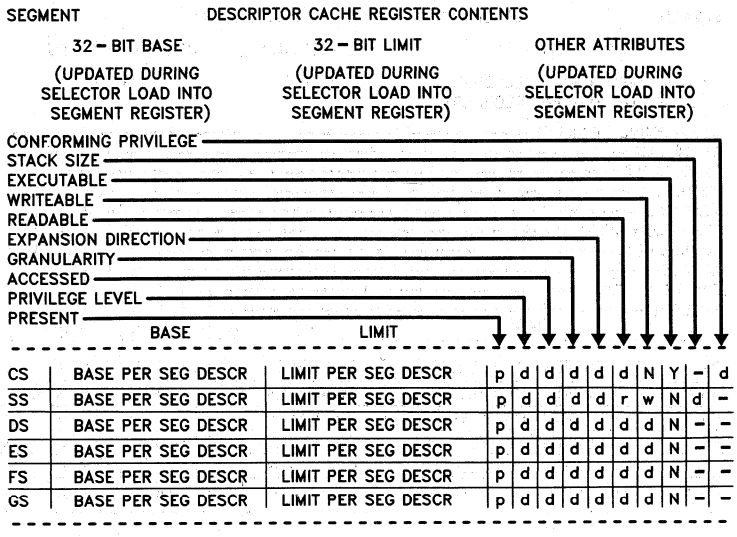


*Except the 32-bit CS base is initialized to FFFFF000H after reset until first intersegment control transfer (i.e., intersegment CALL, or intersegment JMP, or INT). (See Figure 4.13 Example.)

Key:
- Y = yes
- N = no
- 0 = privilege level 0
- 1 = privilege level 1
- 2 = privilege level 2
- 3 = privilege level 3
- U = expand up
- D = expand down
- B = byte granularity
- P = page granularity
- W = push/pop 16-bit words
- F = push/pop 32-bit dwords
- – = does not apply to that segment cache register

Figure 4.11. Segment Descriptor Caches for Real Address Mode (Segment Limit and Attributes are Fixed)

When operating in Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.12. In Protected Mode, each of these fields are defined according to the contents of the segment descriptor indexed by the selector value loaded into the segment register.



240440-16

Key: Y = fixed yes
N = fixed no
d = per segment descriptor
p = per segment descriptor; descriptor must indicate "present" to avoid exception 11 (exception 12 in case of SS)
r = per segment descriptor, but descriptor must indicate "readable" to avoid exception 13 (special case for SS)
w = per segment descriptor, but descriptor must indicate "writable" to avoid exception 13 (special case for SS)
– = does not apply to that segment cache register

Figure 4.12. Segment Descriptor Caches for Protected Mode (Loaded per Descriptor)

When operating in a Virtual 8086 Mode within the Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4.13. For compatibility with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. The virtual program executes at lowest privilege level, level 3, to allow trapping of all IOPL-sensitive instructions and level-0-only instructions.

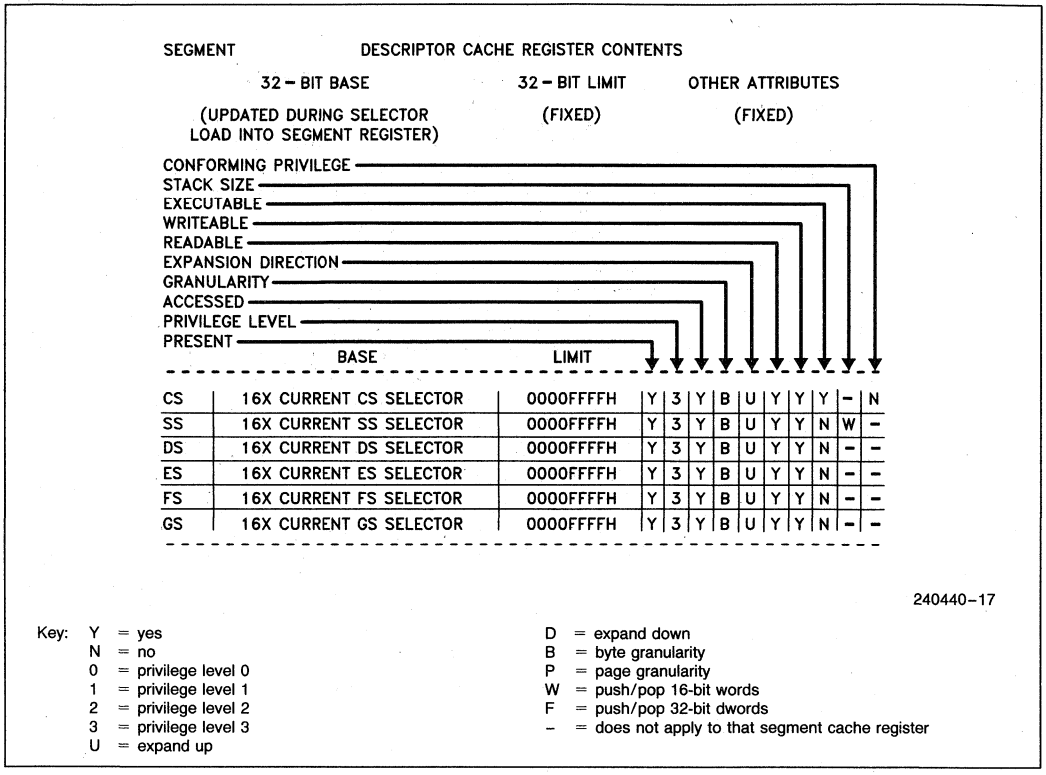


Figure 4.13. Segment Descriptor Caches for Virtual 8086 Mode within Protected Mode (Segment Limit and Attributes are Fixed)

4.4 Protection

4.4.1 PROTECTION CONCEPTS

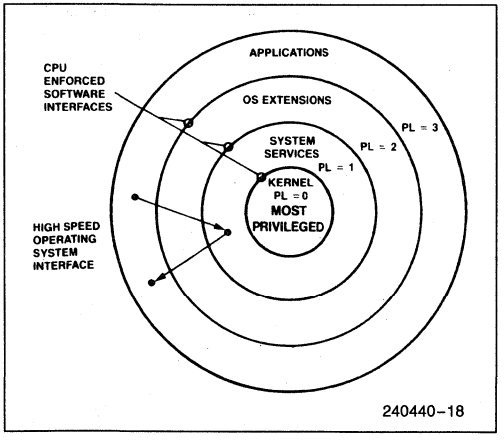


Figure 4.14. Four-Level Hierarchical Protection

The 486 Microprocessor has four levels of protection which are optimized to support the needs of a multi-tasking operating system to isolate and protect user programs from each other and the operating system. The privilege levels control the use of privileged instructions, I/O instructions, and access to segments and segment descriptors. Unlike traditional microprocessor-based systems where this protection is achieved only through the use of complex external hardware and software the 486 Microprocessor provides the protection as part of its integrated Memory Management Unit. The 486 Microprocessor offers an additional type of protection on a page basis, when paging is enabled (See Section 4.5.3 **Page Level Protection**).

The four-level hierarchical privilege system is illustrated in Figure 4-14. It is an extension of the user/supervisor privilege mode commonly used by minicomputers and, in fact, the user/supervisor mode is fully supported by the 486 Microprocessor paging

mechanism. The privilege levels (PL) are numbered 0 through 3. Level 0 is the most privileged or trusted level.

4.4.2 RULES OF PRIVILEGE

The 486 Microprocessor controls access to both data and procedures between levels of a task, according to the following rules.

- Data stored in a segment with privilege level **p** can be accessed only by code executing at a privilege level at least as privileged as **p**.
- A code segment/procedure with privilege level **p** can only be called by a task executing at the same or a lesser privilege level than **p**.

4.4.3 PRIVILEGE LEVELS

4.4.3.1 Task Privilege

At any point in time, a task on the 486 Microprocessor always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies the task's privilege level. A task's CPL may only be changed by control transfers through gate descriptors to a code segment with a different privilege level. (See Section 4.4.4 **Privilege Level Transfers**) Thus, an application program running at PL = 3 may call an operating system routine at PL = 1 (via a gate) which would cause the task's CPL to be set to 1 until the operating system routine was finished.

4.4.3.2 Selector Privilege (RPL)

The privilege level of a selector is specified by the RPL field. The RPL is the two least significant bits of the selector. The selector's RPL is only used to establish a less trusted privilege level than the current privilege level for the use of a segment. This level is called the task's effective privilege level (EPL). The EPL is defined as being the least privileged (i.e. numerically larger) level of a task's CPL and a selector's RPL. Thus, if selector's RPL = 0 then the CPL always specifies the privilege level for making an access using the selector. On the other hand if RPL = 3 then a selector can only access segments at level 3 regardless of the task's CPL. The RPL is most commonly used to verify that pointers passed to an operating system procedure do not access data that is of higher privilege than the procedure that originated the pointer. Since the originator of a selector can specify any RPL value, the Adjust RPL (ARPL) instruction is provided to force the RPL bits to the originator's CPL.

4.4.3.3 I/O Privilege and I/O Permission Bitmap

The I/O privilege level (IOPL, a 2-bit field in the EFLAG register) defines the least privileged level at which I/O instructions can be unconditionally performed. I/O instructions can be unconditionally performed when CPL ≤ IOPL. (The I/O instructions are IN, OUT, INS, OUTS, REP INS, and REP OUTS.) When CPL > IOPL, and the current task is associated with a 286 TSS, attempted I/O instructions cause an exception 13 fault. When CPL > IOPL, and the current task is associated with a 486 Microprocessor TSS, the I/O Permission Bitmap (part of a 486 Microprocessor TSS) is consulted on whether I/O to the port is allowed, or an exception 13 fault is to be generated instead. For diagrams of the I/O Permission Bitmap, refer to Figures 4.15a and 4.15b. For further information on how the I/O Permission Bitmap is used in Protected Mode or in Virtual 8086 Mode, refer to Section 4.6.4 Protection and I/O Permission Bitmap.

The I/O privilege level (IOPL) also affects whether several other instructions can be executed or cause an exception 13 fault instead. These instructions are called "IOPL-sensitive" instructions and they are CLI and STI. (Note that the LOCK prefix is *not* IOPL-sensitive on the 486 Microprocessor.)

The IOPL also affects whether the IF (interrupts enable flag) bit can be changed by loading a value into the EFLAGS register. When CPL ≤ IOPL, then the IF bit can be changed by loading a new value into the EFLAGS register. When CPL > IOPL, the IF bit cannot be changed by a new value POP'ed into (or otherwise loaded into) the EFLAGS register; the IF bit merely remains unchanged and no exception is generated.

Table 4.2. Pointer Test Instructions

Instruction	Operands	Function
ARPL	Selector, Register	Adjust Requested Privilege Level: adjusts the RPL of the selector to the numeric maximum of current selector RPL value and the RPL value in the register. Set zero flag if selector RPL was changed.
VERR	Selector	VERify for Read: sets the zero flag if the segment referred to by the selector can be read.
VERW	Selector	VERify for Write: sets the zero flag if the segment referred to by the selector can be written.
LSL	Register, Selector	Load Segment Limit: reads the segment limit into the register if privilege rules and descriptor type allow. Set zero flag if successful.
LAR	Register, Selector	Load Access Rights: reads the descriptor access rights byte into the register if privilege rules allow. Set zero flag if successful.

4.4.3.4 Privilege Validation

The 486 Microprocessor provides several instructions to speed pointer testing and help maintain system integrity by verifying that the selector value refers to an appropriate segment. Table 4.2 summarizes the selector validation procedures available for the 486 Microprocessor.

This pointer verification prevents the common problem of an application at PL = 3 calling a operating systems routine at PL = 0 and passing the operating system routine a "bad" pointer which corrupts a data structure belonging to the operating system. If the operating system routine uses the ARPL instruction to ensure that the RPL of the selector has no greater privilege than that of the caller, then this problem can be avoided.

4.4.3.5 Descriptor Access

There are basically two types of segment accesses: those involving code segments such as control transfers, and those involving data accesses. Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL as described above.

Any time an instruction loads data segment registers (DS, ES, FS, GS) the 486 Microprocessor makes protection validation checks. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segments or readable code segments. The data access rules are specified in Section 4.4.2 **Rules of Privilege**. The only exception to those rules is readable conforming code segments which can be accessed at any privilege level.

Finally the privilege validation checks are performed. The CPL is compared to the EPL and if the EPL is more privileged than the CPL an exception 13 (general protection fault) is generated.

The rules regarding the stack segment are slightly different than those involving data segments. Instructions that load selectors into SS must refer to data segment descriptors for writeable data segments. The DPL and RPL must equal the CPL. All other descriptor types or a privilege level violation will cause exception 13. A stack not present fault causes exception 12. Note that an exception 11 is used for a not-present code or data segment.

4.4.4 PRIVILEGE LEVEL TRANSFERS

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call or a jump to another routine. There are five types of control transfers which are summarized in Table 4.3. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only via control transfers, by using gates, task switches, and interrupt or trap gates.

Table 4.3. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL	CALL	Call Gate	GDT/LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
	CALL, JMP	Task State Segment	GDT
Task Switch	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag register) = 0
**NT (Nested Task bit of flag register) = 1

Control transfers can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules will cause an exception 13 (e.g. JMP through a call gate, or IRET from a normal subroutine call).

In order to provide further system security, all control transfers are also subject to the privilege rules.

The privilege rules require that:

— Privilege level transitions can only occur via gates.

— JMPs can be made to a non-conforming code segment with the same privilege or to a conforming code segment with greater or equal privilege.

— CALLs can be made to a non-conforming code segment with the same privilege or via a gate to a more privileged level.

— Interrupts handled within the task obey the same privilege rules as CALLs.

— Conforming Code segments are accessible by privilege levels which are the same or less privileged than the conforming-code segment's DPL.

— Both the requested privilege level (RPL) in the selector pointing to the gate and the task's CPL must be of equal or greater privilege than the gate's DPL.

— The code segment selected in the gate must be the same or more privileged than the task's CPL.

— Return instructions that do not switch tasks can only return control to a code segment with same or less privilege.

— Task switches can be performed by a CALL, JMP, or INT which references either a task gate or task state segment who's DPL is less privileged or the same privilege as the old task's CPL.

Any control transfer that changes CPL within a task causes a change of stacks as a result of the privilege level change. The initial values of SS:ESP for privilege levels 0, 1, and 2 are retained in the task state segment (see Section 4.4.6 **Task Switching**). During a JMP or CALL control transfer, the new stack pointer is loaded into the SS and ESP registers and the previous stack pointer is pushed onto the new stack.

When RETurning to the original privilege level, use of the lower-privileged stack is restored as part of the RET or IRET instruction operation. For subroutine calls that pass parameters on the stack and cross privilege levels, a fixed number of words (as specified in the gate's word count field) are copied from the previous stack to the current stack. The inter-segment RET instruction with a stack adjustment value will correctly restore the previous stack pointer upon return.

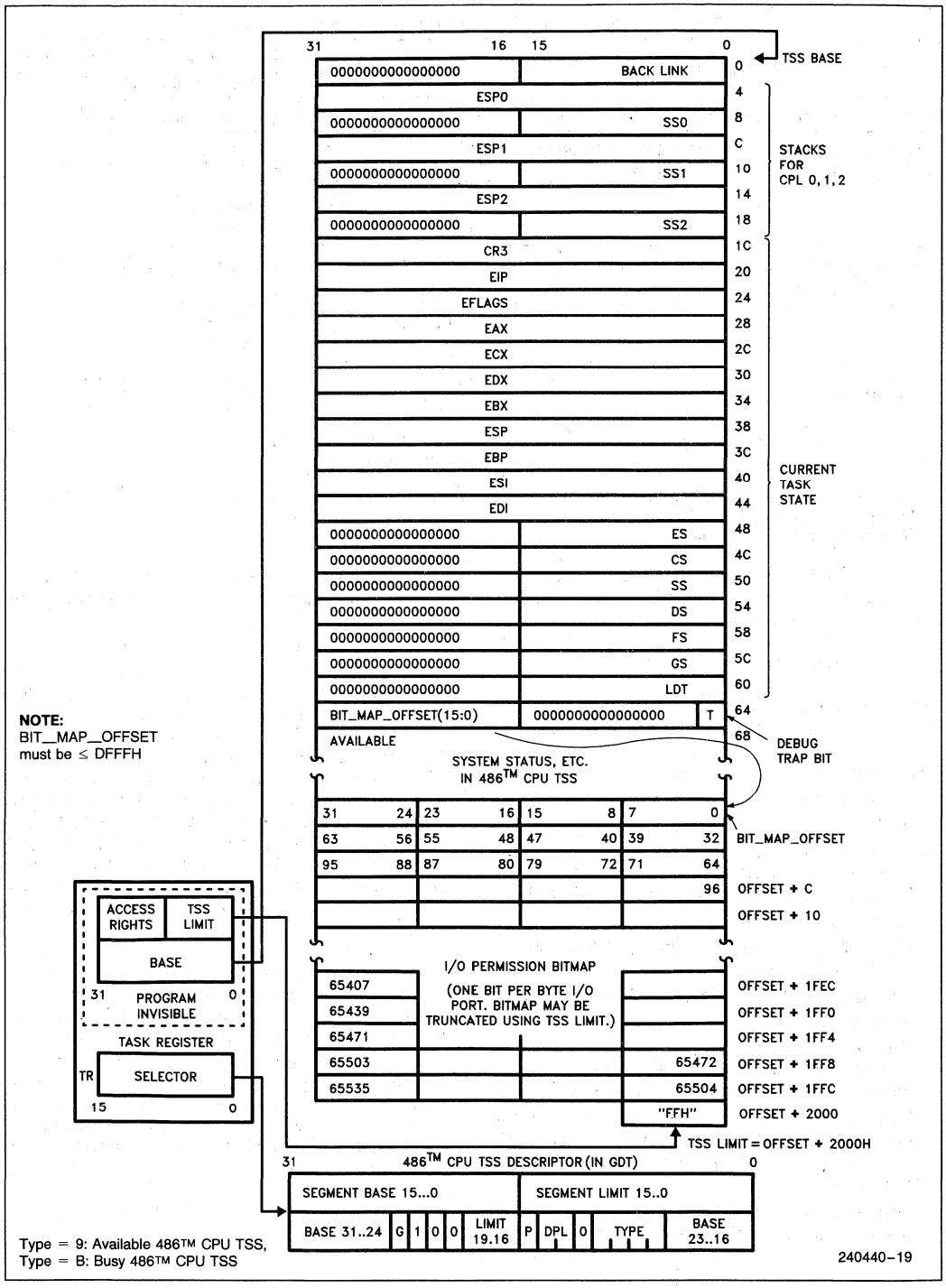


Figure 4.15a. i486™ Microprocessor TSS and TSS Registers

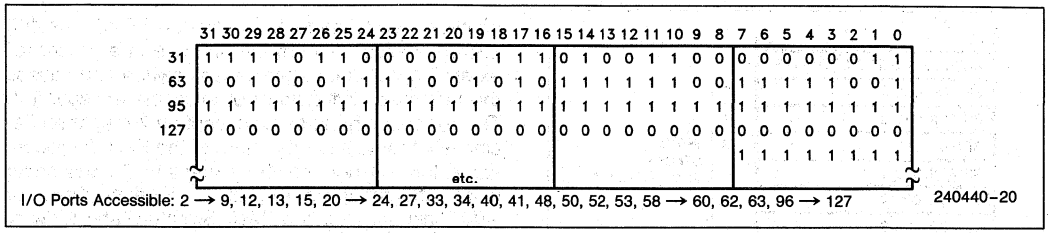


Figure 4.15b. Sample I/O Permission Bit Map

4.4.5 CALL GATES

Gates provide protected, indirect CALLs. One of the major uses of gates is to provide a secure method of privilege transfers within a task. Since the operating system defines all of the gates in a system, it can ensure that all gates only allow entry into a few trusted procedures (such as those which allocate memory, or perform I/O).

Gate descriptors follow the data access rules of privilege; that is, gates can be accessed by a task if the EPL, is equal to or more privileged than the gate descriptor's DPL. Gates follow the control transfer rules of privilege and therefore may only transfer control to a more privileged level.

Call Gates are accessed via a CALL instruction and are syntactically identical to calling a normal subroutine. When an inter-level 486 Microprocessor call gate is activated, the following actions occur.

1. Load CS:EIP from gate check for validity
2. SS is pushed zero-extended to 32 bits
3. ESP is pushed
4. Copy Word Count 32-bit parameters from the old stack to the new stack
5. Push Return address on stack

The procedure is identical for 80286 Call gates, except that 16-bit parameters are copied and 16-bit registers are pushed.

Interrupt Gates and Trap gates work in a similar fashion as the call gates, except there is no copying of parameters. The only difference between Trap and Interrupt gates is that control transfers through an Interrupt gate disable further interrupts (i.e. the IF bit is set to 0), and Trap gates leave the interrupt status unchanged.

4.4.6 TASK SWITCHING

A very important attribute of any multi-tasking/multi-user operating systems is its ability to rapidly switch between tasks or processes. The 486 Microprocessor directly supports this operation by providing a task switch instruction in hardware. The 486 Microprocessor task switch operation saves the entire state of the machine (all of the registers, address space, and a link to the previous task), loads a new execution state, performs protection checks, and commences execution in the new task, in about 10 microseconds. Like transfer of control via gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap, or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot.

The TSS descriptor points to a segment (see Figure 4.15) containing the entire 486 Microprocessor execution state while a task gate descriptor contains a TSS selector. The 486 Microprocessor supports both 80286 and 486 Microprocessor style TSSs. Figure 4.16 shows a 80286 TSS. The limit of a 486 Microprocessor TSS must be greater than 0064H (002BH for a 80286 TSS), and can be as large as 4 Gigabytes. In the additional TSS space, the operating system is free to store additional information such as the reason the task is inactive, time the task has spent running, and open files belong to the task.

Each task must have a TSS associated with it. The current TSS is identified by a special register in the 486 Microprocessor called the Task State Segment Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR are loaded whenever TR is loaded with a new selector. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to the task which was interrupted. The current executing task's state is saved in the TSS and the old task state is restored from its TSS.

Several bits in the flag register and machine status word (CR0) give information about the state of a task which are useful to the operating system. The Nested Task (NT) (bit 14 in EFLAGS) controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular return; when NT = 1, IRET performs a task switch operation back to the previous task. The NT bit is set or reset in the following fashion:

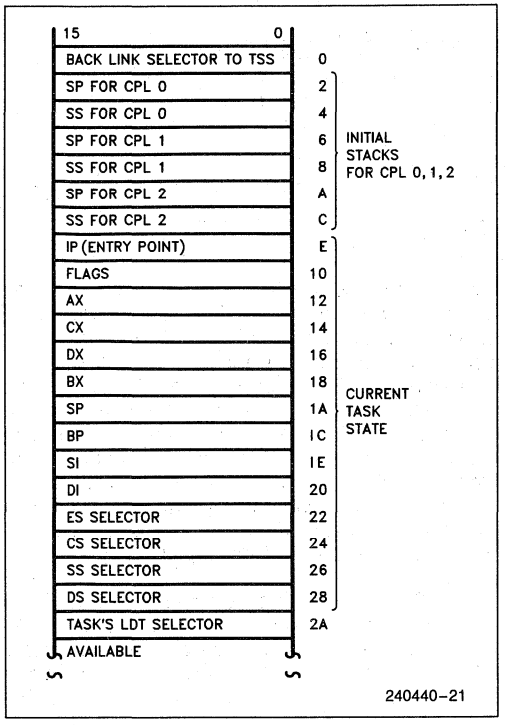


Figure 4.16. 80286 TSS

When a CALL or INT instruction initiates a task switch, the new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT. (The NT bit will be restored after execution of the interrupt handler) NT may also be set or cleared by POPF or IRET instructions.

The 486 Microprocessor task state segment is marked busy by changing the descriptor type field from TYPE 9H to TYPE BH. An 80286 TSS is marked busy by changing the descriptor type field from TYPE 1 to TYPE 3. Use of a selector that references a busy task state segment causes an exception 13.

The Virtual Mode (VM) bit 17 is used to indicate if a task, is a virtual 8086 task. If VM = 1, then the tasks will use the Real Mode addressing mechanism. The virtual 8086 environment is only entered and exited via a task switch (see Section 4.6 **Virtual Mode**).

The FPU's state is not automatically saved when a task switch occurs, because the incoming task may not use the FPU. The Task Switched (TS) Bit (bit 3 in the CR0) helps deal with the FPU's state in a multitasking environment. Whenever the 486 Microprocessor switches tasks, it sets the TS bit. The 486 Microprocessor detects the first use of a processor extension instruction after a task switch and causes the processor extension not available exception 7. The exception handler for exception 7 may then decide whether to save the state of the FPU. A processor extension not present exception (7) will occur when attempting to execute a Floating Point or WAIT instruction if the Task Switched and Monitor coprocessor extension bits are both set (i.e. TS = 1 and MP = 1).

The **T** bit in the 486 Microprocessor TSS indicates that the processor should generate a debug exception when switching to a task. If T = 1 then upon entry to a new task a debug exception 1 will be generated.

4.4.7 INITIALIZATION AND TRANSITION TO PROTECTED MODE

Since the 486 Microprocessor begins executing in Real Mode immediately after RESET it is necessary to initialize the system tables and registers with the appropriate values.

The GDT and IDT registers must refer to a valid GDT and IDT. The IDT should be at least 256 bytes long, and GDT must contain descriptors for the initial code, and data segments. Figure 4.17 shows the tables and Figure 4.18 the descriptors needed for a simple Protected Mode 486 Microprocessor system. It has a single code and single data/stack segment each four gigabytes long and a single privilege level PL = 0.

The actual method of enabling Protected Mode is to load CR0 with the PE bit set, via the MOV CR0, R/M instruction. This puts the 486 Microprocessor in Protected Mode.

After enabling Protected Mode, the next instruction should execute an intersegment JMP to load the CS register and flush the instruction decode queue. The final step is to load all of the data segment registers with the initial selector values.

An alternate approach to entering Protected Mode which is especially appropriate for multi-tasking operating systems, is to use the built in task-switch to load all of the registers. In this case the GDT would contain two TSS descriptors in addition to the code and data descriptors needed for the first task. The first JMP instruction in Protected Mode would jump to the TSS causing a task switch and loading all of the registers with the values stored in the TSS. The Task State Segment Register should be initialized to point to a valid TSS descriptor since a task switch saves the state of the current task in a task state segment.

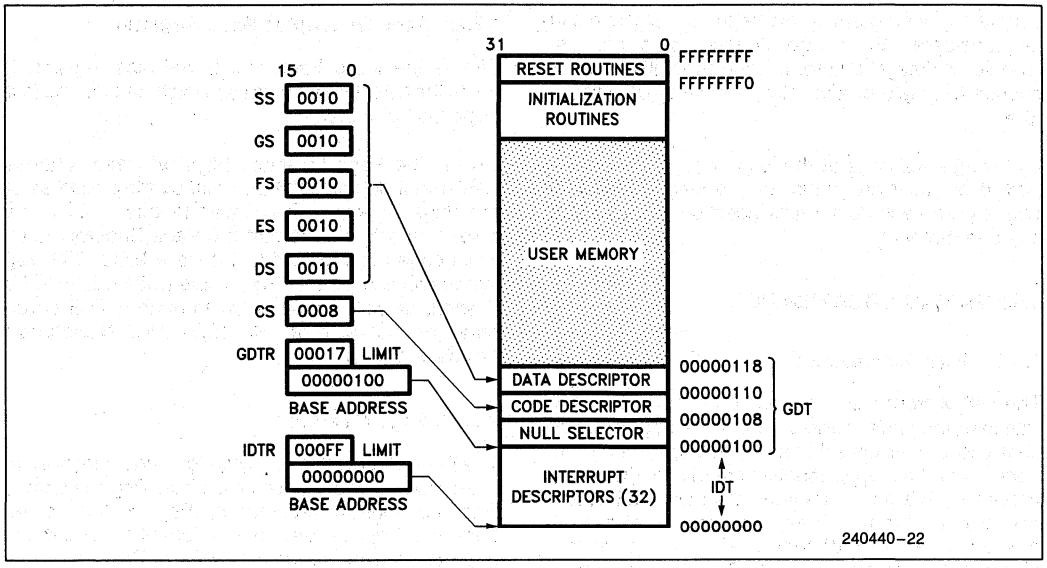


Figure 4.17. Simple Protected System

DATA DESCRIPTOR	2	BASE 31 . . . 24 00 (H)	G 1	D 1	0	0	LIMIT 19.16 F (H)	1	0	0	1	0	0	1	0	BASE 23 . . . 16 00 (H)
		SEGMENT BASE 15 . . . 0 0118 (H)						SEGMENT LIMIT 15 . . . 0 FFFF (H)								
CODE DESCRIPTOR	1	BASE 31 . . . 24 00 (H)	G 1	D 1	0	0	LIMIT 19.16 F (H)	1	0	0	1	1	0	1	0	BASE 23 . . . 16 00 (H)
		SEGMENT BASE 15 . . . 0 0118 (H)						SEGMENT LIMIT 15 . . . 0 FFFF (H)								
		NULL DESCRIPTOR														
	0															
		31 24 16						15 8 0								

Figure 4.18. GDT Descriptors for Simple System

4.4.8 TOOLS FOR BUILDING PROTECTED SYSTEMS

In order to simplify the design of a protected multitasking system, Intel provides a tool which allows the system designer an easy method of constructing the data structures needed for a Protected Mode 486 Microprocessor system. This tool is the builder BLD-386™. BLD-386 lets the operating system writer specify all of the segment descriptors discussed in the previous sections (LDTs, IDTs, GDTs, Gates, and TSSs) in a high-level language.

4.5 Paging

4.5.1 PAGING CONCEPTS

Paging is another type of memory management useful for virtual memory multitasking operating systems. Unlike segmentation which modularizes programs and data into variable length segments, paging divides programs into multiple uniform size pages. Pages bear no direct relation to the logical

structure of a program. While segment selectors can be considered the logical "name" of a program module or data structure, a page most likely corresponds to only a portion of a module or data structure.

By taking advantage of the locality of reference displayed by most programs, only a small number of pages from each active task need be in memory at any one moment.

4.5.2 PAGING ORGANIZATION

4.5.2.1 Page Mechanism

The 486 Microprocessor uses two levels of tables to translate the linear address (from the segmentation unit) into a physical address. There are three components to the paging mechanism of the 486 Microprocessor: the page directory, the page tables, and the page itself (page frame). All memory-resident elements of the 486 Microprocessor paging mechanism are the same size, namely, 4 Kbytes. A uniform size for all of the elements simplifies memory allocation and reallocation schemes, since there is no problem with memory fragmentation. Figure 4.19 shows how the paging mechanism works.

4.5.2.2 Page Descriptor Base Register

CR2 is the Page Fault Linear Address register. It holds the 32-bit linear address which caused the last page fault detected.

CR3 is the Page Directory Physical Base Address Register. It contains the physical starting address of the Page Directory. The lower 12 bits of CR3 are always zero to ensure that the Page Directory is always page aligned. Loading it via a MOV CR3, reg instruction causes the Page Table Entry cache to be flushed, as will a task switch through a TSS which **changes** the value of CR0. (See 4.5.5 **Translation Lookaside Buffer**).

4.5.2.3 Page Directory

The Page Directory is 4 Kbytes long and allows up to 1024 Page Directory Entries. Each Page Directory Entry contains the address of the next level of tables, the Page Tables and information about the page table. The contents of a Page Directory Entry are shown in Figure 4.20. The upper 10 bits of the linear address (A22–A31) are used as an index to select the correct Page Directory Entry.

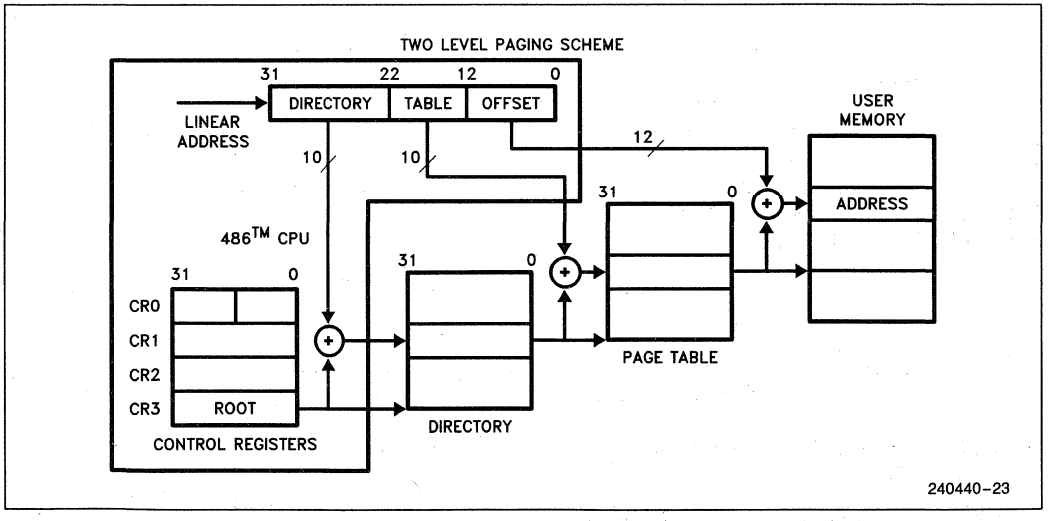


Figure 4.19. Paging Mechanism

31 — 12	11 — 9	8	7	6	5	4	3	2	1	0
PAGE TABLE ADDRESS 31..12	OS RESERVED	0	0	D	A	PCD	PWT	U/S	R/W	P

Figure 4.20. Page Directory Entry (Points to Page Table)

31 12	11 10	9	8	7	6	5	4	3	2	1	0
PAGE FRAME ADDRESS 31..12	OS RESERVED		0	0	D	A	PCD	PWT	U/S	R/W	P

Figure 4.21. Page Table Entry (Points to Page)

4.5.2.4 Page Tables

Each Page Table is 4 Kbytes and holds up to 1024 Page Table Entries. Page Table Entries contain the starting address of the page frame and statistical information about the page (see Figure 4.21). Address bits A12–A21 are used as an index to select one of the 1024 Page Table Entries. The 20 upper-bit page frame address is concatenated with the lower 12 bits of the linear address to form the physical address. Page tables can be shared between tasks and swapped to disks.

4.5.2.5 Page Directory/Table Entries

The lower 12 bits of the Page Table Entries and Page Directory Entries contain statistical information about pages and page tables respectively. The **P** (Present) bit 0 indicates if a Page Directory or Page Table entry can be used in address translation. If P = 1 the entry can be used for address translation if P = 0 the entry can not be used for translation, and all of the other bits are available for use by the software. For example the remaining 31 bits could be used to indicate where on the disk the page is stored.

The **A** (Accessed) bit 5, is set by the 486 Microprocessor for both types of entries before a read or write access occurs to an address covered by the entry. The **D** (Dirty) bit 6 is set to 1 before a write to an address covered by that page table entry occurs. The D bit is undefined for Page Directory Entries. When the P, A and D bits are updated by the 486 Microprocessor, the processor generates a Read-Modify-Write cycle which locks the bus and prevents conflicts with other processors or perpherials. Software which modifies these bits should use the LOCK prefix to ensure the integrity of the page tables in multi-master systems.

The 3 bits marked **OS Reserved** in Figure 4.20 and Figure 4.21 (bits 9–11) are software definable. OSs are free to use these bits for whatever purpose they wish. An example use of the **OS Reserved** bits would be to store information about page aging. By keeping track of how long a page has been in memory since being accessed, an operating system can implement a page replacement algorithm like Least Recently Used.

The (User/Supervisor) U/S bit 2 and the (Read/Write) R/W bit 1 are used to provide protection attributes for individual pages.

4.5.3 PAGE LEVEL PROTECTION (R/W, U/S BITS)

The 486 microprocessor provides a set of protection attributes for paging systems. The paging mechanism distinguishes between two levels of protection: User which corresponds to level 3 of the segmentation based protection, and supervisor which encompasses all of the other protection levels (0, 1, 2).

The R/W and U/S bits are used in conjunction with the WP bit in the flags register (EFLAGS). The 386 microprocessor does not contain the WP bit. The WP bit has been added to the 486 microprocessor to protect read-only pages from supervisor write accesses. The 386 microprocessor allows a read-only page to be written from protection levels 0, 1 or 2. WP=0 is the 386 microprocessor compatible mode. When WP=0 the supervisor can write to a read-only page as defined by the U/S and R/W bits. When WP=1 supervisor access to a read-only page (R/W=0) will cause a page fault (exception 14).

Table 4.4 shows the affect of the WP, U/S and R/W bits on accessing memory. When WP=0, the supervisor can write to pages regardless of the state of the R/W bit. When WP=1 and R/W=0 the supervisor cannot write to a read-only page. A user attempt to access a supervisor only page (U/S=0), or write to a read only page will cause a page fault (exception 14).

The R/W and U/S bits provide protection from user access on a page by page basis since the bits are contained in the Page Table Entry and the Page Directory Table. The U/S and R/W bits in the first level Page Directory Table apply to all entries in the page table pointed to by that directory entry. The U/S and R/W bits in the second level Page Table Entry apply only to the page described by that entry. The most restrictive of the U/S and R/W bits from the Page Directory Table and the Page Table Entry are used to address a page.

Example: If the U/S and R/W bits for the Page Directory entry were 10 (user read/execute) and the

U/S and R/W bits for the Page Table Entry were 01 (no user access at all), the access rights for the page would be 01, the numerically smaller of the two.

Note that a given segment can be easily made read-only for level 0, 1 or 2 via use of segmented protection mechanisms. (Section 4.4 **Protection**).

4.5.4 PAGE CACHEABILITY (PWT AND PCD BITS)

PWT (page write through) and PCD (page cache disable) are two new bits defined in entries in both levels of the page table structure, the Page Directory Table and the Page Table Entry. PCD and PWT control page cacheability and write policy.

PWT controls write policy. PWT = 1 defines a write-through policy for the current page. PWT = 0 allows the possibility of write-back. PWT is ignored internally because the 486 microprocessor has a write-through cache. PWT can be used to control the write policy of a second level cache.

PCD controls cacheability. PCD = 0 enables caching in the on-chip cache. PCD alone does not enable caching, it must be conditioned by the KEN# (cache enable) input signal and the state of the CD (cache disable bit) and NW (no write-through) bits in control register 0 (CR0). When PCD = 1, caching is disabled regardless of the state of KEN#, CD and NW. (See Section 5.0, **On-Chip Cache**).

The state of the PCD and PWT bits are driven out on the PCD and PWT pins during a memory access.

The PWT and PCD bits for a bus cycle are obtained either from control register 3 (CR3), the Page Directory Entry or the Page Table Entry, depending on the type of cycle run. However, when paging is disabled (PG = 0 in CR0) or for cycles which bypass paging (i.e., I/O (input/output) references, INTR (interrupt request) and HALT cycles), the PCD and PWT bits of CR3 are ignored. The i486 CPU assumes PCD = 0 and PWT = 0 and drives these values on the PCD and PWT pins.

When paging is enabled (PG = 1 in CR0), the bits from the page table entry are cached in the translation lookaside buffer (TLB), and are driven any time the page mapped by the TLB entry is referenced. For normal memory cycles run with paging enabled, the PWT and PCD bits are taken from the Page Table Entry. During TLB refresh cycles when the Page Directory and Page Table entries are read, the PWT and PCD bits must be obtained elsewhere. The bits are taken from CR3 when a Page Directory Entry is being read. The bits are taken from the Page Directory Entry when the Page Table Entry is being updated.

The PCD or PWT bits in CR3 are initialized to zero at reset, but can be set to any value by level 0 software.

4.5.5 TRANSLATION LOOKASIDE BUFFER

The 486 Microprocessor paging hardware is designed to support demand paged virtual memory systems. However, performance would degrade substantially if the processor was required to access two levels of tables for every memory reference. To solve this problem, the 486 Microprocessor keeps a cache of the most recently accessed pages, this cache is called the Translation Lookaside Buffer (TLB). The TLB is a four-way set associative 32-entry page table cache. It automatically keeps the most commonly used Page Table Entries in the processor. The 32-entry TLB coupled with a 4K page size, results in coverage of 128 Kbytes of memory addresses. For many common multi-tasking systems, the TLB will have a hit rate of about 98%. This means that the processor will only have to access the two-level page structure on 2% of all memory references. Figure 4.22 illustrates how the TLB complements the 486 Microprocessor's paging mechanism.

Reading a new entry into the TLB (TLB refresh) is a two step process handled by the 486 microprocessor hardware. The sequence of data cycles to perform a TLB refresh are:

Table 4.4. Page Level Protection Attributes

U/S	R/W	WP	User Access	Supervisor Access
0	0	0	None	Read/Write/Execute
0	1	0	None	Read/Write/Execute
1	0	0	Read/Execute	Read/Write/Execute
1	1	0	Read/Write/Execute	Read/Write/Execute
0	0	1	None	Read/Execute
0	1	1	None	Read/Write/Execute
1	0	1	Read/Execute	Read/Execute
1	1	1	Read/Write/Execute	Read/Write/Execute

1. Read the correct Page Directory Entry, as pointed to by the page base register and the upper 10 bits of the linear address. The page base register is in control register 3.

1a. Optionally perform a locked read/write to set the accessed bit in the directory entry. The directory entry will actually get read twice if the 486 microprocessor needs to set any of the bits in the entry. If the page directory entry changes between the first and second reads, the data returned for the second read will be used.

2. Read the correct entry in the Page Table and place the entry in the TLB.

2a. Optionally perform a locked read/write to set the accessed and/or dirty bit in the page table entry. Again, note that the page table entry will actually get read twice if the 486 microprocessor needs to set any of the bits in the entry. Like the directory entry, if the data changes between the first and second read the data returned for the second read will be used.

Note that the directory entry must always be read into the processor, since directory entries are never placed in the paging TLB. Page faults can be signaled from either the page directory read or the page table read. Page directory and page table entries may be placed in the 486 on-chip cache just like normal data.

4.5.6 PAGING OPERATION

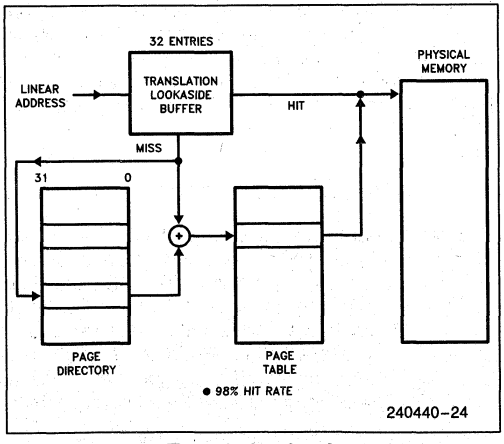


Figure 4.22. Translation Lookaside Buffer

The paging hardware operates in the following fashion. The paging unit hardware receives a 32-bit linear address from the segmentation unit. The upper 20 linear address bits are compared with all 32 entries in the TLB to determine if there is a match. If there is a match (i.e., a TLB hit), then the 32-bit physical address is calculated and will be placed on the address bus.

However, if the page table entry is not in the TLB, the 486 Microprocessor will read the appropriate Page Directory Entry. If P = 1 on the Page Directory Entry indicating that the page table is in memory, then the 486 Microprocessor will read the appropriate Page Table Entry and set the Access bit. If P = 1 on the Page Table Entry indicating that the page is in memory, the 486 Microprocessor will update the Access and Dirty bits as needed and fetch the operand. The upper 20 bits of the linear address, read from the page table, will be stored in the TLB for future accesses. However, if P = 0 for either the Page Directory Entry or the Page Table Entry, then the processor will generate a page fault, an Exception 14.

The processor will also generate an exception 14 page fault, if the memory reference violated the page protection attributes (i.e., U/S or R/W) (e.g., trying to write to a read-only page). CR2 will hold the linear address which caused the page fault. If a second page fault occurs, while the processor is attempting to enter the service routine for the first, then the processor will invoke the page fault (exception 14) handler a second time, rather than the double fault (exception 8) handler. Since Exception 14 is classified as a fault, CS: EIP will point to the instruction causing the page fault. The 16-bit error code pushed as part of the page fault handler will contain status bits which indicate the cause of the page fault.

The 16-bit error code is used by the operating system to determine how to handle the page fault. Figure 4.23a shows the format of the page-fault error code and the interpretation of the bits.

NOTE:
Even though the bits in the error code (U/S, W/R, and P) have similar names as the bits in the Page Directory/Table Entries, the interpretation of the error code bits is different. Figure 4.23b indicates what type of access caused the page fault.

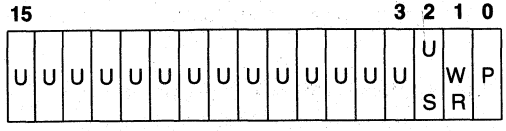


Figure 4.23a. Page Fault Error Code Format

U/S: The U/S bit indicates whether the access causing the fault occurred when the processor was executing in User Mode (U/S = 1) or in Supervisor mode (U/S = 0).

W/R: The W/R bit indicates whether the access causing the fault was a Read (W/R = 0) or a Write (W/R = 1).

P: The P bit indicates whether a page fault was caused by a not-present page (P = 0), or by a page level protection violation (P = 1).

U: UNDEFINED

U/S	W/R	Access Type
0	0	Supervisor* Read
0	1	Supervisor Write
1	0	User Read
1	1	User Write

*Descriptor table access will fault with U/S = 0, even if the program is executing at level 3.

Figure 4.23b. Type of Access Causing Page Fault

4.5.7 OPERATING SYSTEM RESPONSIBILITIES

The 486 Microprocessor takes care of the page address translation process, relieving the burden from an operating system in a demand-paged system. The operating system is responsible for setting up the initial page tables, and handling any page faults. The operating system also is required to invalidate (i.e., flush) the TLB when any changes are made to any of the page table entries. The operating system must reload CR3 to cause the TLB to be flushed.

Setting up the tables is simply a matter of loading CR3 with the address of the Page Directory, and allocating space for the Page Directory and the Page Tables. The primary responsibility of the operating system is to implement a swapping policy and handle all of the page faults.

A final concern of the operating system is to ensure that the TLB cache matches the information in the paging tables. In particular, any time the operating system sets the P present bit of page table entry to zero, the TLB must be flushed. Operating systems may want to take advantage of the fact that CR3 is stored as part of a TSS, to give every task or group of tasks its own set of page tables.

4.6 Virtual 8086 Environment

4.6.1 EXECUTING 8086 PROGRAMS

The 486 Microprocessor allows the execution of 8086 application programs in both Real Mode and in the Virtual 8086 Mode (Virtual Mode). Of the two methods, Virtual 8086 Mode offers the system designer the most flexibility. The Virtual 8086 Mode allows the execution of 8086 applications, while still allowing the system designer to take full advantage of the 486 Microprocessor protection mechanism. In particular, the 486 Microprocessor allows the simultaneous execution of 8086 operating systems and its applications, and a 486 Microprocessor operating system and both 80286 and 486 Microprocessor applications. Thus, in a multi-user 486 Microprocessor computer, one person could be running an MS-DOS spreadsheet, another person using MS-DOS, and a third person could be running multiple Unix utilities and applications. Each person in this scenario would believe that he had the computer completely to himself. Figure 4.24 illustrates this concept.

4.6.2 VIRTUAL 8086 MODE ADDRESSING MECHANISM

One of the major differences between 486 Microprocessor Real and Protected modes is how the segment selectors are interpreted. When the processor is executing in Virtual 8086 Mode the segment registers are used in an identical fashion to Real Mode. The contents of the segment register is shifted left 4 bits and added to the offset to form the segment base linear address.

The 486 Microprocessor allows the operating system to specify which programs use the 8086 style address mechanism, and which programs use Protected Mode addressing, on a per task basis. Through the use of paging, the one megabyte address space of the Virtual Mode task can be mapped to anywhere in the 4 gigabyte linear address space of the 486 Microprocessor. Like Real Mode, Virtual Mode effective addresses (i.e., segment offsets) that exceed 64 Kbyte will cause an exception 13. However, these restrictions should not prove to be important, because most tasks running in Virtual 8086 Mode will simply be existing 8086 application programs.

4.6.3 PAGING IN VIRTUAL MODE

The paging hardware allows the concurrent running of multiple Virtual Mode tasks, and provides protection and operating system isolation. Although it is not strictly necessary to have the paging hardware enabled to run Virtual Mode tasks, it is needed in order to run multiple Virtual Mode tasks or to relocate the address space of a Virtual Mode task to physical address space greater than one megabyte.

The paging hardware allows the 20-bit linear address produced by a Virtual Mode program to be divided into up to 256 pages. Each one of the pages can be located anywhere within the maximum 4 gigabyte physical address space of the 486 Microprocessor. In addition, since CR3 (the Page Directory Base Register) is loaded by a task switch, each Virtual Mode task can use a different mapping scheme to map pages to different physical locations.

Finally, the paging hardware allows the sharing of the 8086 operating system code between multiple 8086 applications. Figure 4.24 shows how the 486 Microprocessor paging hardware enables multiple 8086 programs to run under a virtual memory demand paged system.

4.6.4 PROTECTION AND I/O PERMISSION BITMAP

All Virtual 8086 Mode programs execute at privilege level 3, the level of least privilege. As such, Virtual 8086 Mode programs are subject to all of the protection checks defined in Protected Mode. (This is different from Real Mode which implicitly is executing at privilege level 0, the level of greatest privilege.) Thus, an attempt to execute a privileged instruction when in Virtual 8086 Mode will cause an exception 13 fault.

The following are privileged instructions, which may be executed only at Privilege Level 0. Therefore, attempting to execute these instructions in Virtual 8086 Mode (or anytime CPL > 0) causes an exception 13 fault:

```
LIDT;  MOV DRn,reg;  MOV reg,DRn;
LGDT;  MOV TRn,reg;  MOV reg,TRn;
LMSW;  MOV CRn,reg;  MOV reg,CRn.
CLTS;
HLT;
```

Several instructions, particularly those applying to the multitasking model and protection model, are available only in Protected Mode. Therefore, attempting to execute the following instructions in Real Mode or in Virtual 8086 Mode generates an exception 6 fault:

```
LTR;   STR;
LLDT;  SLDT;
LAR;   VERR;
LSL;   VERW;
ARPL.
```

The instructions which are IOPL-sensitive in Protected Mode are:

```
IN;        STI;
OUT;       CLI
INS;
OUTS;
REP INS;
REP OUTS;
```

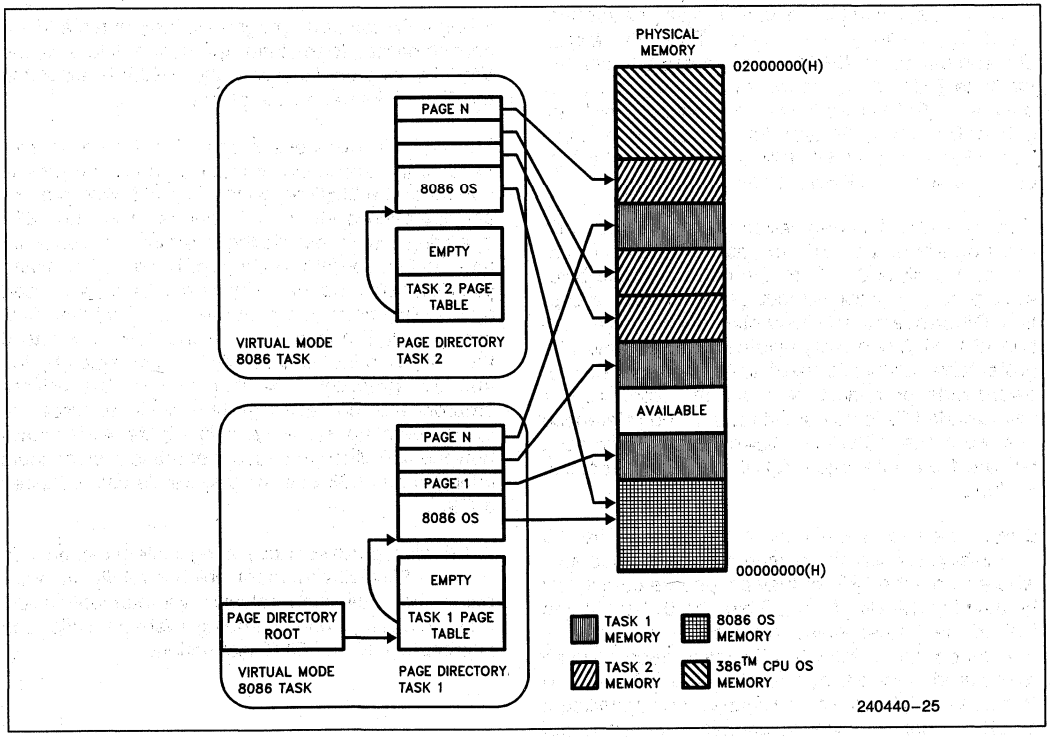


Figure 4.24. Virtual 8086 Environment Memory Management

In Virtual 8086 Mode, a slightly different set of instructions are made IOPL-sensitive. The following instructions are IOPL-sensitive in Virtual 8086 Mode:

```
INT n;    STI;
PUSHF;    CLI;
POPF;     IRET
```

The PUSHF, POPF, and IRET instructions are IOPL-sensitive in Virtual 8086 Mode only. This provision allows the IF flag (interrupt enable flag) to be virtualized to the Virtual 8086 Mode program. The INT n software interrupt instruction is also IOPL-sensitive in Virtual 8086 Mode. Note, however, that the INT 3 (opcode 0CCH), INTO, and BOUND instructions are not IOPL-sensitive in Virtual 8086 mode (they aren't IOPL sensitive in Protected Mode either).

Note that the I/O instructions (IN, OUT, INS, OUTS, REP INS, and REP OUTS) are **not** IOPL-sensitive in Virtual 8086 mode. Rather, the I/O instructions become automatically sensitive to the **I/O Permission Bitmap** contained in the **486 Microprocessor Task State Segment**. The I/O Permission Bitmap, automatically used by the 486 Microprocessor in Virtual 8086 Mode, is illustrated by Figures 4.15a and 4.15b.

The I/O Permission Bitmap can be viewed as a 0–64 Kbit bit string, which begins in memory at offset Bit_Map_Offset in the current TSS. Bit_Map_Offset must be ≤ DFFFH so the entire bit map and the byte FFH which follows the bit map are all at offsets ≤ FFFFH from the TSS base. The 16-bit pointer Bit_Map_Offset (15:0) is found in the word beginning at offset 66H (102 decimal) from the TSS base, as shown in Figure 4.15a.

Each bit in the I/O Permission Bitmap corresponds to a single byte-wide I/O port, as illustrated in Figure 4.15a. If a bit is 0, I/O to the corresponding byte-wide port can occur without generating an exception. Otherwise the I/O instruction causes an exception 13 fault. Since every byte-wide I/O port must be protectable, all bits corresponding to a word-wide or dword-wide port must be 0 for the word-wide or dword-wide I/O to be permitted. If all the referenced bits are 0, the I/O will be allowed. If any referenced bits are 1, the attempted I/O will cause an exception 13 fault.

Due to the use of a pointer to the base of the I/O Permission Bitmap, the bitmap may be located anywhere within the TSS, or may be ignored completely by pointing the Bit_Map_Offset (15:0) beyond the limit of the TSS segment. In the same manner, only a small portion of the 64K I/O space need have an associated map bit, by adjusting the TSS limit to truncate the bitmap. This eliminates the commitment of 8K of memory when a complete bitmap is not required, while allowing the fully general case if desired.

EXAMPLE OF BITMAP FOR I/O PORTS 0–255: Setting the TSS limit to {bit_Map_Offset + 31 +1**} [** see note below] will allow a 32-byte bitmap for the I/O ports #0–255, plus a terminator byte of all 1's [** see note below]. This allows the I/O bitmap to control I/O Permission to I/O port 0–255 while causing an exception 13 fault on attempted I/O to any I/O port 80256 through 65,565.

****IMPORTANT IMPLEMENTATION NOTE:** Beyond the last byte of I/O mapping information in the I/O Permission Bitmap **must** be a byte containing all 1's. The byte of all 1's must be within the limit of the 486 Microprocessor TSS segment (see Figure 4.15a).

4.6.5 INTERRUPT HANDLING

In order to fully support the emulation of an 8086 machine, interrupts in Virtual 8086 Mode are handled in a unique fashion. When running in Virtual Mode all interrupts and exceptions involve a privilege change back to the host 486 Microprocessor operating system. The 486 Microprocessor operating system determines if the interrupt comes from a Protected Mode application or from a Virtual Mode program by examining the VM bit in the EFLAGS image stored on the stack.

When a Virtual Mode program is interrupted and execution passes to the interrupt routine at level 0, the VM bit is cleared. However, the VM bit is still set in the EFLAG image on the stack.

The 486 Microprocessor operating system in turn handles the exception or interrupt and then returns control to the 8086 program. The 486 Microprocessor operating system may choose to let the 8086 operating system handle the interrupt or it may emulate the function of the interrupt handler. For example, many 8086 operating system calls are accessed by PUSHing parameters on the stack, and then executing an INT n instruction. If the IOPL is set to 0 then all INT n instructions will be intercepted by the 486 Microprocessor operating system. The 486 Microprocessor operating system could emulate the 8086 operating system's call. Figure 4.25 shows how the 486 Microprocessor operating system could intercept an 8086 operating system's call to "Open a File".

A 486 Microprocessor operating system can provide a Virtual 8086 Environment which is totally transparent to the application software via intercepting and then emulating 8086 operating system's calls, and intercepting IN and OUT instructions.

4.6.6 ENTERING AND LEAVING VIRTUAL 8086 MODE

Virtual 8086 mode is entered by executing an IRET instruction (at CPL = 0), or Task Switch (at any CPL) to a 486 Microprocessor task whose 486 Microprocessor TSS has a FLAGS image containing a 1 in the VM bit position while the processor is executing in Protected Mode. That is, one way to enter Virtual 8086 mode is to switch to a task with a 486 Microprocessor TSS that has a 1 in the VM bit in the EFLAGS image. The other way is to execute a 32-bit IRET instruction at privilege level 0, where the stack has a 1 in the VM bit in the EFLAGS image. POPF does not affect the VM bit, even if the processor is in Protected Mode or level 0, and so cannot be used to enter Virtual 8086 Mode. PUSHF always pushes a 0 in the VM bit, even if the processor is in Virtual 8086 Mode, so that a program cannot tell if it is executing in REAL mode, or in Virtual 8086 mode.

The VM bit can be set by executing an IRET instruction only at privilege level 0, or by any instruction or Interrupt which causes a task switch in Protected Mode (with VM = 1 in the new FLAGS image), and can be cleared only by an interrupt or exception in Virtual 8086 Mode. IRET and POPF instructions executed in REAL mode or Virtual 8086 mode will not change the value in the VM bit.

The transition out of virtual 8086 mode to 486 Microprocessor protected mode occurs only on receipt of an interrupt or exception (such as due to a sensitive instruction). In Virtual 8086 mode, all interrupts and exceptions vector through the protected mode IDT, and enter an interrupt handler in protected 486 Microprocessor mode. That is, as part of interrupt processing, the VM bit is cleared.

Because the matching IRET must occur from level 0, if an Interrupt or Trap Gate is used to field an interrupt or exception out of Virtual 8086 mode, the Gate must perform an inter-level interrupt only to level 0. Interrupt or Trap Gates through conforming segments, or through segments with DPL > 0, will raise a GP fault with the CS selector as the error code.

4.6.6.1 Task Switches To/From Virtual 8086 Mode

Tasks which can execute in virtual 8086 mode must be described by a TSS with the new 486 Microprocessor format (TYPE 9 or 11 descriptor).

A task switch out of virtual 8086 mode will operate exactly the same as any other task switch out of a task with a 486 Microprocessor TSS. All of the programmer visible state, including the FLAGS register with the VM bit set to 1, is stored in the TSS. The segment registers in the TSS will contain 8086 segment base values rather than selectors.

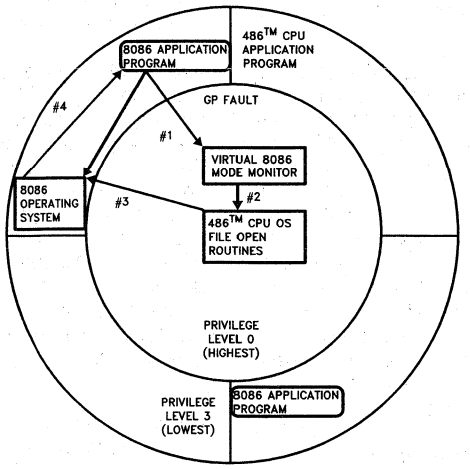
A task switch into a task described by a 486 Microprocessor TSS will have an additional check to determine if the incoming task should be resumed in virtual 8086 mode. Tasks described by 80286 format TSSs cannot be resumed in virtual 8086 mode, so no check is required there (the FLAGS image in 80286 format TSS has only the low order 16 FLAGS bits). Before loading the segment register images from a 486 Microprocessor TSS, the FLAGS image is loaded, so that the segment registers are loaded from the TSS image as 8086 segment base values. The task is now ready to resume in virtual 8086 execution mode.

4.6.6.2 Transitions Through Trap and Interrupt Gates, and IRET

A task switch is one way to enter or exit virtual 8086 mode. The other method is to exit through a Trap or Interrupt gate, as part of handling an interrupt, and to enter as part of executing an IRET instruction. The transition out must use a 486 Microprocessor Trap Gate (Type 14), or 486 Microprocessor Interrupt Gate (Type 15), which must point to a non-conforming level 0 segment (DPL = 0) in order to permit the trap handler to IRET back to the Virtual 8086 program. The Gate must point to a non-conforming level 0 segment to perform a level switch to level 0 so that the matching IRET can change the VM bit. 486 Microprocessor gates must be used, since 80286 gates save only the low 16 bits of the FLAGS register, so that the VM bit will not be saved on transitions through the 80286 gates. Also, the 16-bit IRET (presumably) used to terminate the 80286 interrupt handler will pop only the lower 16 bits from FLAGS, and will not affect the VM bit. The action taken for a 486 Microprocessor Trap or Interrupt gate if an interrupt occurs while the task is executing in virtual 8086 mode is given by the following sequence.

(1) Save the FLAGS register in a temp to push later. Turn off the VM and TF bits, and if the interrupt is serviced by an Interrupt Gate, turn off IF also.

(2) Interrupt and Trap gates must perform a level switch from 3 (where the VM86 program executes) to level 0 (so IRET can return). This process involves a stack switch to the stack given in the TSS for privilege level 0. Save the Virtual 8086 Mode SS and ESP registers to push in a later step. The segment register load of SS will be done as a Protected Mode segment load, since the VM bit was turned off above.



8086 Application makes "Open File Call" → causes
General Protection Fault (Arrow #1)
Virtual 8086 Monitor intercepts call. Calls 486™ CPU OS (Arrow #2)
486™ CPU OS opens file returns control to 8086 OS (Arrow #3)
8086 OS returns control to application. (Arrow #4)
Transparent to Application

Figure 4.25. Virtual 8086 Environment Interrupt and Call Handling

(3) Push the 8086 segment register values onto the new stack, in the order: GS, FS, DS, ES. These are pushed as 32-bit quantities, with undefined values in the upper 16 bits. Then load these 4 registers with null selectors (0).

(4) Push the old 8086 stack pointer onto the new stack by pushing the SS register (as 32-bits, high bits undefined), then pushing the 32-bit ESP register saved above.

(5) Push the 32-bit FLAGS register saved in step 1.

(6) Push the old 8086 instruction pointer onto the new stack by pushing the CS register (as 32-bits, high bits undefined), then pushing the 32-bit EIP register.

(7) Load up the new CS:EIP value from the interrupt gate, and begin execution of the interrupt routine in protected 486 Microprocessor mode.

The transition out of virtual 8086 mode performs a level change and stack switch, in addition to changing back to protected mode. In addition, all of the 8086 segment register images are stored on the stack (behind the SS:ESP image), and then loaded with null (0) selectors before entering the interrupt handler. This will permit the handler to safely save and restore the DS, ES, FS, and GS registers as 80286 selectors. This is needed so that interrupt handlers which don't care about the mode of the interrupted program can use the same prolog and epilog code for state saving (i.e., push all registers in prolog, pop all in epilog) regardless of whether or not a "native" mode or Virtual 8086 mode program was interrupted. Restoring null selectors to these registers before executing the IRET will not cause a trap in the interrupt handler. Interrupt routines which expect values in the segment registers, or return values in segment registers will have to obtain/return values from the 8086 register images pushed onto the new stack. They will need to know the mode of the interrupted program in order to know where to find/return segment registers, and also to know how to interpret segment register values.

The IRET instruction will perform the inverse of the above sequence. Only the extended 486 Microprocessors IRET instruction (operand size=32) can be used, and must be executed at level 0 to change the VM bit to 1.

(1) If the NT bit in the FLAGs register is on, an inter-task return is performed. The current state is stored in the current TSS, and the link field in the current TSS is used to locate the TSS for the interrupted task which is to be resumed.

Otherwise, continue with the following sequence.

(2) Read the FLAGS image from SS:8[ESP] into the FLAGS register. This will set VM to the value active in the interrupted routine.

(3) Pop off the instruction pointer CS:EIP. EIP is popped first, then a 32-bit word is popped which contains the CS value in the lower 16 bits. If

VM=0, this CS load is done as a protected mode segment load. If VM=1, this will be done as an 8086 segment load.

(4) Increment the ESP register by 4 to bypass the FLAGS image which was "popped" in step 1.

(5) If VM=1, load segment registers ES, DS, FS, and GS from memory locations SS:[ESP+8], SS:[ESP+12], SS:[ESP+16], and SS:[ESP+20], respectively, where the new value of ESP stored in step 4 is used. Since VM=1, these are done as 8086 segment register loads.

Else if VM=0, check that the selectors in ES, DS, FS, and GS are valid in the interrupted routine. Null out invalid selectors to trap if an attempt is made to access through them.

(6) If (RPL(CS) > CPL), pop the stack pointer SS:ESP from the stack. The ESP register is popped first, followed by 32-bits containing SS in the lower 16 bits. If VM=0, SS is loaded as a protected mode segment register load. If VM=1, an 8086 segment register load is used.

(7) Resume execution of the interrupted routine. The VM bit in the FLAGS register (restored from the interrupt routine's stack image in step 1) determines whether the processor resumes the interrupted routine in Protected mode of Virtual 8086 mode.

5.0 ON-CHIP CACHE

To meet its performance goals the 486 microprocessor contains an eight Kbyte cache. The cache is software transparent to maintain binary compatibility with previous generations of the x86 architecture.

The on-chip cache has been designed for maximum flexibility and performance. The cache has several operating modes offering flexibility during program execution and debugging. Memory areas can be defined as non-cacheable by software and external hardware. Protocols for cache line invalidations and replacement are implemented in hardware, easing system design.

5.1 Cache Organization

The on-chip cache is a unified code and data cache. The cache is used for both instruction and data accesses and acts on physical addresses.

The cache organization is 4-way set associative and each line is 16 bytes wide. The eight Kbytes of cache memory are logically organized as 128 sets, each containing four lines.

The cache memory is physically split into four 2-Kbyte blocks each containing 128 lines (see Figure 5.1). Associated with each 2-Kbyte block are 128 21-bit tags. There is a valid bit for each line in the cache. Each line in the cache is either valid or not valid. There are no provisions for partially valid lines.

5

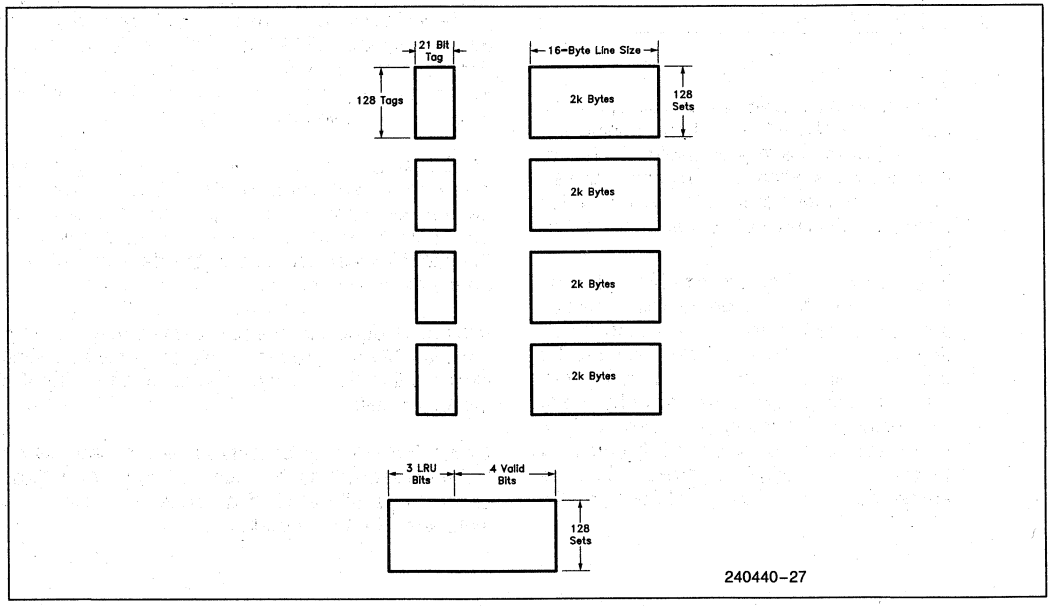


Figure 5.1. On-Chip Cache Physical Organization

The write strategy of on-chip cache is write-through. All writes will drive an external write bus cycle in addition to writing the information to the internal cache if the write was a cache hit. A write to an address not contained in the internal cache will only be written to external memory. Cache allocations are not made on write misses.

5.2 Cache Control

Control of the cache is provided by the CD and NW bits in CR0. CD enables and disables the cache. NW controls memory write-through and invalidates.

The CD and NW bits define four operating modes of the on-chip cache as given in Table 5.1. These modes provide flexibility in how the on-chip cache is used.

The CD and NW bits define four operating modes of the on-chip code and data cache, as given in the following table:

Table 5.1. Cache Operating Modes

CD	NW	Operating Mode
1	1	Cache fills disabled, write-through and invalidates disabled
1	0	Cache fills disabled, write-through and invalidates enabled
0	1	INVALID. IF CR0 is loaded with this configuration of bits, a GP fault with error code of 0 is raised.
0	0	Cache fills enabled, write-through and invalidates enabled

CD = 1, NW = 1

The cache is completely disabled by setting CD = 1 and NW = 1 and then flushing the cache. This mode may be useful for debugging programs where it is important to see all memory cycles at the pins. Writes which hit in the cache will not appear on the external bus.

It is possible to use the on-chip cache as fast static RAM by "pre-loading" certain memory areas into the cache and then setting CD = 1 and NW = 1. Pre-loading can be done by careful choice of memory references with the cache turned on or by use of the testability functions (see Section 8.2). When the cache is turned off the memory mapped by the cache is "frozen" into the cache since fills and invalidates are disabled.

CD = 1, NW = 0

Cache fills are disabled but write-throughs and invalidates are enabled. This mode is the same as if the KEN# pin was strapped HIGH disabling cache fills. Write-throughs and invalidates may still occur to keep the cache valid. This mode is useful if the software must disable the cache for a short period of time, and then re-enable it without flushing the original contents.

CD = 0, NW = 1

INVALID. If CR0 is loaded with this bit configuration, a General Protection fault with error code of 0 is raised. Note that this mode would imply a non-transparent write-back cache. A future processor may define this combination of bits to implement a write-back cache.

CD = 0, NW = 0

This is the normal operating mode.

Completely disabling the cache is a two step process. First CD and NW must be set to 1 and then the cache must be flushed. If the cache is not flushed, cache hits on reads will still occur and data will be read from the cache.

5.3 Cache Line Fills

Any area of memory can be cached in the 486 microprocessor. Non-cacheable portions of memory can be defined by the external system or by software. The external system can inform the 486 microprocessor that a memory address is non-cacheable by returning the KEN# pin inactive during a memory access (refer to Section 7.2.3). Software can prevent certain pages from being cached by setting the PCD bit in the page table entry.

A read request can be generated from program operation or by an instruction pre-fetch. The data will be supplied from the on-chip cache if a cache hit occurs on the read address. If the address is not in the cache, a read request for the data is generated on the external bus.

If the read request is to a cacheable portion of memory, the 486 microprocessor initiates a cache line fill. During a line fill a 16-byte line is read into the 486 microprocessor.

Cache fills will only be generated for read misses. Write misses will never cause a line in the internal cache to be allocated. If a cache hit occurs on a write, the line will be updated.

Cache line fills can be performed over 8- and 16-bit busses using the dynamic bus sizing feature. Refer to Section 7.1.3 for a description of dynamic bus sizing.

Refer to Section 7.2.3 for further information on cacheable cycles.

5.4 Cache Line Invalidations

The 486 microprocessor contains both a hardware and software mechanism for invalidating lines in its internal cache. Cache line invalidations are needed to keep the 486 microprocessor's cache contents consistent with external memory.

Refer to Section 7.2.8 for further information on cache line invalidations.

5.5 Cache Replacement

When a line needs to be placed in its internal cache the 486 microprocessor first checks to see if there is a non-valid line in the set that can be replaced. If all four lines in the set are valid, a pseudo least-recently-used mechanism is used to determine which line should be replaced.

A valid bit is associated with each line in the cache. When a line needs to be placed in a set, the four valid bits are checked to see if there is a non-valid line that can be replaced. If a non-valid line is found, that line is marked for replacement.

The four lines in the set are labeled I0, I1, I2, and I3. The order in which the valid bits are checked during an invalidation is I0, I1, I2 and I3. All valid bits are cleared when the processor is reset or when the cache is flushed.

Replacement in the cache is handled by a pseudo least recently used (LRU) mechanism when all four lines in a set are valid. Three bits, B0, B1 and B2, are defined for each of the 128 sets in the cache. These bits are called the LRU bits. The LRU bits are updated for every hit or replace in the cache.

If the most recent access to the set was to I0 or I1, B0 is set to 1. B0 is set to 0 if the most recent access was to I2 or I3. If the most recent access to I0:I1 was to I0, B1 is set to 1, else B1 is set to 0. If the most recent access to I2:I3 was to I2, B2 is set to 1, else B2 is set to 0.

The pseudo LRU mechanism works in the following manner. When a line must be replaced, the cache will first select which of I0:I1 and I2:I3 was least recently used. Then the cache will determine which of the two lines was least recently used and mark it for replacement. This decision tree is shown in Figure 5.2. When the processor is reset or when the cache is flushed all 128 sets of three LRU bits are set to 0.

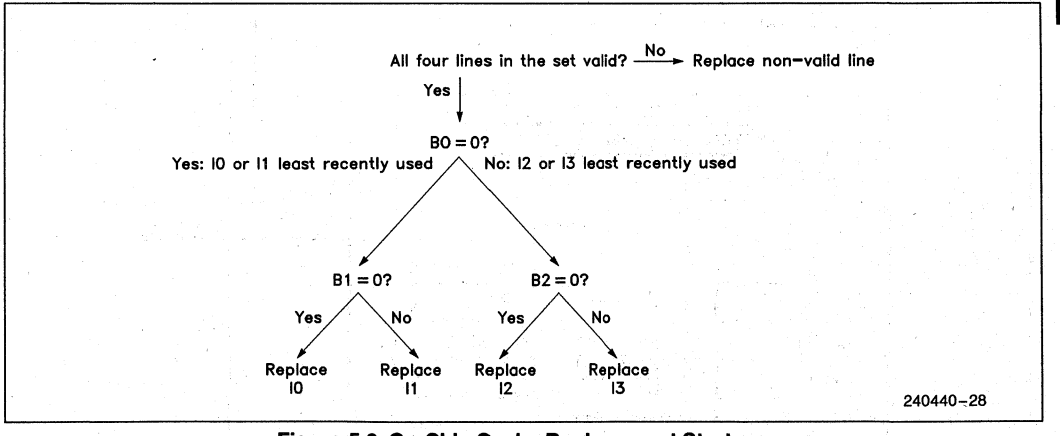


Figure 5.2. On-Chip Cache Replacement Strategy

5.6 Page Cacheability

Two bits for cache control, PWT and PCD, are defined in the page table and page directory entries. The state of these bits are driven out on the PWT and PCD pins during memory access cycles.

The PWT bit controls write policy for second level caches used with the 486 microprocessor. Setting PWT = 1 defines a write-through policy for the current page while PWT = 0 allows the possibility of write-back. The state of PWT is ignored internally by the 486 microprocessor since the on-chip cache is write through.

The PCD bit controls cacheability on a page by page basis. The PCD bit is internally ANDed with the KEN# signal to control cacheability on a cycle by cycle basis (see Figure 5.3). PCD = 0 enables caching while PCD = 1 forbids it. Note that cache fills are enabled when PCD = 0 AND KEN# = 0. This logical AND is implemented physically with a NOR gate.

The state of the PCD bit in the page table entry is driven on the PCD pin when a page in external memory is accessed. The state of the PCD pin informs the external system of the cacheability of the requested information. The external system then returns KEN# telling the 486 microprocessor if the area is cacheable. The 486 microprocessor initiates a cache line fill if PCD and KEN# indicate that the requested information is cacheable.

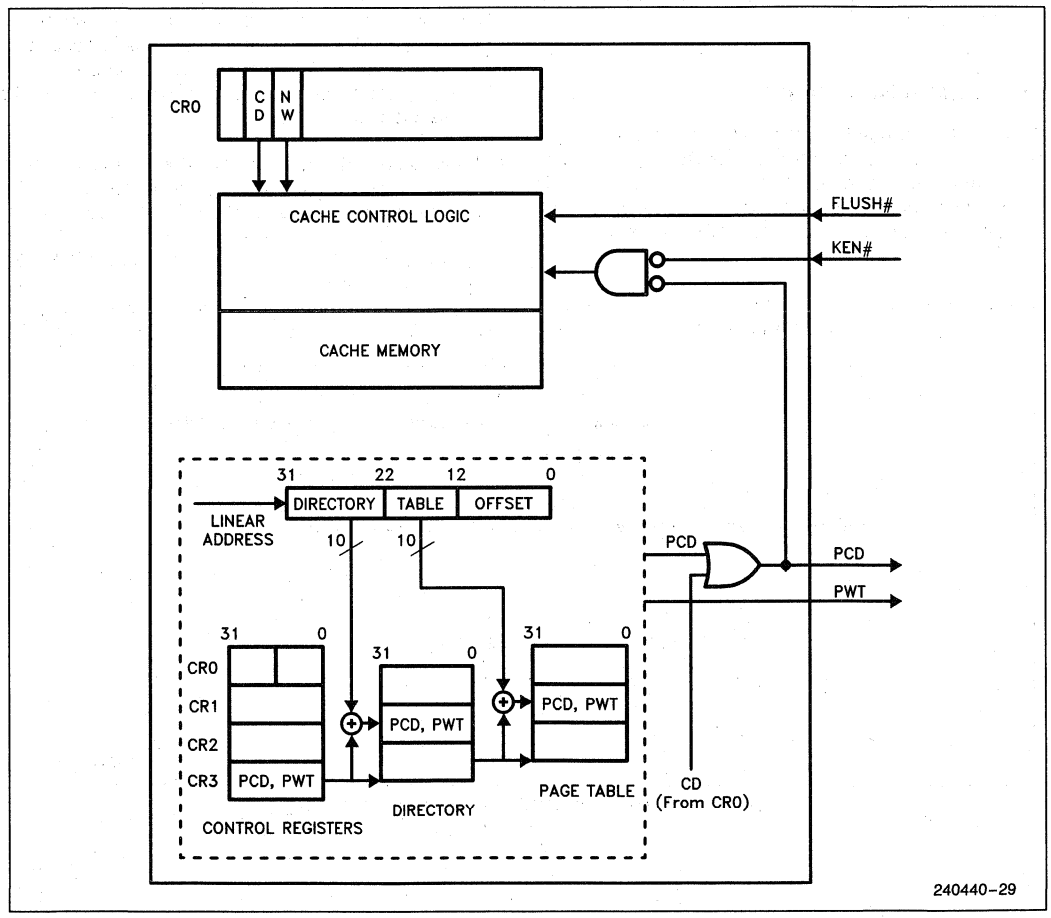


Figure 5.3. Page Cacheability

The PCD bit is masked with the CD (cache disable) bit in control register 0 to determine the state of the PCD pin. If CD=1 the 486 microprocessor forces the PCD pin HIGH. If CD=0 the PCD pin is driven with the value for the page table entry/directory. See Figure 5.3.

The PWT and PCD bits for a bus cycle are obtained from either CR3, the page directory or page table entry. These bits are assumed to be zero during real mode, whenever paging is disabled, or for cycles that bypass paging, (I/O references, interrupt acknowledge and Halt cycles), the PWT and PCD bits are taken from CR3. These bits are initialized to 0 on reset, but can be set to any value by level 0 software.

When paging is enabled, the bits from the page table entry are cached in the TLB, and are driven any time the page mapped by the TLB entry is referenced. For normal memory cycles, PWT and PCD are taken from the page table entry. During TLB refresh cycles where the page table and directory entries are read, the PWT and PCD bits must be obtained elsewhere. During page table updates the bits are obtained from the page directory. When the page directory is updated the bits are obtained from CR3.

5.7 Cache Flushing

The on-chip cache can be flushed by external hardware or by software instructions. Flushing the cache clears all valid bits for all lines in the cache. The cache is flushed when external hardware asserts the FLUSH# pin.

The flush pin needs to be asserted for one clock if driven synchronously or for two clocks if driven asynchronously. The flush input is asynchronous but setup and hold times must be met. The flush pin should be deasserted after the cache flush is complete. Failure to deassert the pin will cause execution to stop as the processor will be repeatedly flushing the cache. If external hardware activates flush in response to an I/O write, flush must be asserted for at least two clocks prior to ready being returned for the I/O write. This ensures that the flush completes before the CPU begins execution of the instruction following the OUT instruction.

Flush is recognized during HOLD just like EADS#.

The instructions INVD and WBINVD cause the on-cache to be flushed. External caches connected to the 486 microprocessor are signalled to flush their contents when these instructions are executed.

WBINVD will cause an external write-back cache to write back dirty lines before flushing its contents. The external cache is signalled using the bus cycle definition pins and the byte enables (refer to Section 6.2.5 for the bus cycle definition pins and Section 7.2.11 for special bus cycles). Refer to the 486 microprocessor programmers reference manual for detailed instruction definitions.

The results of the INVD and WBINVD instructions are identical for the operation of the 486 microprocessor's on-chip cache since the cache is write-through. Note that the INVD and WBINVD instructions are machine dependent. Future members of the 486 microprocessor family may change the definition of this instruction.

5.8 Caching Translation Lookaside Buffer Entries

The 486 microprocessor contains an integrated paging unit with a translation lookaside buffer (TLB). The TLB contains 32 entries. The TLB has been enhanced over the 386 microprocessor's TLB by upgrading the replacement strategy to a pseudo-LRU (least recently used) algorithm. The pseudo-LRU replacement algorithm is the same as that used in the on-chip cache.

The paging TLB operation is automatic whenever paging is enabled. The TLB contains the most recently used page table entries. A page table entry translates the linear address pointing to a particular page to the physical address where the page is stored in memory (refer to Section 4.5, **Paging**).

The paging unit will look up the linear address in the TLB in response to an internal bus request. The corresponding physical address is passed on to the on-chip cache or the external bus (in the event of a cache miss) when the linear address is present in the TLB.

The paging unit will access the page tables in external memory if the linear address is not in the TLB. The required page table entry will be read into the TLB and then the cache or bus cycle for the actual data will take place. The process of reading a new page table entry into the TLB is called a TLB refresh.

A TLB refresh is a two step process. The paging unit must first read the page directory entry which points to the appropriate page table. The page table entry to be stored in the TLB is then read from the page table. Control register 3 (CR3) points to the base of the page directory table.

The 486 microprocessor will allow page directory and page table entries (returned during TLB refreshes) to be stored in the on-chip cache. Setting the PCD bits in CR3 and the page directory entry to 1 will prevent the page directory and page table entries from being stored in the on-chip cache (see Section 5.6, **Page Cacheability**).

6.0 HARDWARE INTERFACE

6.1 Introduction

The 486 microprocessor bus has been designed to be similar to the 386 microprocessor bus whenever possible. Several new features have been added to the 486 microprocessor bus resulting in increased performance and functionality. New features include a 1X clock, a burst bus mechanism for high-speed internal cache fills, a cache line invalidation mechanism, enhanced bus arbitration capabilities, a BS8# bus sizing mechanism and parity support.

The 486 microprocessor is driven by a 1X clock as opposed to a 2X clock in the 386 microprocessor. A 25 MHz 486 microprocessor uses a 25 MHz clock in contrast to a 25 MHz 386 microprocessor which requires a 50 MHz clock. A 1X clock allows simpler system design by cutting in half the clock speed required in the external system.

Like the 386 microprocessor, the 486 microprocessor has separate parallel busses for data and addresses. The bidirectonal data bus is 32 bits in width. The address bus consists of two components: 30 address lines (A2–A31) and 4 byte enable lines (BE0#–BE3#). The address bus addresses external memory in the same manner as the 386 microprocessor: The address lines form the upper 30 bits of the address and the byte enables select individual bytes within a 4 byte location. The address lines are bidirectional for use in cache line invalidations.

The 486 microprocessor's burst bus mechanism enables high-speed cache fills from external memory. Burst cycles can strobe data into the processor at a rate of one item every clock. Non-burst cycles have a maximum rate of one item every two clocks. Burst cycles are not limited to cache fills: all bus cycles requiring more than a single data cycle can be bursted.

The 486 microprocessor has a bus hold feature similar to that of the 386 microprocessor. During bus hold, the 486 microprocessor relinquishes control of the local bus by floating its address, data and control busses.

The 486 microprocessor has an address hold feature in addition to bus hold. During address hold only the address bus is floated, the data and control busses can remain active. Address hold is used for cache line invalidations.

Ahead is a brief description of the 486 microprocessor input and output signals arranged by functional

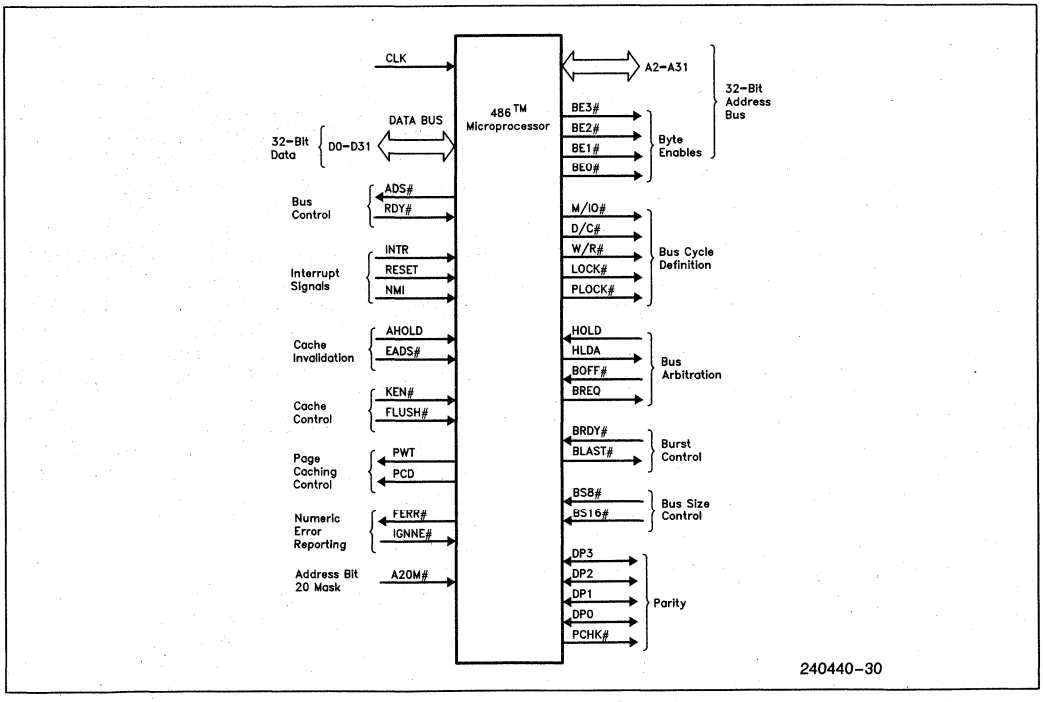


Figure 6.1. Functional Signal Groupings

groups. Before beginning the signal descriptions a few terms need to be defined. The # symbol at the end of a signal name indicates the active, or asserted, state occurs when the signal is at a low voltage. When a # is not present after the signal name, the signal is active at the high voltage level. The term "ready" is used to indicate that the cycle is terminated with RDY# or BRDY#.

Section 6 and 7 will discuss bus cycles and data cycles. A bus cycle is at least two clocks long and begins with ADS# active in the first clock and ready active in the last clock. Data is transferred to or from the 486 microprocessor during a data cycle. A bus cycle contains one or more data cycles.

6.2 Signal Descriptions

6.2.1 CLOCK (CLK)

CLK provides the fundamental timing and the internal operating frequency for the 486 microprocessor. All external timing parameters are specified with respect to the rising edge of CLK.

The 486 microprocessor can operate over a wide frequency range but CLK's frequency cannot change rapidly while RESET is inactive. CLK's frequency must be stable for proper chip operation since a single edge of CLK is used internally to generate two phases. CLK only needs TTL levels for proper operation. Figure 6.2 illustrates the CLK waveform.

6.2.2 Address Bus (A31–A2, BE0#–BE3#)

A31–A2 and BE0#–BE3# form the address bus and provide physical memory and I/O port addresses. The 486 microprocessor is capable of addressing 4 gigabytes of physical memory space (00000000H through FFFFFFFFH), and 64 Kbytes of I/O address space (00000000H through 0000FFFFH). A31–A2 identify addresses to a 4-byte location. BE0#–BE3# identify which bytes within the 4-byte location are involved in the current transfer.

Addresses are driven back into the 486 microprocessor over A31–A4 during cache line invalidations. The address lines are active HIGH. When used as inputs into the processor, A31–A4 must meet the setup and hold times, t_{22} and t_{23}. A31–A2 are not driven during bus or address hold.

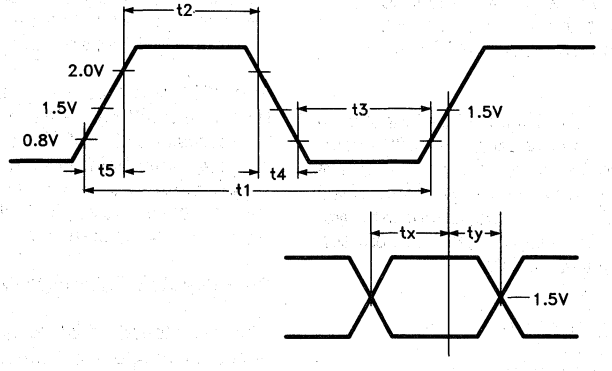
The byte enable outputs, BE0#–BE3#, determine which bytes must be driven valid for read and write cycles to external memory.

BE3# applies to D24–D31
BE2# applies to D16–D23
BE1# applies to D8–D15
BE0# applies to D0–D7

BE0#–BE3# can be decoded to generate A0, A1 and BHE# signals used in 8- and 16-bit systems (see Table 7.5). BE0#–BE3# are active LOW and are not driven during bus hold.

6.2.3 DATA LINES (D31–D0)

The bidirectional lines, D31–D0, form the data bus for the 486 microprocessor. D0–D7 define the least significant byte and D24–D31 the most significant byte. Data transfers to 8- or 16-bit devices is possible using the data bus sizing feature controlled by the BS8# or BS16# input pins.



tx = input setup times
ty = input hold times, output float, valid and hold times

Figure 6.2. CLK waveform

D31-D0 are active HIGH. For reads, D31-D0 must meet the setup and hold times, t_{22} and t_{23}. D31-D0 are not driven during read cycles and bus hold.

6.2.4 PARITY

Data Parity Input/Outputs (DP0-DP3)

DP0-DP3 are the data parity pins for the processor. There is one pin for each byte of the data bus. Even parity is generated or checked by the parity generators/checkers. Even parity means that there are an even number of HIGH inputs on the eight corresponding data bus pins and parity pin.

Data parity is generated on all write data cycles with the same timing as the data driven by the 486 microprocessor. Even parity information must be driven back to the 486 microprocessor on these pins with the same timing as read information to insure that the correct parity check status is indicated by the 486 microprocessor.

The values read on these pins do not affect program execution. It is the responsibility of the system to take appropriate actions if a parity error occurs.

Input signals on DP0-DP3 must meet setup and hold times t_{22} and t_{23} for proper operation.

Parity Status Output (PCHK#)

Parity status is driven on the PCHK# pin, and a parity error is indicated by this pin being LOW. PCHK# is driven the clock after ready for read operations to indicate the parity status for the data sampled at the end of the previous clock. Parity is checked during code reads, memory reads and I/O reads. Parity is not checked during interrupt acknowledge cycles. PCHK# only checks the parity status for enabled bytes as indicated by the byte enable and bus size signals. It is valid only in the clock immediately after read data is returned to the 486 microprocessor. At all other times it is inactive (HIGH). PCHK# is never floated.

Driving PCHK# is the only effect that bad input parity has on the 486 microprocessor. The 486 microprocessor will not vector to a bus error interrupt when bad data parity is returned. In systems that will not employ parity, PCHK# can be ignored. In systems not using parity, DP0-DP3 should be connected to V_{CC} through a pullup resistor.

6.2.5 BUS CYCLE DEFINITION

M/IO#, D/C#, W/R# Outputs

M/IO#, D/C# and W/R# are the primary bus cycle definition signals. They are driven valid as the ADS# signal is asserted. M/IO# distinguishes between memory and I/O cycles, D/C# distinguishes between data and control cycles and W/R# distinguishes between write and read cycles.

Bus cycle definitions as a function of M/IO#, D/C# and W/R# are given in Table 6.1. Note there is a difference between the 486 microprocessor and 386 microprocessor bus cycle definitions. The halt bus cycle type has been moved to location 001 in the 486 microprocessor from location 101 in the 386 microprocessor. Location 101 is now reserved and will never be generated by the 486 microprocessor.

Table 6.1. AD5# Initiated Bus Cycle Definitions

M/IO#	D/C#	W/R#	Bus Cycle Initiated
0	0	0	Interrupt Acknowledge
0	0	1	Halt/Special Cycle
0	1	0	I/O Read
0	1	1	I/O Write
1	0	0	Code Read
1	0	1	Reserved
1	1	0	Memory Read
1	1	1	Memory Write

Special bus cycles are discussed in Section 7.2.11.

Bus Lock Output (LOCK#)

LOCK# indicates that the 486 microprocessor is running a read-modify-write cycle where the external bus must not be relinquished between the read and write cycles. Read-modify-write cycles are used to implement memory-based semaphores. Multiple reads or writes can be locked.

When LOCK# is asserted, the current bus cycle is locked and the 486 microprocessor should be allowed exclusive access to the system bus. LOCK# goes active in the first clock of the first locked bus cycle and goes inactive after ready is returned indicating the last locked bus cycle.

The 486 microprocessor will not acknowledge bus hold when LOCK# is asserted (though it will allow an address hold). LOCK# is active LOW and is floated during bus hold. Locked read cycles will not be transformed into cache fill cycles if KEN# is returned active. Refer to Section 7.2.6 for a detailed discussion of Locked bus cycles.

Pseudo-Lock Output (PLOCK#)

The pseudo-lock feature allows atomic reads and writes of memory operands greater than 32 bits. These operands require more than one cycle to

transfer. The 486 microprocessor asserts PLOCK# during floating point long reads and writes (64 bits), segment table descriptor reads (64 bits) and cache line fills (128 bits).

When PLOCK# is asserted no other master will be given control of the bus between cycles. A bus hold request (HOLD) is not acknowledged during pseudo-locked reads and writes, with one exception. During non-cacheable non-bursted code prefetches, HOLD is recognized on memory cycle boundaries even though PLOCK# is asserted. The 486 microprocessor will drive PLOCK# active until the addresses for the last bus cycle of the transaction have been driven regardless of whether BRDY# or RDY# are returned.

A pseudo-locked transfer is meaningful only if the memory operand is aligned and if its completely contained within a single cache line. A 64-bit floating point number must be aligned to an 8-byte boundary to guarantee an atomic access.

Normally PLOCK# and BLAST# are inverse of each other. However during the first cycle of a 64-bit floating point write, both PLOCK# and BLAST# will be asserted.

Since PLOCK# is a function of the bus size and KEN# inputs, PLOCK# should be sampled only in the clock ready is returned. This pin is active LOW and is not driven during bus hold. Refer to Section 7.2.7 for a detailed discussion of pseudo-locked bus cycles.

6.2.6 BUS CONTROL

The bus control signals allow the processor to indicate when a bus cycle has begun, and allow other system hardware to control burst cycles, data bus width and bus cycle termination.

Address Status Output (ADS#)

The ADS# output indicates that the address and bus cycle definition signals are valid. This signal will go active in the first clock of a bus cycle and go inactive in the second and subsequent clocks of the cycle. ADS# is also inactive when the bus is idle.

ADS# is used by external bus circuitry as the indication that the processor has started a bus cycle. The external circuit must sample the bus cycle definition pins on the next rising edge of the clock after ADS# is driven active.

ADS# is active LOW and is not driven during bus hold.

Non-burst Ready Input (RDY#)

RDY# indicates that the current bus cycle is complete. In response to a read, RDY# indicates that the external system has presented valid data on the data pins. In response to a write request, RDY# indicates that the external system has accepted the 486 microprocessor data. RDY# is ignored when the bus is idle and at the end of the first clock of the bus cycle. Since RDY# is sampled during address hold, data can be returned to the processor when AHOLD is active.

RDY# is active LOW, and is not provided with an internal pullup resistor. This input must satisfy setup and hold times t_{16} and t_{17} for proper chip operation.

6.2.7 BURST CONTROL

Burst Ready Input (BRDY#)

BRDY# performs the same function during a burst cycle that RDY# performs during a non-burst cycle. BRDY# indicates that the external system has presented valid data on the data pins in response to a read or that the external system has accepted the 486 microprocessor data in response to a write. BRDY# is ignored when the bus is idle and at the end of the first clock in a bus cycle.

During a burst cycle, BRDY# will be sampled each clock, and if active, the data presented on the data bus pins will be strobed into the 486 microprocessor. ADS# is negated during the second through last data cycles in the burst, but address lines A2–A3 and byte enables will change to reflect the next data item expected by the 486 microprocessor.

If RDY# is returned simultaneously with BRDY#, BRDY# is ignored and the burst cycle is prematurely aborted. An additional complete bus cycle will be initiated after an aborted burst cycle if the cache line fill was not complete. BRDY# is treated as a normal ready for the last data cycle in a burst transfer or for non-burstable cycles. Refer to Section 7.2.2 for burst cycle timing.

BRDY# is active LOW and is provided with a small internal pullup resistor. BRDY# must satisfy the setup and hold times t_{16} and t_{17}.

Burst Last Output (BLAST#)

BLAST# indicates that the next time BRDY# is returned it will be treated as a normal RDY#, terminating the line fill or other multiple-data-cycle transfer. BLAST# is active for all bus cycles regardless of whether they are cacheable or not. This pin is active LOW and is not driven during bus hold.

6.2.8 INTERRUPT SIGNALS (RESET, INTR, NMI)

The interrupt signals can interrupt or suspend execution of the processor's current instruction stream.

Reset Input (RESET)

RESET forces the 486 microprocessor to begin execution at a known state. For a power-up (cold start) reset, V_{CC} and CLK must reach their proper DC and AC specifications for at least 1 ms before the 486 microprocessor begins instruction execution. The RESET pin should remain active during this time to ensure proper 486 microprocessor operation. However, for a warm boot-up case, RESET is required to remain active for a minimum of 15 clocks. The testability operating modes are programmed by the falling (inactive going) edge of RESET. (Refer to Section 8.0 for a description of the test modes during reset.)

Maskable Interrupt Request Input (INTR)

INTR indicates that an external interrupt has been generated. Interrupt processing is initiated if the IF flag is active in the EFLAGS register.

The 486 microprocessor will generate two locked interrupt acknowledge bus cycles in response to asserting the INTR pin. An 8-bit interrupt number will be latched from an external interrupt controller at the end of the second interrupt acknowledge cycle. INTR must remain active until the interrupt acknowledges have been performed to assure program interruption. Refer to Section 7.2.10 for a detailed discussion of interrupt acknowledge cycles.

The INTR pin is active HIGH and is not provided with an internal pulldown resistor. INTR is asynchronous, but the INTR setup and hold times, t_{20} and t_{21}, must be met to assure recognition on any specific clock.

Non-maskable Interrupt Request Input (NMI)

NMI is the non-maskable interrupt request signal. Asserting NMI causes an interrupt with an internally supplied vector value of 2. External interrupt acknowledge cycles are not generated since the NMI interrupt vector is internally generated. When NMI processing begins, the NMI signal will be masked internally until the IRET instruction is executed.

NMI is rising edge sensitive after internal synchronization. NMI must be held LOW for at least four CLK periods before this rising edge for proper operation. NMI is not provided with an internal pulldown resistor. NMI is asynchronous but setup and hold times, t_{20} and t_{21} must be met to assure recognition on any specific clock.

6.2.9 BUS ARBITRATION SIGNALS

This section describes the mechanism by which the processor relinquishes control of its local bus when requested by another bus master.

Bus Request Output (BREQ)

The 486 asserts BREQ whenever a bus cycle is pending internally. Thus, BREQ is always asserted in the first clock of a bus cycle, along with ADS#. Furthermore, if the 486 is currently not driving the bus (due to HOLD, AHOLD, or BOFF#), BREQ is asserted in the same clock that ADS# would have been asserted if the processor were driving the bus. After the first clock of the bus cycle, BREQ may change state. It will be asserted if additional cycles are necessary to complete a transfer (via BS8#, BS16#, KEN#), or if more cycles are pending internally. However, if no additional cycles are necessary to complete the current transfer, BREQ can be negated before ready comes back for the current cycle. External logic can use the BREQ signal to arbitrate among multiple processors. This pin is driven regardless of the state of bus hold or address hold. BREQ is active HIGH and is never floated. During a hold state, internal events may cause BREQ to be deasserted prior to any bus cycles.

Bus Hold Request Input (HOLD)

HOLD allows another bus master complete control of the 486 microprocessor bus. The 486 microprocessor will respond to an active HOLD signal by asserting HLDA and placing most of its output and input/output pins in a high impedance state (floated) after completing its current bus cycle, burst cycle, or sequence of locked cycles. The BREQ, HLDA, PCHK# and FERR# pins are not floated during bus hold. The 486 microprocessor will maintain its bus in this state until the HOLD is deasserted. Refer to Section 7.2.9 for timing diagrams for a bus hold cycle.

Unlike the 386 microprocessor, the 486 microprocessor will recognize HOLD during reset. Pullup resistors are not provided for the outputs that are floated in response to HOLD. HOLD is active HIGH and is not provided with an internal pulldown resistor. HOLD must satisfy setup and hold times t_{18} and t_{19} for proper chip operation.

Bus Hold Acknowledge Output (HLDA)

HLDA indicates that the 486 microprocessor has given the bus to another local bus master. HLDA goes active in response to a hold request presented on the HOLD pin. HLDA is driven active in the same clock that the 486 microprocessor floats its bus.

HLDA will be driven inactive when leaving bus hold and the 486 microprocessor will resume driving the bus. The 486 microprocessor will not cease internal activity during bus hold since the internal cache will satisfy the majority of bus requests. HLDA is active HIGH and remains driven during bus hold.

Backoff Input (BOFF#)

Asserting the BOFF# input forces the 486 microprocessor to release control of its bus in the next clock. The pins floated are exactly the same as in response to HOLD. The response to BOFF# differs from the response to HOLD in two ways: First, the bus is floated immediately in response to BOFF# while the 486 completes the current bus cycle before floating its bus in response to HOLD. Second the 486 does not assert HLDA in response to BOFF#.

The processor remains in bus hold until BOFF# is negated. Upon negation, the 486 microprocessor restarts the bus cycle aborted when BOFF# was asserted. To the internal execution engine the effect of BOFF# is the same as inserting a few wait states to the original cycle. Refer to Section 7.2.12 for a description of bus cycle restart.

Any data returned to the processor while BOFF# is asserted is ignored. BOFF# has higher priority than RDY# or BRDY#. If both BOFF# and ready are returned in the same clock, BOFF# takes effect. If BOFF# is asserted while the bus is idle, the 486 microprocessor will float its bus in the next clock. BOFF# is active LOW and must meet setup and hold times t_{18} and t_{19} for proper chip operation.

6.2.10 CACHE INVALIDATION

The AHOLD and EADS# inputs are used during cache invalidation cycles. AHOLD conditions the 486 microprocessors address lines, A4–A31, to accept an address input. EADS# indicates that an external address is actually valid on the address inputs. Activating EADS# will cause the 486 microprocessor to read the external address bus and perform an internal cache invalidation cycle to the address indicated. Refer to Section 7.2.8 for cache invalidation cycle timing.

Address Hold Request Input (AHOLD)

AHOLD is the address hold request. It allows another bus master access to the 486 microprocessor address bus for performing an internal cache invalidation cycle. Asserting AHOLD will force the 486 microprocessor to stop driving its address bus in the next clock. While AHOLD is active only the address bus will be floated, the remainder of the bus can remain active. For example, data can be returned for a previously specified bus cycle when AHOLD is active. The 486 microprocessor will not initiate another bus cycle during address hold. Since the 486 microprocessor floats its bus immediately in response to AHOLD, an address hold acknowledge is not required. If AHOLD is asserted while a bus cycle is in progress, and no readies are returned during the time AHOLD is asserted, the 486 will redrive the same address (that it originally sent out) once AHOLD is negated.

AHOLD is recognized during reset. Since the entire cache is invalidated by reset, any invalidation cycles run during reset will be unnecessary. AHOLD is active HIGH and is provided with a small internal pulldown resistor. It must satisfy the setup and hold times t_{18} and t_{19} for proper chip operation. This pin determines whether or not the built in self test features of the 486 microprocessor will be exercised on assertion of RESET.

External Address Valid Input (EADS#)

EADS# indicates that a valid external address has been driven onto the 486 address pins. This address will be used to perform an internal cache invalidation cycle. The external address will be checked with the current cache contents. If the address specified matches any areas in the cache, that area will immediately be invalidated.

An invalidation cycle may be run by asserting EADS# regardless of the state of AHOLD, HOLD and BOFF#. EADS# is active LOW and is provided with an internal pullup resistor. EADS# must satisfy the setup and hold times t_{12} and t_{13} for proper chip operation.

6.2.11 CACHE CONTROL

Cache Enable Input (KEN#)

KEN# is the cache enable pin. KEN# is used to determine whether the data being returned by the current cycle is cacheable. When KEN# is active and the 486 microprocessor generates a cycle that can be cached (most any memory read cycle), the cycle will be transformed into a cache line fill cycle.

A cache line is 16 bytes long. During the first cycle of a cache line fill the byte-enable pins should be ignored and data should be returned as if all four byte enables were asserted. The 486 microprocessor will run between 4 and 16 contiguous bus cycles to fill the line depending on the bus data width selected by BS8# and BS16#. Refer to Section 7.2.3 for a description of cache line fill cycles.

The KEN# input is active LOW and is provided with a small internal pullup resistor. It must satisfy the setup and hold times t_{14} and t_{15} for proper chip operation.

Cache Flush Input (FLUSH#)

The FLUSH# input forces the 486 microprocessor to flush its entire internal cache. FLUSH# is active LOW and need only be asserted for one clock. FLUSH# is asynchronous but setup and hold times t_{20} and t_{21} must be met for recognition on any specific clock.

FLUSH# also determines whether or not the tristate test mode of the 486 microprocessor will be invoked on assertion of RESET.

6.2.12 PAGE CACHEABILITY (PWT, PCD)

The PWT and PCD output signals correspond to two user attribute bits in the page table entry. When paging is enabled, PWT and PCD correspond to bits 3 and 4 of the page table entry respectively. When paging is disabled, or for cycles that are not paged when paging is enabled (for example I/O cycles) PWT and PCD correspond to bits 3 and 4 in control register 3.

PCD is masked by the CD (cache disable) bit in control register 0 (CR0). When CD=1 (cache line fills disabled) the 486 microprocessor forces PCD HIGH. When CD=0, PCD is driven with the value of the page table entry/directory.

The purpose of PCD is to provide a cacheable/non-cacheable indication on a page by page basis. The 486 will not perform a cache fill to any page in which bit 4 of the page table entry is set. PWT corresponds to the write-back bit and can be used by an external cache to provide this functionality. PCD and PWT bits are assigned to be zero during real mode or whenever paging is disabled. Refer to Sections 4.5.4 and 5.6 for a discussion of non-cacheable pages.

PCD and PWT have the same timing as the cycle definition pins (M/IO#, D/C#, W/R#). PCD and PWT are active HIGH and are not driven during bus hold.

6.2.13 NUMERIC ERROR REPORTING (FERR#, IGNNE#)

To allow PC-type floating point error reporting, the 486 microprocessor provides two pins, FERR# and IGNNE#.

Floating Point Error Output (FERR#)

The 486 microprocessor asserts FERR# whenever an unmasked floating point error is encountered. FERR# is similar to the ERROR# pin on the 387 math coprocessor. FERR# can be used by external logic for PC-type floating point error reporting in 486 microprocessor systems. FERR# is active LOW, and is not floated during bus hold.

In some cases, FERR# is asserted when the next floating point instruction is encountered and in other cases it is asserted before the next floating point instruction is encountered depending upon the execution state of the instruction causing the exception.

The following class of floating point exceptions drive FERR# at the time the exception occurs (i.e., before encountering the next floating point instruction).

1. The stack fault, invalid operation, and denormal exceptions on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exceptions on store instructions (including integer store instructions).

The following class of floating point exceptions drive FERR# only after encountering the next floating point instruction.

1. Exceptions other than on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exception on all basic arithmetic, load, compare, and control instructions (i.e., all other instructions).

Ignore Numeric Error Input (IGNNE#)

The 486 microprocessor will ignore a numeric error and continue executing non-control floating point instructions when IGNNE# is asserted, but FERR# will still be activated. When deasserted, the 486 microprocessor will freeze on a non-control floating point instruction if a previous instruction caused an error. IGNNE# has no effect when the NE bit in control register 0 is set.

The IGNNE# input is active LOW and is provided with a small internal pullup resistor. This input is asynchronous, but must meet setup and hold times t_{20} and t_{21} to insure recognition on any specific clock.

6.2.14 BUS SIZE CONTROL (BS16#, BS8#)

The BS16# and BS8# inputs allow external 16- and 8-bit busses to be supported with a small number of external components. The 486 CPU samples these

pins every clock. The value sampled in the clock before ready determines the bus size. When asserting BS16# or BS8# only 16 or 8 bits of the data bus need be valid. If both BS16# and BS8# are asserted, an 8-bit bus width is selected.

When BS16# or BS8# are asserted the 486 microprocessor will convert a larger data request to the appropriate number of smaller transfers. The byte enables will also be modified appropriately for the bus size selected.

BS16# and BS8# are active LOW and are provided with small internal pullup resistors. BS16# and BS8# must satisfy the setup and hold times t_{14} and t_{15} for proper chip operation.

6.2.15 ADDRESS BIT 20 MASK (A20M#)

Asserting the A20M# input causes the 486 microprocessor to mask physical address bit 20 before performing a lookup in the internal cache and before driving a memory cycle to the outside world. When A20M# is asserted, the 486 microprocessor emulates the 1 Mbyte address wraparound that occurs on the 8086. A20M# is active LOW and must be asserted only when the processor is in real mode. The A20M# is not defined in Protected Mode. A20M# is asynchronous but should meet setup and hold times t_{20} and t_{21} for recognition in any specific clock. For correct operation of the chip, A20M# should be sampled high 2 clocks before and 2 clocks after RESET goes low.

6.3 Write Buffers

The 486 microprocessor contains four write buffers to enhance the performance of consecutive writes to memory. The buffers can be filled at a rate of one write per clock until all four buffers are filled.

When all four buffers are empty and the bus is idle, a write request will propagate directly to the external bus bypassing the write buffers. If the bus is not available at the time the write is generated internally, the write will be placed in the write buffers and propagate to the bus as soon as the bus becomes available. The write is stored in the on-chip cache immediately if the write is a cache hit.

Writes will be driven onto the external bus in the same order in which they are received by the write buffers. Under certain conditions a memory read will go onto the external bus before the memory writes pending in the buffer even though the writes occurred earlier in the program execution.

A memory read will only be reordered in front of all writes in the buffers under the following conditions: If all writes pending in the buffers are cache hits and the read is a cache miss. Under these conditions the 486 microprocessor will not read from an external memory location that needs to be updated by one of the pending writes.

Reordering of a read with the writes pending in the buffers can only occur once before all the buffers are emptied. Reordering read once only maintains cache consistency. Consider the following example: The CPU writes to location X. Location X is in the internal cache, so it is updated there immediately. However, the bus is busy so the write out to main memory is buffered (see Figure 6.3(a)). At this point, any reads to location X would be cache hits and most up-to-date data would be read.

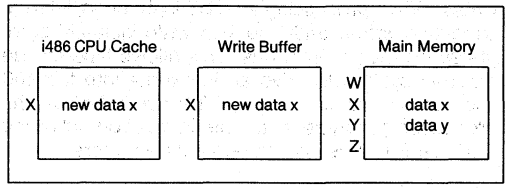


Figure 6.3(a)

The next instruction causes a read to location Y. Location Y is not in the cache (a cache miss). Since the write in the write buffer is a cache hit, the read is reordered. When location Y is read, it is put into the cache. The possibility exists that location Y will replace location X in the cache. If this is true, location X would no longer be cached (see Figure 6.3(b)).

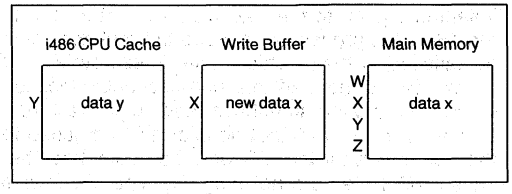


Figure 6.3(b)

Cache consistency has been maintained up to this point. If a subsequent read is to location X (now a cache miss) and it was reordered in front of the buffered write to location X, stale data would be read. This is why only 1 read is allowed to be reordered. Once a read is reordered, all the writes in the write buffer are flagged as cache misses to ensure that no more reads are reordered. Since one of the conditions to reorder a read is that all writes in the write buffer must be cache hits, no more reordering is allowed until all of those flagged writes propogate to the bus. Similarly, if an invalidation cycle is run all entries in the write buffer are flagged as cache misses.

For multiple processor systems and/or systems using DMA techniques, such as bus snooping, locked semaphores should be used to maintain cache consistency.

6.3.1 WRITE BUFFERS AND I/O CYCLES

Input/Output (I/O) cycles must be handled in a different manner by the write buffers.

I/O reads are never reordered in front of buffered memory writes. This insures that the 486 microprocessor will update all memory locations before reading status from an I/O device.

The 486 microprocessor never buffers single I/O writes. When processing an OUT instruction, internal execution stops until the I/O write actually completes on the external bus. This allows time for the external system to drive an invalidate into the 486 microprocessor or to mask interrupts before the processor progresses to the instruction following OUT. REP OUTS instructions will be buffered.

I/O device recovery time must be handled slightly differently by the 486 microprocessor than with the 386 microprocessor. I/O device back-to-back write recovery times could be guaranteed by the 386 microprocessor by inserting a jump to the next instruction in the code that writes to the device. The jump forces the 386 microprocessor to generate a prefetch bus cycle which can't begin until the I/O write completes.

Inserting a jump to the next write will not work with the 486 microprocessor because the prefetch could be satisfied by the on-chip cache. A read cycle must be explicitly generated to a non-cacheable location in memory to guarantee that a read bus cycle is performed. This read will not be allowed to proceed to the bus until after the I/O write has completed because I/O writes are not buffered. The I/O device will have time to recover to accept another write during the read cycle.

6.3.2 WRITE BUFFERS IMPLICATIONS ON LOCKED BUS CYCLES

Locked bus cycles are used for read-modify-write accesses to memory. During a read-modify-write access, a memory base variable is read, modified and then written back to the same memory location. It is important that no other bus cycles, generated by other bus masters or by the 486 microprocessor itself, be allowed on the external bus between the read and write portion of the locked sequence.

During a locked read cycle the 486 microprocessor will always access external memory, it will never look for the location in the on-chip cache, but for write cycles, data is written in the internal cache (if cache hit) and in the external memory. All data pending in the 486 microprocessor's write buffers will be written to memory before a locked cycle is allowed to proceed to the external bus.

The 486 microprocessor will assert the LOCK# pin after the write buffers are emptied during a locked bus cycle. With the LOCK# pin asserted, the microprocessor will read the data, operate on the data and place the results in a write buffer. The contents of the write buffer will then be written to external memory. LOCK# will become inactive after the write part of the locked cycle.

6.4 Interrupt and Non-Maskable Interrupt Interface

The 486 microprocessor provides two asynchronous interrupt inputs, INTR (interrupt request) and NMI (non-maskable interrupt input). This section describes the hardware interface between the instruction execution unit and the pins. For a description of the algorithmic response to interrupts refer to Section 2.7. For interrupt timings refer to Section 7.2.10.

6.4.1 INTERRUPT LOGIC

The 486 microprocessor contains a two-clock synchronizer on the interrupt line. An interrupt request will reach the internal instruction execution unit two clocks after the INTR pin is asserted, if proper setup is provided to the first stage of the synchronizer.

There is no special logic in the interrupt path other than the synchronizer. The INTR signal is level sensitive and must remain active for the instruction execution unit to recognize it. The interrupt will not be serviced by the 486 microprocessor if the INTR signal does not remain active.

The instruction execution unit will look at the state of the synchronized interrupt signal at specific clocks during the execution of instructions (if interrupts are enabled). These specific clocks are at instruction boundaries, or iteration boundaries in the case of string move instructions. Interrupts will only be accepted at these boundaries.

An interrupt must be presented to the 486 microprocessor INTR pin three clocks before the end of an instruction for the interrupt to be acknowledged. Presenting the interrupt 3 clocks before the end of an instruction allows the interrupt to pass through the two clock synchronizer leaving one clock to prevent the initiation of the next sequential instruction and to begin interrupt service. If the interrupt is not received in time to prevent the next instruction, it will be accepted at the end of next instruction, assuming INTR is still held active. The interrupt service microcode will start after two dead clocks.

The longest latency between when an interrupt request is presented on the INTR pin and when the interrupt service begins is: longest instruction used + the two clocks for synchronization + one clock required to vector into the interrupt service microcode.

6.4.2 NMI LOGIC

The NMI pin has a synchronizer like that used on the INTR line. Other than the synchronizer, the NMI logic is different from that of the maskable interrupt.

NMI is edge triggered as opposed to the level triggered INTR signal. The rising edge of the NMI signal is used to generate the interrupt request. The NMI input need not remain active until the interrupt is actually serviced. The NMI pin only needs to remain active for a single clock if the required setup and hold times are met. NMI will operate properly if it is held active for an arbitrary number of clocks.

The NMI input must be held inactive for at least four clocks after it is asserted to reset the edge triggered logic. A subsequent NMI may not be generated if the NMI is not held inactive for at least two clocks after being asserted.

The NMI input is internally masked whenever the NMI routine is entered. The NMI input will remain masked until an IRET (return from interrupt) instruction is executed. Masking the NMI signal prevents recursive NMI calls. If another NMI occurs while the NMI is masked off, the pending NMI will be executed after the current NMI is done. Only one NMI can be pending while NMI is masked.

6.5 Reset and Initialization

The 486 microprocessor has a built in self test (BIST) that can be run during reset. The BIST is invoked if the AHOLD pin is asserted for 2 clocks before and 2 clocks after RESET is deasserted. RESET must be active for 15 clocks with or with no BIST being enabled. Refer to Section 8.0 for information on 486 microprocessor testability.

The 486 microprocessor registers have the values shown in Table 6.2 after RESET is performed. The EAX register contains information on the success or failure of the BIST if the self test is executed. The DX register always contains a component identifier at the conclusion of RESET. The upper byte of DX (DH) will contain 04 and the lower byte (DL) will contain a stepping identifier (see Table 6-3). The floating point registers are initialized as if the FINIT/FNINIT (initialize processor) instruction was executed if the BIST was performed. If the BIST is not executed, the floating point registers are unchanged.

Table 6.2. Register Values after Reset

Register	Initial Value (BIST)	Initial Value (No Bist)
EAX	Zero (Pass)	Undefined
ECX	Undefined	Undefined
EDX	0400 + Revision ID	0400 + Revision ID
EBX	Undefined	Undefined
ESP	Undefined	Undefined
EBP	Undefined	Undefined
ESI	Undefined	Undefined
EDI	Undefined	Undefined
EFLAGS	00000002h	00000002h
EIP	0FFF0h	0FFF0h
ES	0000h	0000h
CS	F000h*	F000h*
SS	0000h	0000h
DS	0000h	0000h
FS	0000h	0000h
GS	0000h	0000h
IDTR	Base = 0, Limit = 3FFh	Base = 0, Limit = 3FFh
CR0	60000010h	60000010h
DR7	00000000h	00000000h
CW	037Fh	Unchanged
SW	0000h	Unchanged
TW	FFFFh	Unchanged
FIP	00000000h	Unchanged
FEA	00000000h	Unchanged
FCS	0000h	Unchanged
FDS	0000h	Unchanged
FOP	000h	Unchanged
FSTACK	Undefined	Unchanged

Table 6-3. i486™ CPU Revision ID

i486™ CPU Stepping Name	Revision ID
B3	01
B4	01
B5	01
B6	01
C0	02

The 486 microprocessor will start executing instructions at location FFFFFFF0H after RESET. When the first InterSegment Jump or Call is executed, address lines A20–A31 will drop LOW for CS-relative memory cycles, and the 486 microprocessor will only execute instructions in the lower one Mbyte of physical memory. This allows the system designer to use a ROM at the top of physical memory to initialize the system and take care of RESETs.

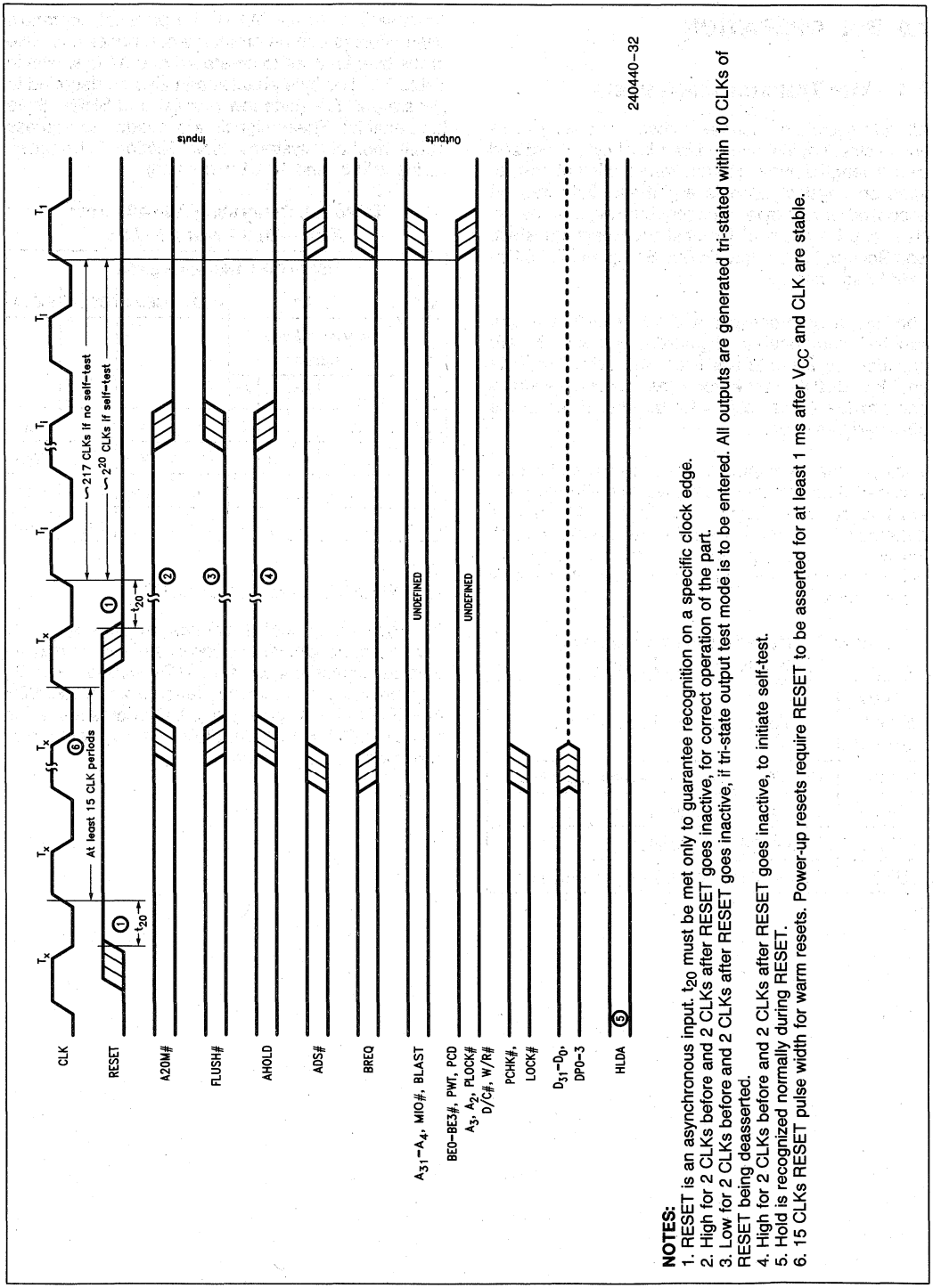
RESET forces the 486 microprocessor to terminate all execution and local bus activity. No instruction or bus activity will occur as long as RESET is active.

All entries in the cache are invalidated by RESET.

6.5.1 PIN STATE DURING RESET

The 486 microprocessor recognizes and can respond to HOLD, AHOLD, and BOFF# requests regardless of the state of RESET. Thus, even though the processor is in reset, it can still float its bus in response to any of these requests.

While in reset, the 486 microprocessor bus is in the state shown in Figure 6.4 if the HOLD, AHOLD and BOFF# requests are inactive. Note that the address (A31–A2, BE3#–BE0#) and cycle definition (M/IO#, D/C#, W/R#) pins are undefined from the time reset is asserted up to the start of the first bus cycle. All undefined pins (except FERR#) assume known values at the beginning of the first bus cycle. The first bus cycle is always a code fetch to address FFFFFFF0H. FERR# reflects the state of the ES (error summary status) bit in the floating point unit status word. The ES bit is initialized whenever the floating point unit state is initialized.



NOTES:
1. RESET is an asynchronous input. t_{20} must be met only to guarantee recognition on a specific clock edge.
2. High for 2 CLKs before and 2 CLKs after RESET goes inactive, for correct operation of the part.
3. Low for 2 CLKs before and 2 CLKs after RESET goes inactive, if tri-state output test mode is to be entered. All outputs are generated tri-stated within 10 CLKs of RESET being deasserted.
4. High for 2 CLKs before and 2 CLKs after RESET goes inactive, to initiate self-test.
5. Hold is recognized normally during RESET.
6. 15 CLKs RESET pulse width for warm resets. Power-up resets require RESET to be asserted for at least 1 ms after V_{CC} and CLK are stable.

Figure 6.4. Pin States during RESET

7.0 BUS OPERATION

7.1 Data Transfer Mechanism

All data transfers occur as a result of one or more bus cycles. Logical data operands of byte, word and dword lengths may be transferred without restrictions on physical address alignment. Data may be accessed at any byte boundary but two or three cycles may be required for unaligned data transfers. See Section 7.1.3 Dynamic Bus Sizing and 7.1.6 Operand Alignment.

The 486 microprocessor address signals are split into two components. High-order address bits are provided by the address lines, A2–A31. The byte enables, BE0#–BE3#, form the low-order address and provide linear selects for the four bytes of the 32-bit address bus.

The byte enable outputs are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 7.1. Byte enable patterns which have a negated byte enable separating two or three asserted byte enables will never occur (see Table 7.5). All other byte enable patterns are possible.

Table 7.1. Byte Enables and Associated Data and Operand Bytes

Byte Enable Signal	Associated Data Bus Signals	
BE0#	D0–D7	(byte 0—least significant)
BE1#	D8–D15	(byte 1)
BE2#	D16–D23	(byte 2)
BE3#	D24–D31	(byte 3—most significant)

Address bits A0 and A1 of the physical operand's base address can be created when necessary. Use of the byte enables to create A0 and A1 is shown in Table 7.2. The byte enables can also be decoded to generate BLE# (byte low enable) and BHE# (byte high enable). These signals are needed to address 16-bit memory systems (see Section 7.1.4 Interfacing with 8- and 16-bit memories).

Table 7.2. Generating A0–A31 from BE0#–BE3# and A2–A31

486™ CPU Address Signals								
A31 ..... A2					BE3#	BE2#	BE1#	BE0#
	Physical Base Address							
A31	.....	A2	A1	A0				
A31	.....	A2	0	0	X	X	X	Low
A31	.....	A2	0	1	X	X	Low	High
A31	.....	A2	1	0	X	Low	High	High
A31	.....	A2	1	1	Low	High	High	High

7.1.1 MEMORY AND I/O SPACES

Bus cycles may access physical memory space or I/O space. Peripheral devices in the system may either be memory-mapped, or I/O-mapped, or both. Physical memory addresses range from 00000000H to FFFFFFFFH (4 gigabytes). I/O addresses range from 00000000H to 0000FFFFH (64 Kbytes) for programmed I/O. See Figure 7.1.

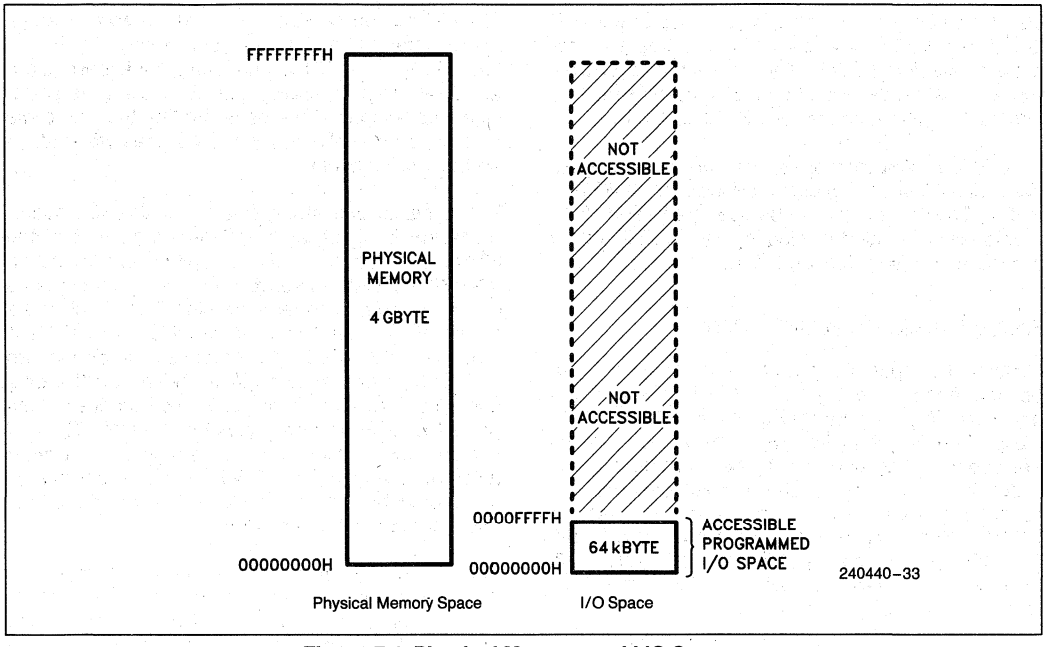


Figure 7.1. Physical Memory and I/O Spaces

7.1.2 MEMORY AND I/O SPACE ORGANIZATION

The 486 microprocessor datapath to memory and input/output (I/O) spaces can be 32-, 16- or 8-bits wide. The byte enable signals, BE0#–BE3#, allow byte granularity when addressing any memory or I/O structure whether 8, 16 or 32 bits wide.

The 486 microprocessor includes bus control pins, BS16# and BS8#, which allow direct connection to 16- and 8-bit memories and I/O devices. Cycles to 32-, 16- and 8-bit may occur in any sequence, since the BS8# and BS16# signals are sampled during each bus cycle.

32-bit wide memory and I/O spaces are organized as arrays of physical 4-byte words. Each memory or I/O 4-byte word has four individually addressable bytes at consecutive byte addresses (see Figure 7.2). The lowest addressed byte is associated with data signals D0–D7; the highest-addressed byte with D24–D31. Physical 4-byte words begin at addresses divisible by four.

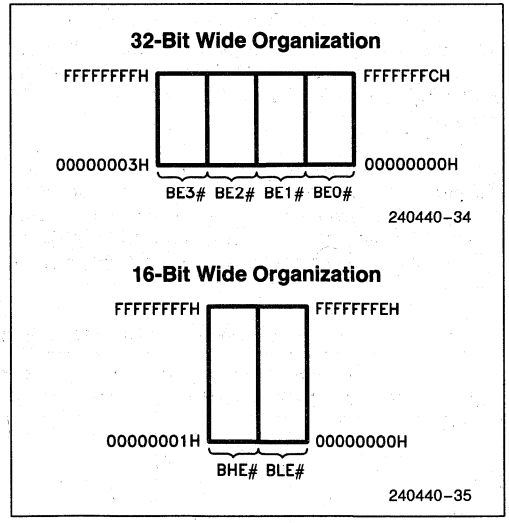


Figure 7.2. Physical Memory and I/O Space Organization

16-bit memories are organized as arrays of physical 2-byte words. Physical 2-byte words begin at addresses divisible by two. The byte enables BE0# – BE3#, must be decoded to A1, BLE# and BHE# to address 16-bit memories (see Section 7.1.4).

To address 8-bit memories, the two low order address bits A0 and A1, must be decoded from BE0# – BE3#. The same logic can be used for 8- and 16-bit memories since the decoding logic for BLE# and A0 are the same (see Section 7.1.4).

7.1.3 DYNAMIC DATA BUS SIZING

Dynamic data bus sizing is a feature allowing processor connection to 32-, 16- or 8-bit buses for memory or I/O. A processor may connect to all three bus sizes. Transfers to or from 32-, 16- or 8-bit devices are supported by dynamically determining the bus width during each bus cycle. Address decoding circuitry may assert BS16# for 16-bit devices, or BS8# for 8-bit devices during each bus cycle. BS8# and BS16# must be negated when addressing 32-bit devices. An 8-bit bus width is selected if both BS16# and BS8# are asserted.

BS16# and BS8# force the 486 microprocessor to run additional bus cycles to complete requests larger than 16- or 8 bits. A 32-bit transfer will be converted into two 16-bit transfers (or 3 transfers if the data is misaligned) when BS16# is asserted. Asserting BS8# will convert a 32-bit transfer into four 8-bit transfers.

Extra cycles forced by BS16# or BS8# should be viewed as independent bus cycles. BS16# or BS8# must be driven active during each of the extra cycles unless the addressed device has the ability to change the number of bytes it can return between cycles.

The 486 microprocessor will drive the byte enables appropriately during extra cycles forced by BS8# and BS16#. A2–A31 will not change if accesses are to a 32-bit aligned area. Table 7.3 shows the set of byte enables that will be generated on the next cycle for each of the valid possibilities of the byte enables on the current cycle.

The dynamic bus sizing feature of the 486 microprocessor is significantly different than that of the 386 microprocessor. Unlike the 386 microprocessor, the 486 microprocessor requires that data bytes be driven on the addressed data pins. The simplest example of this function is a 32-bit aligned, BS16# read. When the 486 microprocessor reads the two high order bytes, they must be driven on the data bus pins D16–D31. The 486 microprocessor expects the two low order bytes on D0–D15. The 386 microprocessor expects both the high and low order bytes on D0–D15. The 386 microprocessor always reads or writes data on the lower 16 bits of the data bus when BS16# is asserted.

The external system must contain buffers to enable the 486 microprocessor to read and write data on the appropriate data bus pins. Table 7.4 shows the data bus lines where the 486 microprocessor expects data to be returned for each valid combination of byte enables and bus sizing options.

Valid data will only be driven onto data bus pins corresponding to active byte enables during write cycles. Other pins in the data bus will be driven but they will not contain valid data. Unlike the 386 microprocessor, the 486 microprocessor will not duplicate write data onto parts of the data bus for which the corresponding byte enable is negated.

Table 7.3. Next Byte Enable Values for BSn# Cycles

Current				Next with BS8#				Next with BS16#			
BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#	BE3#	BE2#	BE1#	BE0#
1	1	1	0	n	n	n	n	n	n	n	n
1	1	0	0	1	1	0	1	n	n	n	n
1	0	0	0	1	0	0	1	1	0	1	1
0	0	0	0	0	0	0	1	0	0	1	1
1	1	0	1	n	n	n	n	n	n	n	n
1	0	0	1	1	0	1	1	1	0	1	1
0	0	0	1	0	0	1	1	0	0	1	1
1	0	1	1	n	n	n	n	n	n	n	n
0	0	1	1	0	1	1	1	n	n	n	n
0	1	1	1	n	n	n	n	n	n	n	n

"n" means that another bus cycle will not be required to satisfy the request.

Table 7.4. Data Pins Read with Different Bus Sizes

BE3#	BE2#	BE1#	BE0#	w/o BS8#/BS16#	w BS8#	W BS16#
1	1	1	0	D7–D0	D7–D0	D7–D0
1	1	0	0	D15–D0	D7–D0	D15–D0
1	0	0	0	D23–D0	D7–D0	D15–D0
0	0	0	0	D31–D0	D7–D0	D15–D0
1	1	0	1	D15–D8	D15–D8	D15–D8
1	0	0	1	D23–D8	D15–D8	D15–D8
0	0	0	1	D31–D8	D15–D8	D15–D8
1	0	1	1	D23–D16	D23–D16	D23–D16
0	0	1	1	D31–D16	D23–D16	D31–D16
0	1	1	1	D31–D24	D31–D24	D31–D24

7.1.4 INTERFACING WITH 8-, 16- AND 32-BIT MEMORIES

In 32-bit physical memories such as Figure 7.3, each 4-byte word begins at a byte address that is a multiple of four. A2–A31 are used as a 4-byte word select. BE0#–BE3# select individual bytes within the 4-byte word. BS8# and BS16# are negated for all bus cycles involving the 32-bit array.

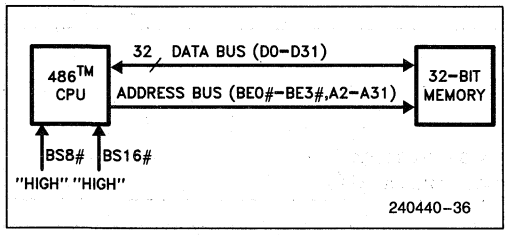


Figure 7.3. i486™ Microprocessor with 32-Bit Memory

16- and 8-bit memories require external byte swapping logic for routing data to the appropriate data lines and logic for generating BHE#, BLE# and A1. In systems where mixed memory widths are used, extra address decoding logic is necessary to assert BS16# or BS8#.

Figure 7.4 shows the 486 microprocessor address bus interface to 32-, 16- and 8-bit memories. To address 16-bit memories the byte enables must be decoded to produce A1, BHE# and BLE# (A0). For 8-bit wide memories the byte enables must be decoded to produce A0 and A1. The same byte select logic can be used in 16- and 8-bit systems since BLE# is exactly the same as A0 (see Table 7.5).

BE0#–BE3# can be decoded as shown in Table 7.5 to generate A1, BHE# and BLE#. The byte select logic necessary to generate BHE# and BLE# is shown in Figure 7.5.

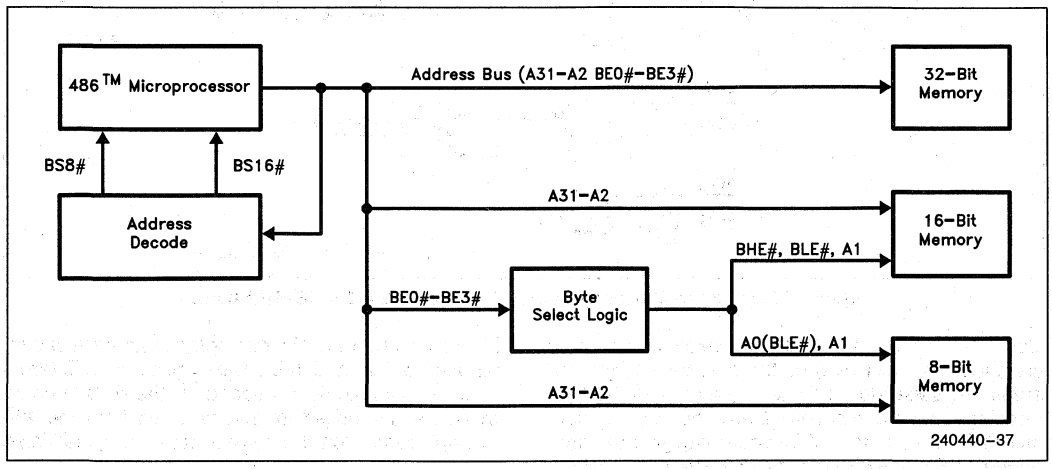


Figure 7.4. Addressing 16- and 8-Bit Memories

Table 7.5. Generating A1, BHE# and BLE# for Addressing 16-Bit Devices

i486™ CPU Signals				8, 16-Bit Bus Signals			Comments
BE3#	BE2#	BE1#	BE0#	A1	BHE#	BLE# (A0)	
H*	H*	H*	H*	x	x	x	x—no active bytes
H	H	H	L	L	H	L	
H	H	L	H	L	L	H	
H	H	L	L	L	L	L	
H	L	H	H	H	H	L	
H*	L*	H*	L*	x	x	x	x—not contiguous bytes
H	L	L	H	L	L	H	
H	L	L	L	L	L	L	
L	H	H	H	H	L	H	
L*	H*	H*	L*	x	x	x	x—not contiguous bytes
L*	H*	L*	H*	x	x	x	x—not contiguous bytes
L*	H*	L*	L*	x	x	x	x—not contiguous bytes
L	L	H	H	H	L	L	
L*	L*	H*	L*	x	x	x	x—not contiguous bytes
L	L	L	H	L	L	H	
L	L	L	L	L	L	L	

BLE# asserted when D0–D7 of 16-bit bus is active.
BHE# asserted when D8–D15 of 16-bit bus is active.
A1 low for all even words; A1 high for all odd words.

Key:
x = don't care
H = high voltage level
L = low voltage level
* = a non-occurring pattern of Byte Enables; either none are asserted, or the pattern has Byte Enables asserted for non-contiguous bytes

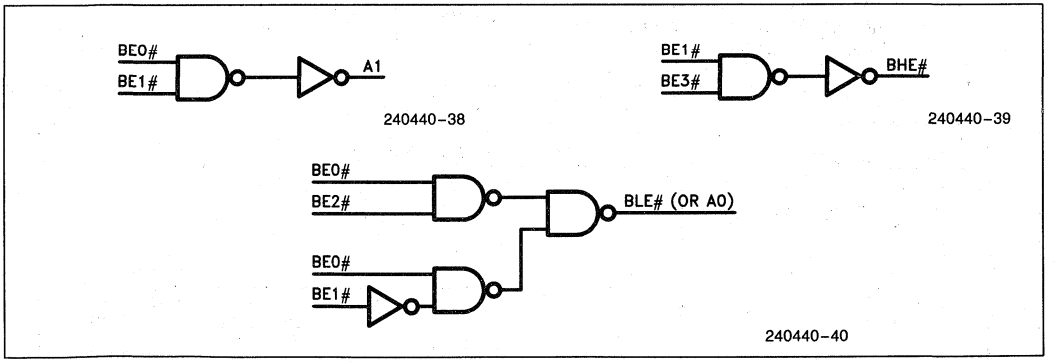


Figure 7.5. Logic to Generate A1, BHE# and BLE# for 16-Bit Busses

Combinations of BE0#–BE3# which never occur are those in which two or three asserted byte enables are separated by one or more negated byte enables. These combinations are "don't care" conditions in the decoder. A decoder can use the non-occurring BE0#–BE3# combinations to its best advantage.

Figure 7.6 shows a 486 microprocessor data bus interface to 16- and 8-bit wide memories. External byte swapping logic is needed on the data lines so that data is supplied to, and received from the 486 microprocessor on the correct data pins (see Table 7.4).

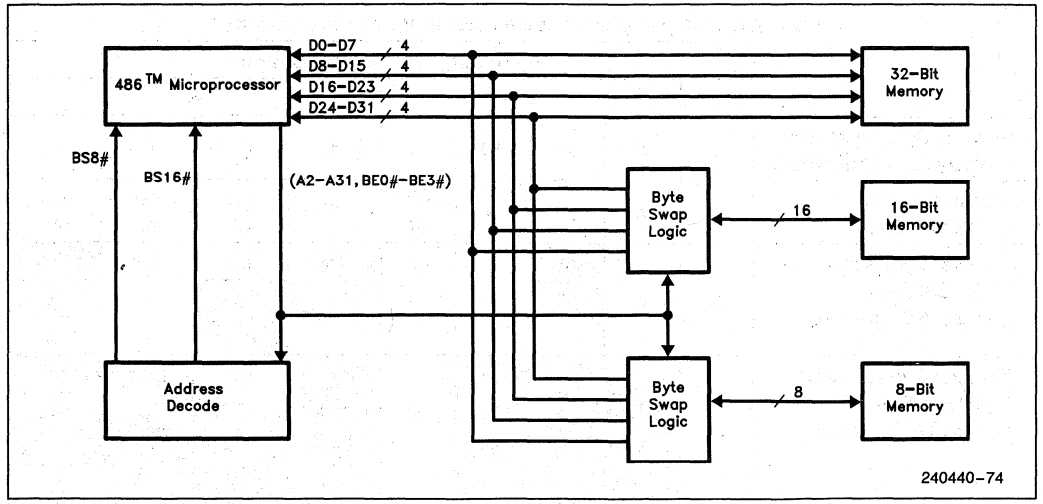


Figure 7.6. Data Bus Interface to 16- and 8-bit Memories

7.1.5 DYNAMIC BUS SIZING DURING CACHE LINE FILLS

BS8# and BS16# can be driven during cache line fills. The 486 microprocessor will generate enough 8- or 16-bit cycles to fill the cache line. This can be up to 16 8-bit cycles.

The external system should assume that all byte enables are active for the first cycle of a cache line fill. The 486 microprocessor will generate proper byte enables for subsequent cycles in the line fill. Table 7.6 shows the appropriate A0 (BLE#), A1 and BHE# for the various combinations of the 486 microprocessor byte enables on both the first and subsequent cycles of the cache line fill. The "*" marks all combinations of byte enables that will be generated by the 486 microprocessor during a cache line fill.

7.1.6 OPERAND ALIGNMENT

Physical 4-byte words begin at addresses that are multiples of four. It is possible to transfer a logical operand that spans more than one physical 4-byte word of memory or I/O at the expense of extra cycles. Examples are 4-byte operands beginning at addresses that are not evenly divisible by 4, or 2-byte words split between two physical 4-byte words. These are referred to as unaligned transfers.

Operand alignment and data bus size dictate when multiple bus cycles are required. Table 7.7 describes the transfer cycles generated for all combinations of logical operand lengths, alignment, and data bus sizing. When multiple cycles are required to transfer a multi-byte logical operand, the highest-order bytes are transferred first. For example, when the processor does a 4-byte unaligned read beginning at location x11 in the 4-byte aligned space, the three high order bytes are read in the first bus cycle. The low byte is read in a subsequent bus cycle.

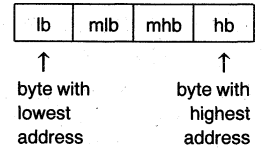
Table 7.6. Generating A0, A1 and BHE# from the i486™ Microprocessor Byte Enables

BE3#	BE2#	BE1#	BE0#	First Cache Fill Cycle			Any Other Cycle		
				A0	A1	BHE#	A0	A1	BHE#
1	1	1	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
*0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	1	0	0
1	0	0	1	0	0	0	1	0	0
*0	0	0	1	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	1
*0	0	1	1	0	0	0	0	1	0
*0	1	1	1	0	0	0	1	1	0

Table 7.7. Transfer Bus Cycles for Bytes, Words and Dwords

	Byte-Length of Logical Operand								
	1	2				4			
Physical Byte Address in Memory (Low Order Bits)	xx	00	01	10	11	00	01	10	11
Transfer Cycles over 32-Bit Bus	b	w	w	w	hb lb	d	hb l3	hw lw	h3 lb
Transfer Cycles over 16-Bit Data Bus ▒ = BS16# Asserted	b	w	lb hb	w	hb lb	lw hw	hb lb mw	hw lw	mw hb lb
Transfer Cycles over 8-Bit Data Bus ▒ = BS8# Asserted	b	lb hb	lb hb	lb hb	hb lb	lb mlb mhb hb	hb lb mlb mhb	mhb hb lb mlb	mlb mhb hb lb

KEY:
b = byte transfer
w = 2-byte transfer
3 = 3-byte transfer
d = 4-byte transfer
h = high-order portion
l = low-order portion
m = mid-order portion



The function of unaligned transfers with dynamic bus sizing is not obvious. When the external systems asserts BS16# or BS8# forcing extra cycles, low-order bytes or words are transferred first (opposite to the example above). When the 486 microprocessor requests a 4-byte read and the external system asserts BS16#, the lower 2 bytes are read first followed by the upper 2 bytes.

In the unaligned transfer described above, the processor requested three bytes on the first cycle. If the external system asserted BS16# during this 3-byte transfer, the lower word is transferred first followed by the upper byte. In the final cycle the lower byte of the 4-byte operand is transferred as in the 32-bit example above.

7.2 Bus Functional Description

The 486 microprocessor supports a wide variety of bus transfers to meet the needs of high performance systems. Bus transfers can be single cycle or multiple cycle, burst or non-burst, cacheable or non-cacheable, 8-, 16- or 32-bit, and pseudo-locked. To support multiprocessing systems there are cache invalidation cycles and locked cycles.

This section begins with basic non-cacheable non-burst single cycle transfers. It moves on to multiple cycle transfers and introduces the burst mode. Cacheability is introduced in Section 7.2.3. The remaining sections describe locked, pseudo-locked, invalidate, bus hold and interrupt cycles.

Bus cycles and data cycles are discussed in this section. A bus cycle is at least two clocks long and begins with ADS# active in the first clock and ready active in the last clock. Data is transferred to or from the 486 microprocessor during a data cycle. A bus cycle contains one or more data cycles.

Refer to Section 7.2.13 for a description of the bus states shown in the timing diagrams.

7.2.1 NON-CACHEABLE NON-BURST SINGLE CYCLE

7.2.1.1 No Wait States

The fastest non-burst bus cycle that the 486 microprocessor supports is two clocks long. These cycles are called 2-2 cycles because reads and writes take two cycles each. The first 2 refers to reads and the

second to writes. For example, if a wait state needs to be added to a write, the cycle would be called 2-3.

Basic two clock read and write cycles are shown in Figure 7.7. The 486 microprocessor initiates a cycle by asserting the address status signal (ADS#) at the rising edge of the first clock. The ADS# output indicates that a valid bus cycle definition and address is available on the cycle definition lines and address bus.

The non-burst ready input (RDY#) is returned by the external system in the second clock. RDY# indicates that the external system has presented valid data on the data pins in response to a read or the external system has accepted data in response to a write.

The 486 microprocessor samples RDY# at the end of the second clock. The cycle is complete if RDY# is active (LOW) when sampled. Note that RDY# is ignored at the end of the first clock of the bus cycle.

The burst last signal (BLAST#) is asserted (LOW) by the 486 microprocessor during the second clock of the first cycle in all bus transfers illustrated in Figure 7.7. This indicates that each transfer is complete after a single cycle. The 486 microprocessor asserts BLAST# in the last cycle of a bus transfer.

The timing of the parity check output (PCHK#) is shown in Figure 7.7. The 486 microprocessor drives the PCHK# output one clock after ready terminates a read cycle. PCHK# indicates the parity status for the data sampled at the end of the previous clock. The PCHK# signal can be used by the external system. The 486 microprocessor does nothing in response to the PCHK# output.

7.2.1.2 Inserting Wait States

The external system can insert wait states into the basic 2-2 cycle by driving RDY# inactive at the end of the second clock. RDY# must be driven inactive to insert a wait state. Figure 7.8 illustrates a simple non-burst, non-cacheable signal with one wait state added. Any number of wait states can be added to a 486 microprocessor bus cycle by maintaining RDY# inactive.

The burst ready input (BRDY#) must be driven inactive on all clock edges where RDY# is driven inactive for proper operation of these simple non-burst cycles.

7.2.2 MULTIPLE AND BURST CYCLE BUS TRANSFERS

Multiple cycle bus transfers can be caused by internal requests from the 486 microprocessor or by the external memory system. An internal request for a 64-bit floating point load or a 128-bit pre-fetch must take more than one cycle. Internal requests for unaligned data may also require multiple bus cycles. A cache line fill requires multiple cycles to complete. The external system can cause a multiple cycle transfer when it can only supply 8 or 16 bits per cycle.

Only multiple cycle transfers caused by internal requests are considered in this section. Cacheable cycles and 8- and 16-bit transfers are covered in Sections 7.2.3 and 7.2.5.

7.2.2.1 Burst Cycles

The 486 microprocessor can accept burst cycles for any bus requests that require more than a single data cycle. During burst cycles, a new data item is strobed into the 486 microprocessor every clock rather than every other clock as in non-burst cycles. The fastest burst cycle requires 2 clocks for the first data item with subsequent data items returned every clock.

The 486 microprocessor is capable of bursting a maximum of 32 bits during a write. Burst writes can only occur if BS8# or BS16# is asserted. For example, the 486 microprocessor can burst write four 8-bit operands or two 16-bit operands in a single burst cycle. But the 486 microprocessor cannot burst multiple 32-bit writes in a single burst cycle.

Burst cycles begin with the 486 microprocessor driving out an address and asserting ADS# in the same manner as non-burst cycles. The 486 microprocessor indicates that it is willing to perform a burst cycle by holding the burst last signal (BLAST#) inactive in the second clock of the cycle. The external system indicates its willingness to do a burst cycle by returning the burst ready signal (BRDY#) active.

The addresses of the data items in a burst cycle will all fall within the same 16-byte aligned area (corresponding to an internal 486 microprocessor cache line). A 16-byte aligned area begins at location XXXXXXX0 and ends at location XXXXXXXF. During a burst cycle, only BE0–3#, A_2, and A_3 may change. A_4–A_{31}, M/IO#, D/C#, and W/R# will remain stable throughout a burst. Given the first address in a burst, external hardware can easily calculate the address of subsequent transfers in advance. An external memory system can be designed to quickly fill the 486 microprocessor internal cache lines.

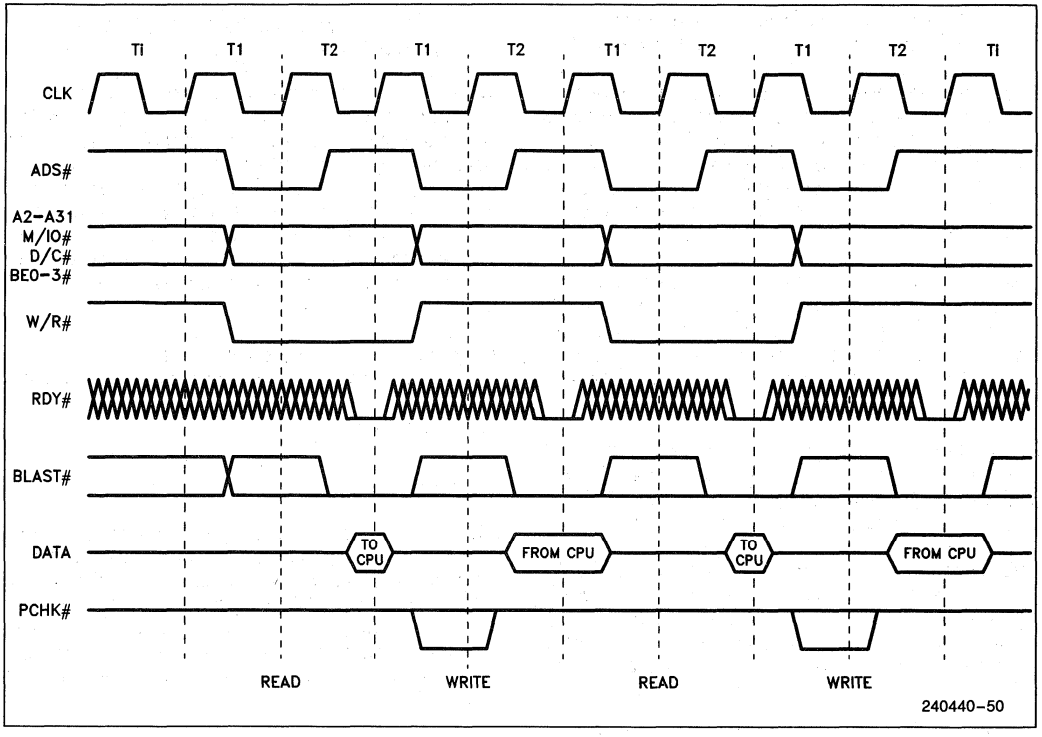


Figure 7.7. Basic 2-2 Bus Cycle

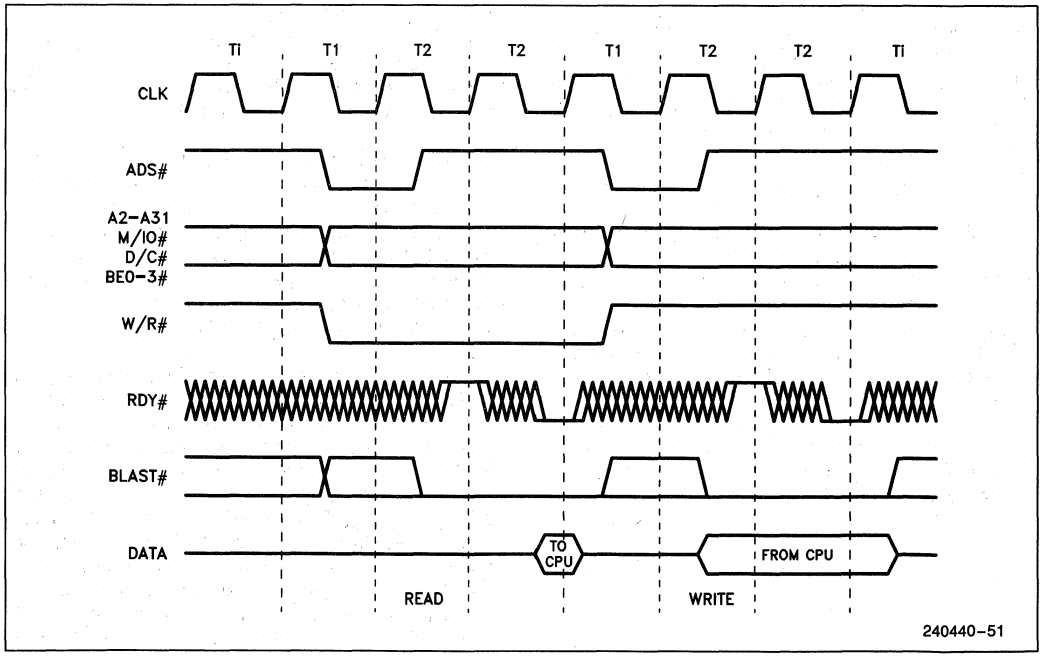


Figure 7.8. Basic 3-3 Bus Cycle

Burst cycles are not limited to cache line fills. Any multiple cycle read request by the 486 microprocessor can be converted into a burst cycle. The 486 microprocessor will only burst the number of bytes needed to complete a transfer. For example, eight bytes will be bursted in for a 64-bit floating point non-cacheable read.

The external system converts a multiple cycle request into a burst cycle by returning BRDY# active rather than RDY# (non-burst ready) in the first cycle of a transfer. For cycles that cannot be bursted such as interrupt acknowledge and halt, BRDY# has the same effect as RDY#. BRDY# is ignored if both BRDY# and RDY# are returned in the same clock. Memory areas and peripheral devices that cannot perform bursting must terminate cycles with RDY#.

7.2.2.2 Terminating Multiple and Burst Cycle Transfers

The 486 microprocessor drives BLAST# inactive for all but the last cycle in a multiple cycle transfer. BLAST# is driven inactive in the first cycle to inform the external system that the transfer could take additional cycles. BLAST# is driven active in the last cycle of the transfer indicating that the next time BRDY# or RDY# is returned the transfer is complete.

BLAST# is not valid in the first clock of a bus cycle. It should be sampled only in the second and subsequent clocks when RDY# or BRDY# is returned.

The number of cycles in a transfer is a function of several factors including the number of bytes the microprocessor needs to complete an internal request (1, 2, 4, 8, or 16), the state of the bus size inputs (BS8# and BS16#), the state of the cache enable input (KEN#) and alignment of the data to be transferred.

When the 486 microprocessor initiates a request it knows how many bytes will be transferred and if the data is aligned. The external system must tell the microprocessor whether the data is cacheable (if the transfer is a read) and the width of the bus by returning the state of the KEN#, BS8# and BS16# inputs one clock before RDY# or BRDY# is returned. The 486 microprocessor determines how many cycles a transfer will take based on its internal information and inputs from the external system.

BLAST# is not valid in the first clock of a bus cycle because the 486 microprocessor cannot determine the number of cycles a transfer will take until the external system returns KEN#, BS8# and BS16#. BLAST# should only be sampled in the second and subsequent clocks of a cycle when the external system returns RDY# or BRDY#.

The system may terminate a burst cycle by returning RDY# instead of BRDY#. BLAST# will remain deasserted until the last transfer. However, any transfers required to complete a cache line fill will follow the burst order, e.g., if burst order was 4, 0, C, 8 and RDY# was returned at after 0, the next transfers will be from C and 8.

7.2.2.3 Non-Cacheable, Non-Burst, Multiple Cycle Transfers

Figure 7.9 illustrates a 2 cycle non-burst, non-cacheable multiple cycle read. This transfer is simply a sequence of two single cycle transfers. The 486 microprocessor indicates to the external system that this is a multiple cycle transfer by driving BLAST# inactive during the second clock of the first cycle. The external system returns RDY# active indicating that it will not burst the data. The external system also indicates that the data is not cacheable by returning KEN# inactive one clock before it returns RDY# active. When the 486 microprocessor samples RDY# active it ignores BRDY#.

Each cycle in the transfer begins when ADS# is driven active and the cycle is complete when the external system returns RDY# active.

The 486 microprocessor indicates the last cycle of the transfer by driving BLAST# active. The next RDY# returned by the external system terminates the transfer.

7.2.2.4 Non-Cacheable Burst Cycles

The external system converts a multiple cycle request into a burst cycle by returning BRDY# active rather than RDY# in the first cycle of the transfer. This is illustrated in Figure 7.10.

There are several features to note in the burst read. ADS# is only driven active during the first cycle of the transfer. RDY# must be driven inactive when BRDY# is returned active.

BLAST# behaves exactly as it does in the non-burst read. BLAST# is driven inactive in the second clock of the first cycle of the transfer indicating more cycles to follow. In the last cycle, BLAST# is driven active telling the external memory system to end the burst after returning the next BRDY#.

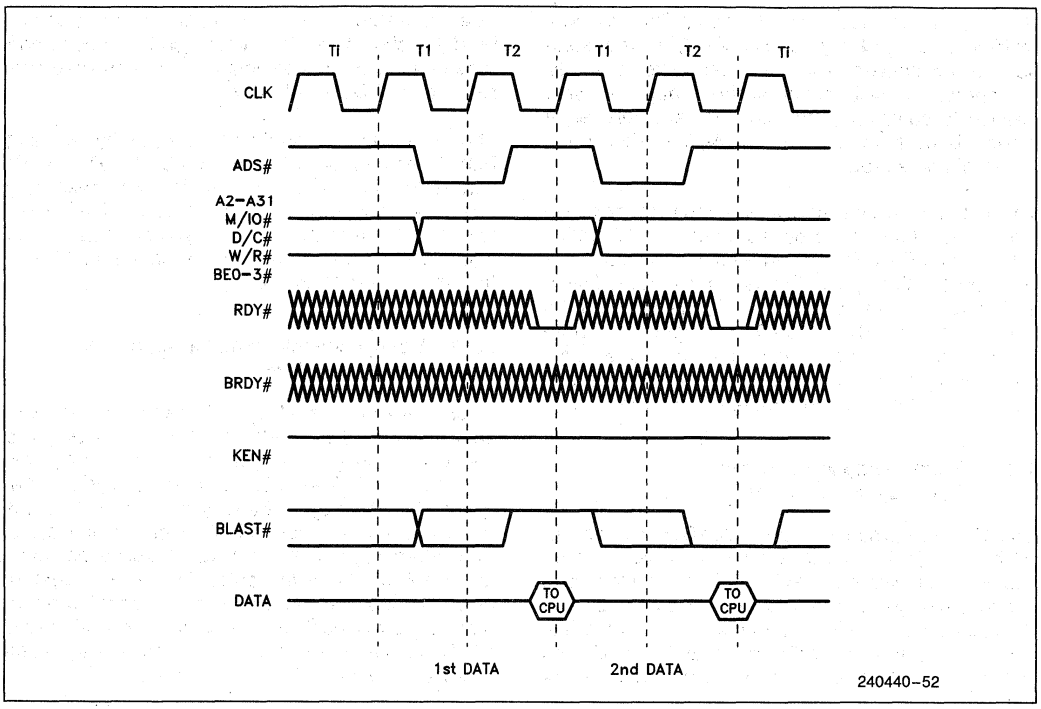


Figure 7.9. Non-Cacheable, Non-Burst, Multiple Cycle Transfers

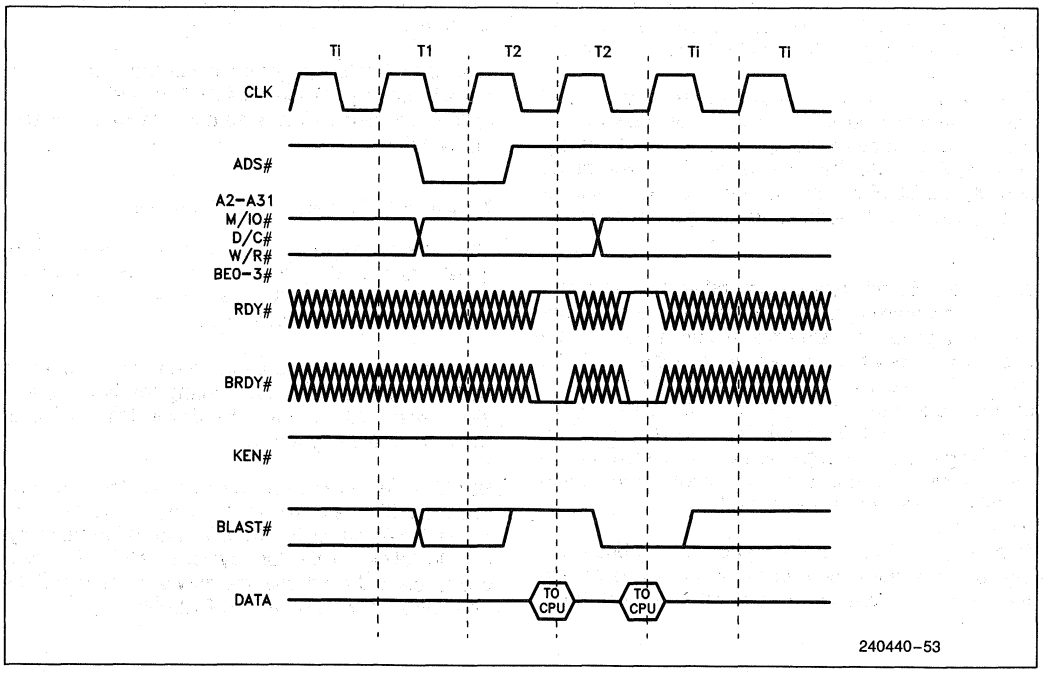


Figure 7.10. Non-Cacheable Burst Cycle

7.2.3 CACHEABLE CYCLES

Any memory read can become a cache fill operation. The external memory system can allow a read request to fill a cache line by returning KEN# active one clock before RDY# or BRDY# during the first cycle of the transfer on the external bus. Once KEN# is asserted and the remaining three requirements described below are met, the 486 microprocessor will fetch an entire cache line regardless of the state of KEN#. KEN# must be returned active in the last cycle of the transfer for the data to be written into the internal cache. The 486 microprocessor will only convert memory reads or prefetches into a cache fill.

KEN# is ignored during write or I/O cycles. Memory writes will only be stored in the on-chip cache if there is a cache hit. I/O space is never cached in the internal cache.

To transform a read or a prefetch into a cache line fill the following conditions must be met:

1. The KEN# pin must be asserted one clock prior to RDY# or BRDY# being returned for the first data cycle.
2. The cycle must be of the type that can be internally cached. (Locked reads, I/O reads, and interrupt acknowledge cycles are never cached).
3. The page table entry must have the page cache disable bit (PCD) set to 0. To cache a page table entry, the page directory must have PCD=0. To cache reads or prefetches when paging is disabled, or to cache the page directory entry, control register 3 (CR3) must have PCD=0.
4. The cache disable (CD) bit in control register 0 (CR0) must be clear.

External hardware can determine when the 486 microprocessor has transformed a read or prefetch into a cache fill by examining the KEN#, M/IO#, D/C#, W/R#, LOCK#, and PCD pins. These pins convey to the system the outcome of conditions 1–3 in the above list. In addition, the 486 drives PCD high whenever the CD bit in CR0 is set, so that external hardware can evaluate condition 4.

Cacheable cycles can be burst or non-burst.

7.2.3.1 Byte Enables during a Cache Line Fill

For the first cycle in the line fill, the state of the byte enables should be ignored. In a non-cacheable memory read, the byte enables indicate the bytes actually required by the memory or code fetch.

The 486 microprocessor expects to receive valid data on its entire bus (32 bits) in the first cycle of a cache line fill. Data should be returned with the assumption that all the byte enable pins are driven active. However if BS8# is asserted only one byte need be returned on data lines D0–D7. Similarly if BS16# is asserted two bytes should be returned on D0–D15.

The 486 microprocessor will generate the addresses and byte enables for all subsequent cycles in the line fill. The order in which data is read during a line fill depends on the address of the first item read. Byte ordering is discussed in Section 7.2.4.

7.2.3.2 Non-Burst Cacheable Cycles

Figure 7.11 shows a non-burst cacheable cycle. The cycle becomes a cache fill when the 486 microprocessor samples KEN# active at the end of the first clock. The 486 microprocessor drives BLAST# inactive in the second clock in response to KEN#. BLAST# is driven inactive because a cache fill requires 3 additional cycles to complete. BLAST# remains inactive until the last transfer in the cache line fill. KEN# must be returned active in the last cycle of the transfer for the data to be written into the internal cache.

Note that this cycle would be a single bus cycle if KEN# was not sampled active at the end of the first clock. The subsequent three reads would not have happened since a cache fill was not requested.

The BLAST# output is invalid in the first clock of a cycle. BLAST# may be active during the first clock due to earlier inputs. Ignore BLAST# until the second clock.

During the first cycle of the cache line fill the external system should treat the byte enables as if they are all active. In subsequent cycles in the burst, the 486 microprocessor drives the address lines and byte enables (see Section 7.2.4.2 for **Burst and Cache Line Fill Order**).

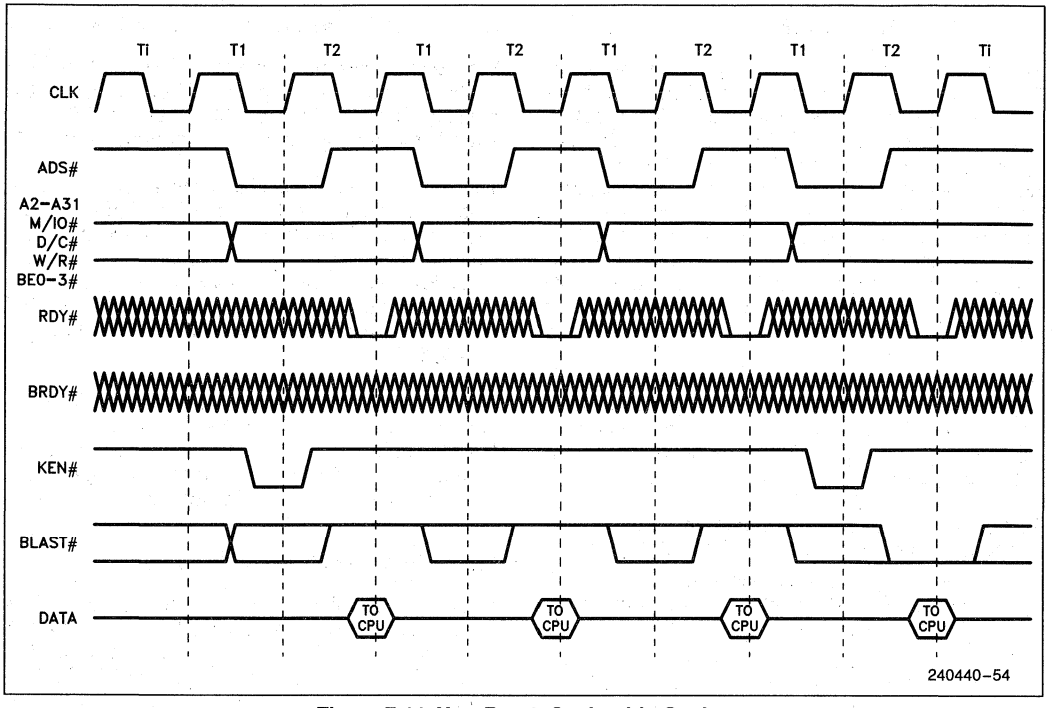


Figure 7.11. Non-Burst, Cacheable Cycles

7.2.3.3 Burst Cacheable Cycles

Figure 7.12 illustrates a burst mode cache fill. As in Figure 7.11, the transfer becomes a cache line fill when the external system returns KEN# active at the end of the first clock in the cycle.

The external system informs the 486 microprocessor that it will burst the line in by driving BRDY# active at the end of the first cycle in the transfer.

Note that during a burst cycle ADS# is only driven with the first address.

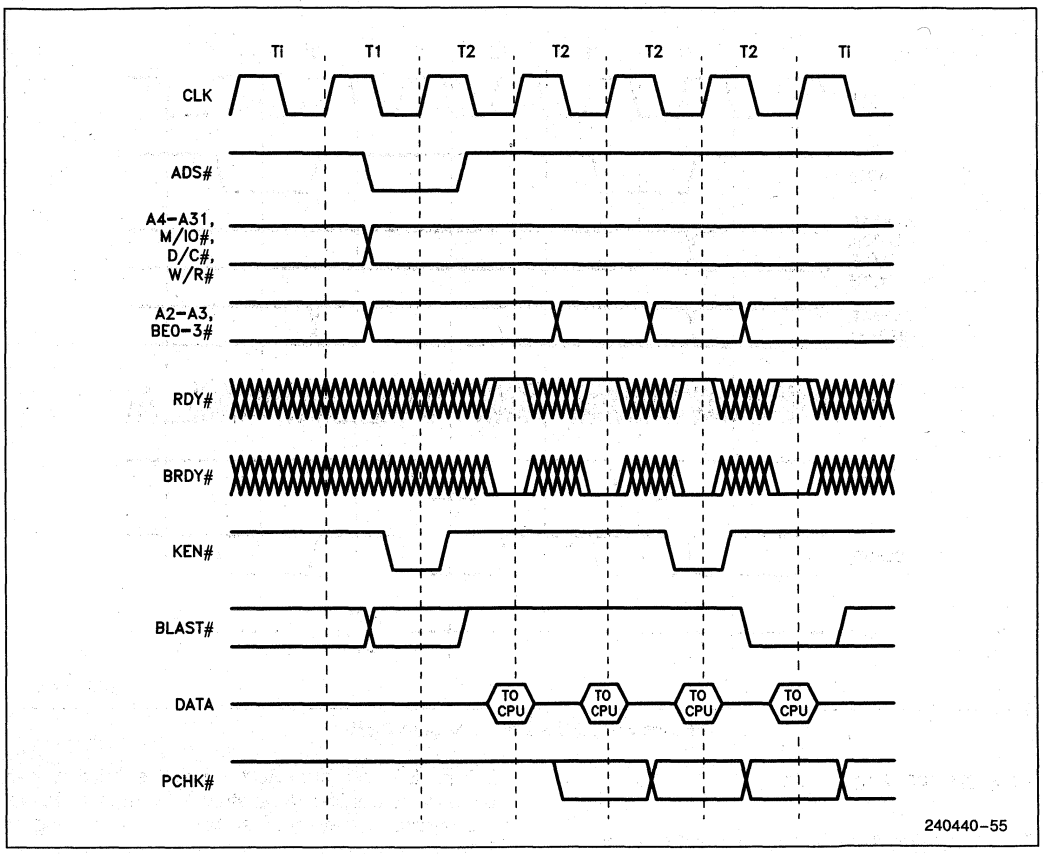


Figure 7.12. Burst Cacheable Cycle

7.2.3.4 Effect of Changing KEN# during a Cache Line Fill

KEN# can change multiple times as long as it arrives at its final value in the clock before RDY# or BRDY# is returned. This is illustrated in Figure 7.13. Note that the timing of BLAST# follows that of KEN# by one clock. The i486 samples KEN# every clock and uses the value returned in the clock before ready to determine if a bus cycle would be a cache line fill. Similarly, it uses the value of KEN# in the last cycle, before early RDY# to load the line just retrieved from the memory into the cache. KEN# is sampled every clock, it must satisfy setup and hold time.

KEN# can also change multiple times before a burst cycle as long as it arrives at its final value one clock before ready is returned active.

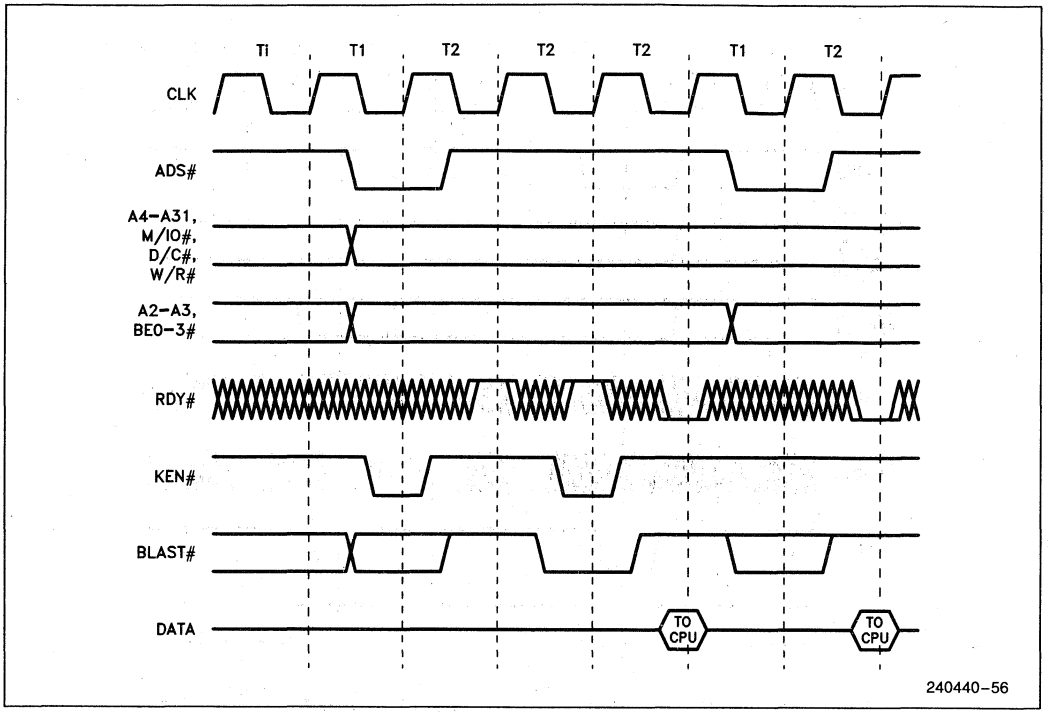


Figure 7.13. Effect of Changing KEN#

7.2.4 BURST MODE DETAILS

7.2.4.1 Adding Wait States to Burst Cycles

Burst cycles need not return data on every clock. The 486 microprocessor will only strobe data into the chip when either RDY# or BRDY# are active. Driving BRDY# and RDY# inactive adds a wait state to the transfer. A burst cycle where two clocks are required for every burst item is shown in Figure 7.14.

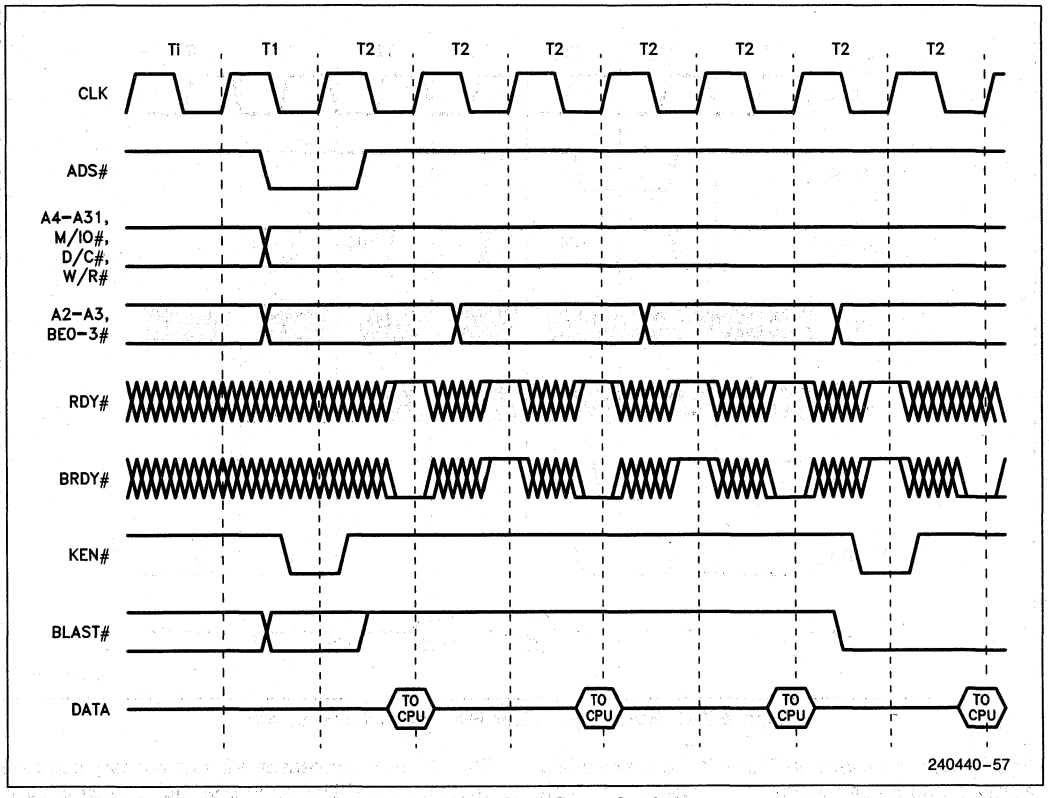


Figure 7.14. Slow Burst Cycle

7.2.4.2 Burst and Cache Line Fill Order

The burst order used by the 486 microprocessor is shown in Table 7.7. This burst order is followed by any burst cycle (cache or not), cache line fill (burst or not) or code prefetch.

The microprocessor presents each request for data in an order determined by the first address in the transfer. For example, if the first address was 104 the next three addresses in the burst will be 100, 10C and 108.

Table 7.7. Burst Order

First Addr.	Second Addr.	Third Addr.	Fourth Addr.
0	4	8	C
4	0	C	8
8	C	0	4
C	8	4	0

An example of burst address sequencing is shown in Figure 7.15.

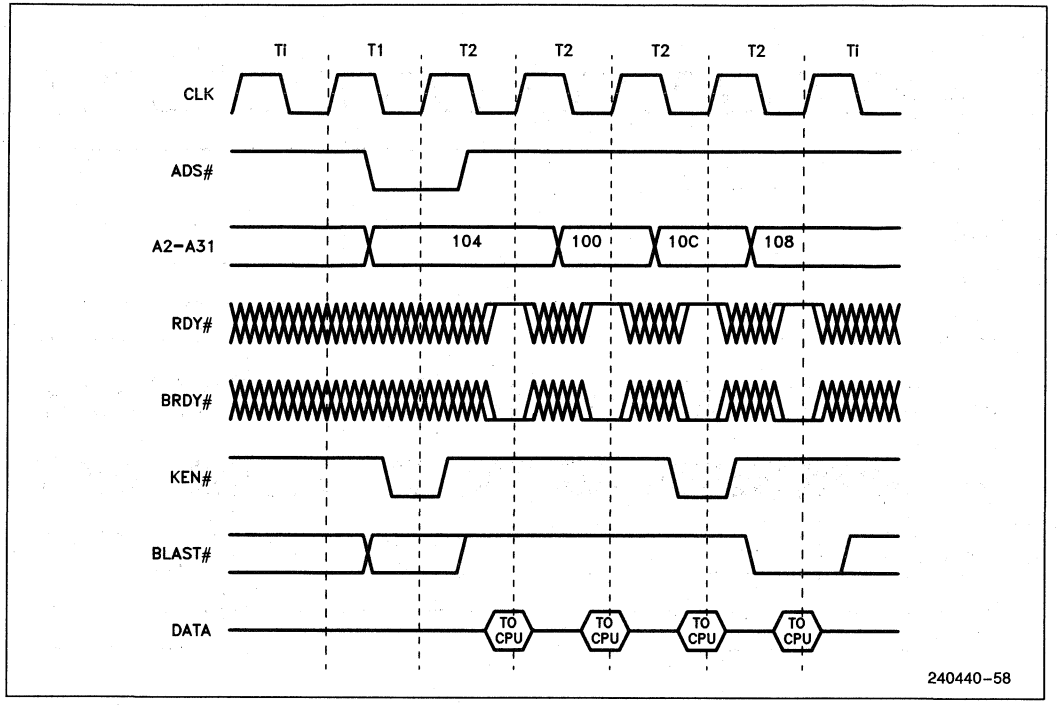


Figure 7.15. Burst Cycle Showing Order of Addresses

The sequences shown in Table 7.7 accommodate systems with 64-bit busses as well as systems with 32-bit data busses. The sequence applies to all bursts, regardless of whether the purpose of the burst is to fill a cache line, do a 64-bit read, or do a pre-fetch. If either BS8# or BS16# is returned active, the 486 microprocessor completes the transfer of the current 32-bit word before progressing to the next 32-bit word. For example, a BS16# burst to address 4 has the following order: 4-6-0-2-C-E-8-A.

7.2.4.3 Interrupted Burst Cycles

Some memory systems may not be able to respond with burst cycles in the order defined in Table 7.7. To support these systems the 486 microprocessor allows a burst cycle to be interrupted at any time. The 486 microprocessor will automatically generate another normal bus cycle after being interrupted to complete the data transfer. This is called an interrupted burst cycle. The external system can respond to an interrupted burst cycle with another burst cycle.

The external system can interrupt a burst cycle by returning RDY# instead of BRDY#. RDY# can be returned after any number of data cycles terminated with BRDY#.

An example of an interrupted burst cycle is shown in Figure 7.16. The 486 microprocessor immediately drives ADS# active to initiate a new bus cycle after RDY# is returned active. BLAST# is driven inactive one clock after ADS# begins the second bus cycle indicating that the transfer is not complete.

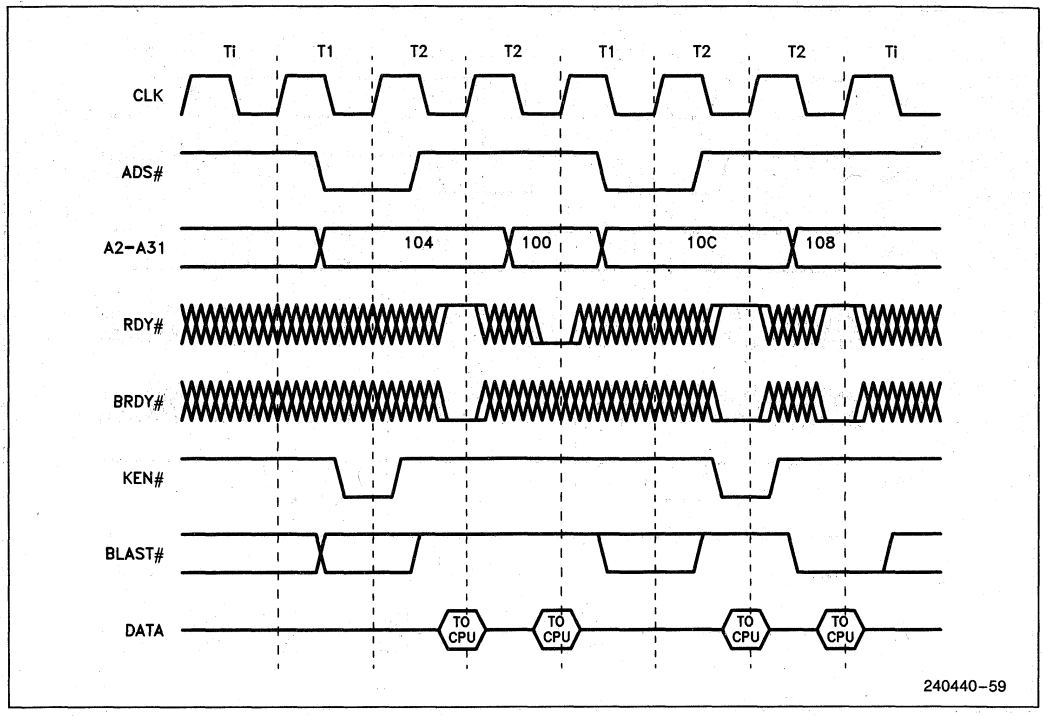


Figure 7.16. Interrupted Burst Cycle

KEN# need not be returned active in the first data cycle of the second part of the transfer in Figure 7.16. The cycle had been converted to a cache fill in the first part of the transfer and the 486 microprocessor expects the cache fill to be completed. Note that the first half and second half of the transfer in Figure 7.16 are each two cycle burst transfers.

The order in which the 486 microprocessor requests operands during an interrupted burst transfer is determined by Table 7.7. Mixing RDY# and BRDY# does not change the order in which operand addresses are requested by the 486 microprocessor.

An example of the order in which the 486 microprocessor requests operands during a cycle in which the external system mixes RDY# and BRDY# is shown in Figure 7.17. The 486 microprocessor initially requests a transfer beginning at location 104. The transfer becomes a cache line fill when the external system returns KEN# active. The first cycle of the cache fill transfers the contents of location 104 and is terminated with RDY#. The 486 microprocessor drives out a new request (by asserting ADS#) to address 100. If the external system terminates the second cycle with BRDY#, the 486 microprocessor will next request/expect address 10C. The correct order is determined by the first cycle in the transfer, which may not be the first cycle in the burst if the system mixes RDY# with BRDY#.

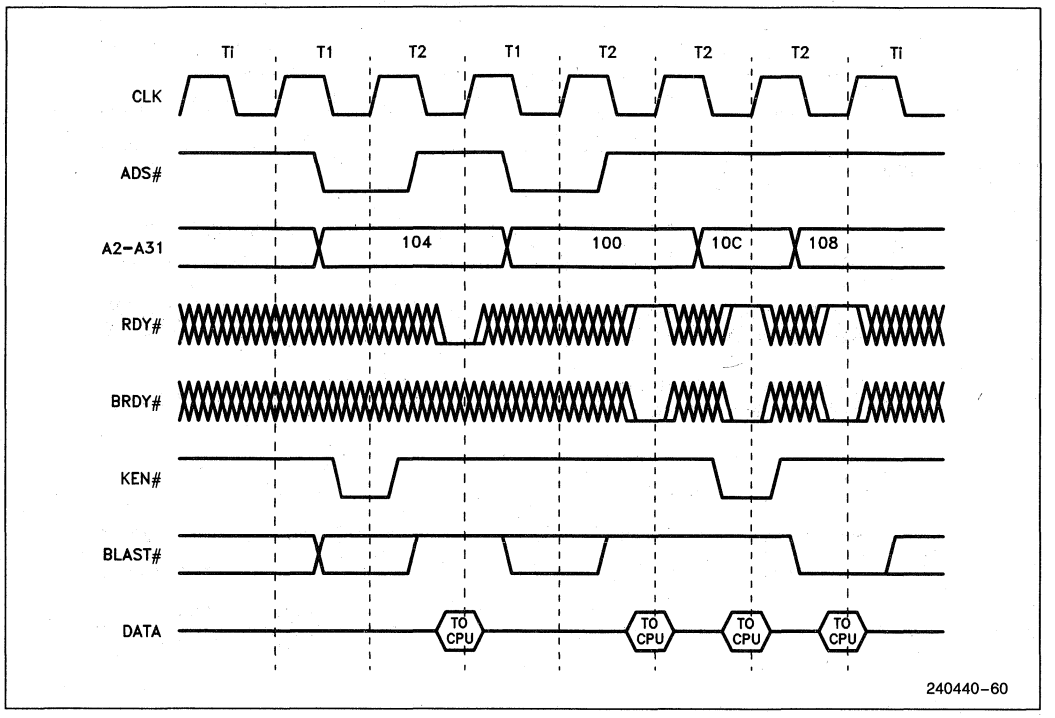


Figure 7.17. Interrupted Burst Cycle with Unobvious Order of Addresses

7.2.5 8- AND 16-BIT CYCLES

The 486 microprocessor supports both 16- and 8-bit external busses through the BS16# and BS8# inputs. BS16# and BS8# allow the external system to specify, on a cycle by cycle basis, whether the addressed component can supply 8, 16 or 32 bits. BS16# and BS8# can be used in burst cycles as well as non-burst cycles. If both BS16# and BS8# are returned active for any bus cycle, the 486 microprocessor will respond as if only BS8# were active.

The timing of BS16# and BS8# is the same as that of KEN#. BS16# and BS8# must be driven active before the first RDY# or BRDY# is driven active. Driving the BS16# and BS8# active can force the 486 microprocessor to run additional cycles to complete what would have been only a single 32-bit cycle. BS8# and BS16# may change the state of BLAST# when they force subsequent cycles from the transfer.

Figure 7.18 shows an example in which BS8# forces the 486 microprocessor to run two extra cycles to complete a transfer. The 486 microprocessor issues a request for 24 bits of information. The external system drives BS8# active indicating that only eight bits of data can be supplied per cycle. The 486 microprocessor issues two extra cycles to complete the transfer.

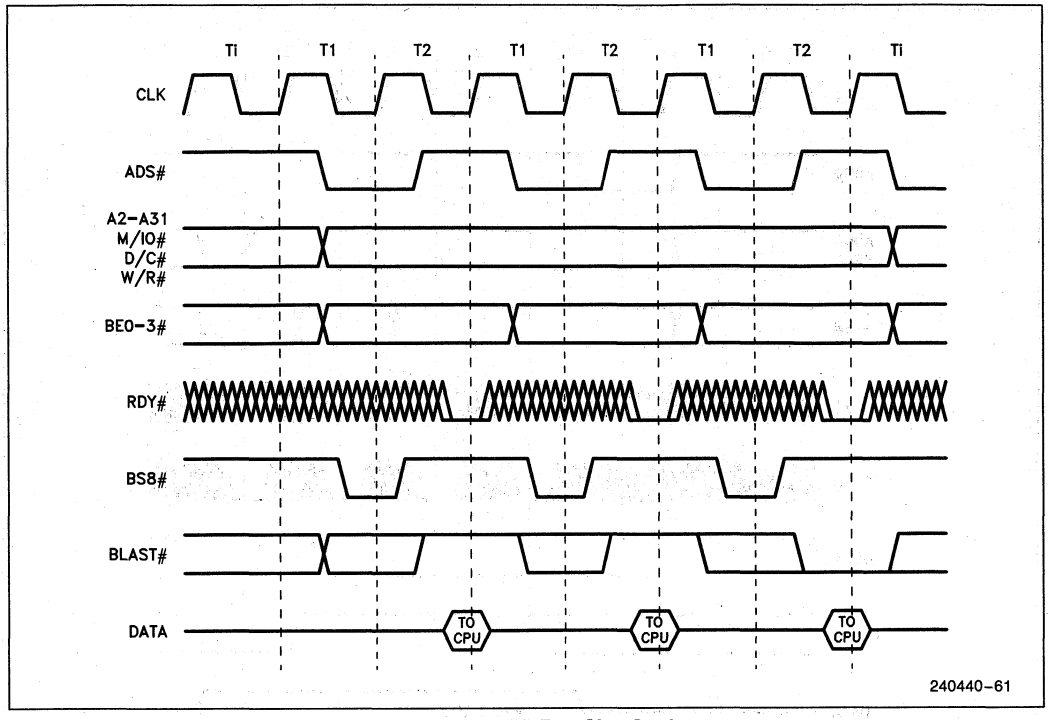


Figure 7.18. 8-Bit Bus Size Cycle

Extra cycles forced by the BS16# and BS8# should be viewed as independent bus cycles. BS16# and BS8# should be driven active for each additional cycle unless the addressed device has the ability to change the number of bytes it can return between cycles. The 486 microprocessor will drive BLAST# inactive until the last cycle before the transfer is complete.

Refer to Section 7.1.3 for the sequencing of addresses while BS8# or BS16# are active.

BS8# and BS16# operate during burst cycles in exactly the same manner as non-burst cycles. For example, a single non-cacheable read could be transferred by the 486 microprocessor as four 8-bit burst data cycles. Similarly, a single 32-bit write could be written as four 8-bit burst data cycles. An example of a burst write is shown in Figure 7.19. Burst writes can only occur if BS8# or BS16# is asserted.

5

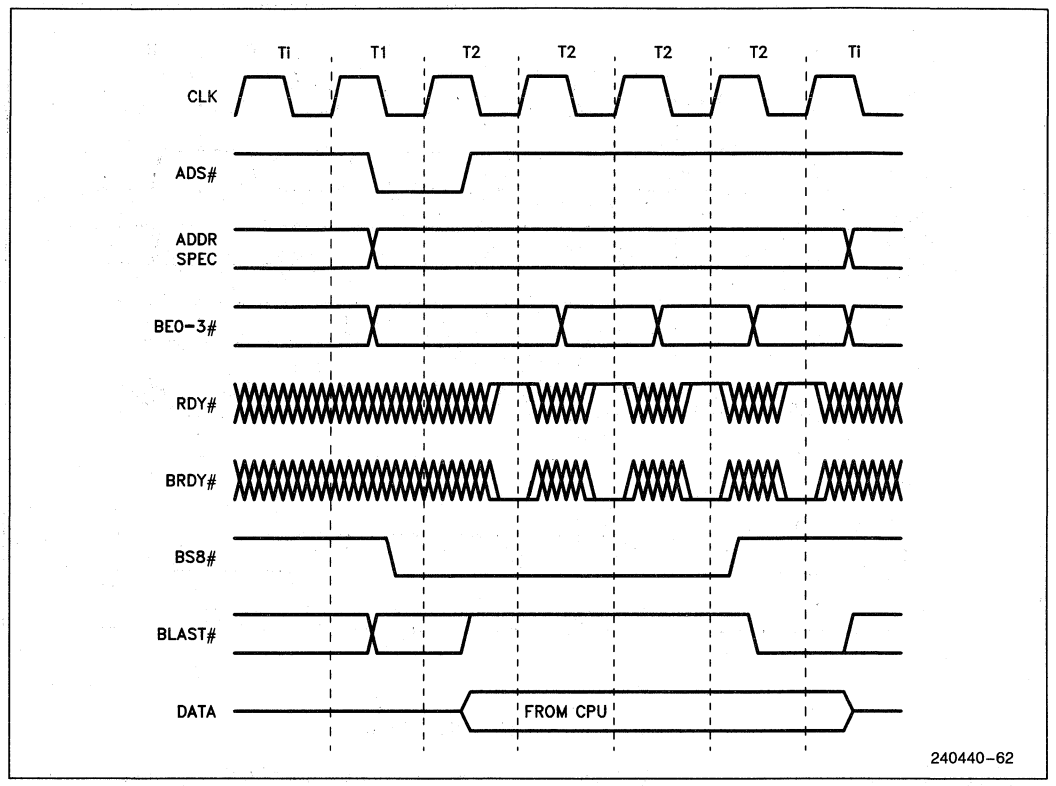


Figure 7.19. Burst Write as a Result of BS8# or BS16#

7.2.6 LOCKED CYCLES

Locked cycles are generated in software for any instruction that performs a read-modify-write operation. During a read-modify-write operation the processor can read and modify a variable in external memory and be assured that the variable is not accessed between the read and write.

Locked cycles are automatically generated during certain bus transfers. The xchg (exchange) instruction generates a locked cycle when one of its operands is memory based. Locked cycles are generated when a segment or page table entry is updated and during interrupt acknowledge cycles. Locked cycles are also generated when the LOCK instruction prefix is used with selected instructions.

Locked cycles are implemented in hardware with the LOCK# pin. When LOCK# is active, the processor is performing a read-modify-write operation and the external bus should not be relinquished until the cycle is complete. Multiple reads or writes can be locked. A locked cycle is shown in Figure 7.20. LOCK# goes active with the address and bus definition pins at the beginning of the first read cycle and remains active until RDY# is returned for the last write cycle. For unaligned 32 bits read-modify-write operation, the LOCK# remains active for the entire duration of the multiple cycle. It will go inactive when RDY# is returned for the last write cycle.

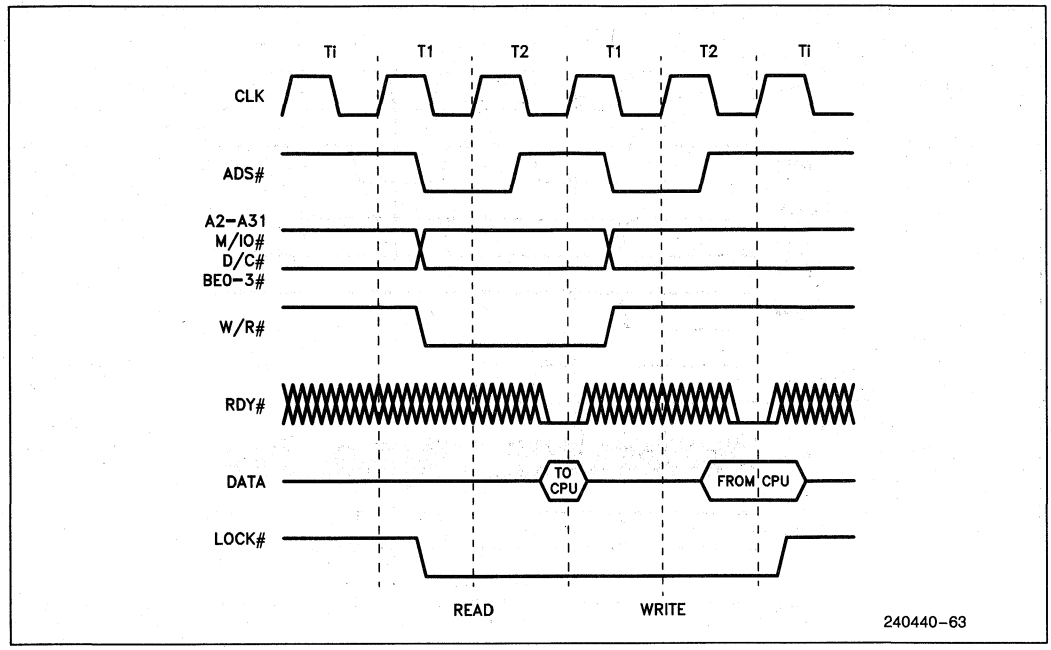


Figure 7.20. Locked Bus Cycle

When LOCK# is active, the 486 microprocessor will recognize address hold and backoff but will not recognize bus hold. It is left to the external system to properly arbitrate a central bus when the 486 microprocessor generates LOCK#.

7.2.7 PSEUDO-LOCKED CYCLES

Pseudo-locked cycles assure that no other master will be given control of the bus during operand transfers which take more than one bus cycle. Examples include 64-bit floating point read and writes, 64-bit descriptor loads and cache line fills.

Pseudo-locked transfers are indicated by the PLOCK# pin. The memory operands must be aligned for correct operation of a pseudo-locked cycle.

PLOCK# need not be examined during burst reads. A 64-bit aligned operand can be retrieved in one burst (note: this is only valid in systems that do not interrupt bursts).

The system must examine PLOCK# during 64-bit writes since the 486 microprocessor cannot burst write more than 32 bits. However, burst can be used within each 32-bit write cycle if BS8# or BS16# is asserted. BLAST will be deasserted in response to BS8# or BS16#. A 64-bit write will be driven out as two non-burst bus cycles. BLAST# is asserted during both writes since a burst is not possible. PLOCK# is asserted during the first write to indicate that another write follows. This behavior is shown in Figure 7.21.

The first cycle of a 64-bit floating point write is the only case in which both PLOCK# and BLAST# are asserted. Normally PLOCK# and BLAST# are the inverse of each other.

During all of the cycles where PLOCK# is asserted, HOLD is not acknowledged until the cycle completes. This results in a large HOLD latency, especially when BS8# or BS16# is asserted. To reduce the HOLD latency during these cycles, windows are available between transfers to allow HOLD to be acknowledged during non-cacheable code prefetches. PLOCK# will be asserted since BLAST# is negated, but it is ignored and HOLD is recognized during the prefetch.

PLOCK# can change several times during a cycle settling to its final value in the clock ready is returned.

7.2.8 INVALIDATE CYCLES

Invalidate cycles are needed to keep the 486 microprocessor's internal cache contents consistent with external memory. The 486 microprocessor contains a mechanism for listening to writes by other devices to external memory. When the processor finds a write to a Section of external memory contained in

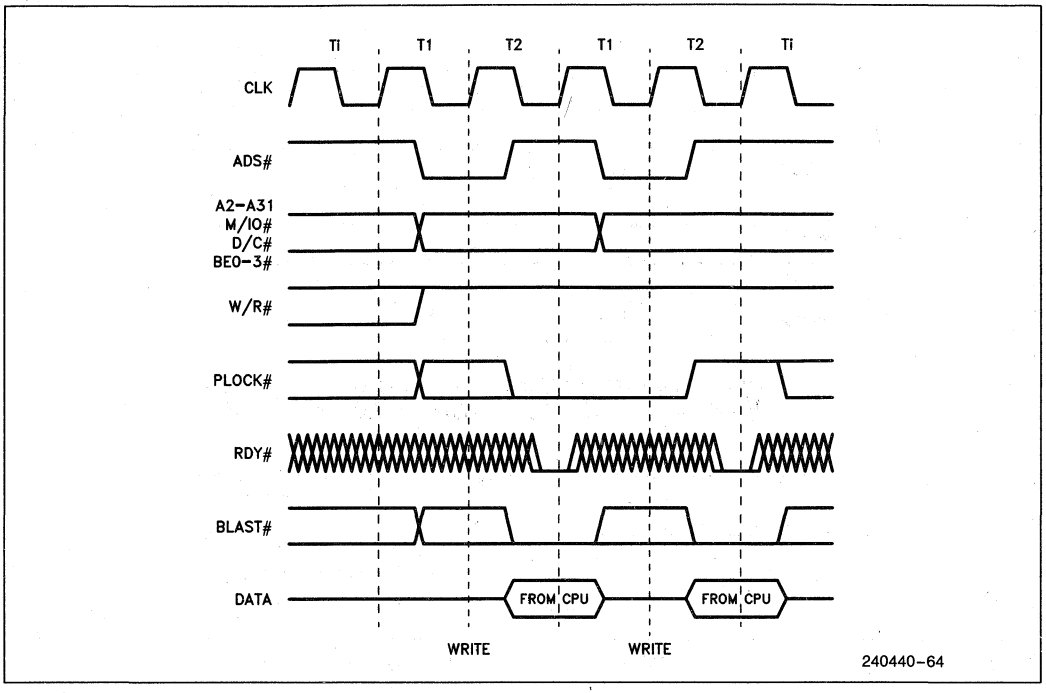


Figure 7.21. Pseudo Lock Timing

its internal cache, the processor's internal copy is invalidated.

Invalidations use two pins, address hold request (AHOLD) and valid external address (EADS#). There are two steps in an invalidation cycle. First, the external system asserts the AHOLD input forcing the 486 microprocessor to immediately relinquish its address bus. Next, the external system asserts EADS# indicating that a valid address is on the 486 microprocessor's address bus. EADS# and the invalidation address, Figure 7-22 shows the fastest possible invalidation cycle. The i486 cycle CPU recognizes AHOLD on one CLK edge and floats the address bus in response. To allow the address bus to float and avoid contention, EADS# and the invalidation address should not be driven until the following CLK edge. The microprocessor reads the address over its address lines. If the microprocessor finds this address in its internal cache, the cache entry is invalidated. Note that the 486 microprocessor's address bus is input/output unlike the 386 microprocessor's bus, which is output only.

The 486 microprocessor immediately relinquishes its address bus in the next clock upon assertion of AHOLD. For example, the bus could be 3 wait states into a read cycle. If AHOLD is activated, the 486 microprocessor will immediately float its address bus before ready is returned terminating the bus cycle.

When AHOLD is asserted only the address bus is floated, the data bus can remain active. Data can be returned for a previously specified bus cycle during address hold (see Figures 7.22, 7.23).

EADS# is normally asserted when an external master drives an address onto the bus. AHOLD need not be driven for EADS# to generate an internal invalidate. If EADS# alone is asserted while the 486 microprocessor is driving the address bus, it is possible that the invalidation address will come from the 486 microprocessor itself.

Note that it is also possible to run an invalidation cycle by asserting EADS# when HOLD or BUFF# is asserted.

Running an invalidate cycle prevents the 486 microprocessor cache from satisfying other internal requests, so invalidations should be run only when necessary. The fastest possible invalidate cycle is shown in Figure 7.22, while a more realistic invalidation cycle is shown in 7.23. Both of the examples take one clock of cache access from the rest of the 486 microprocessor.

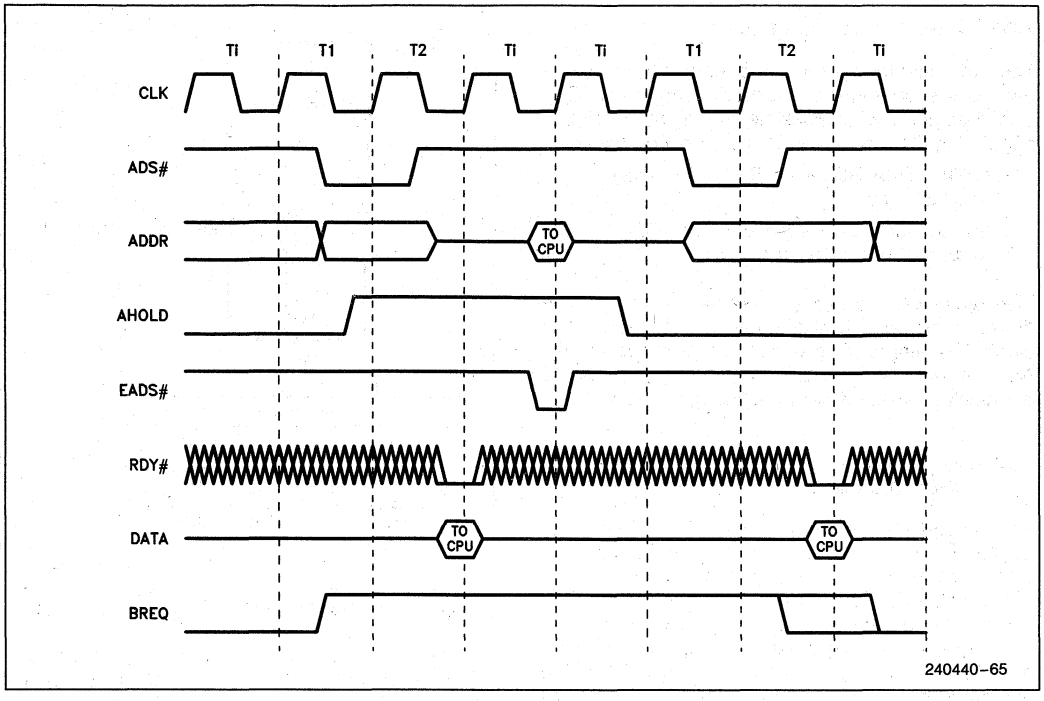


Figure 7.22. Fast Internal Cache Invalidation Cycle

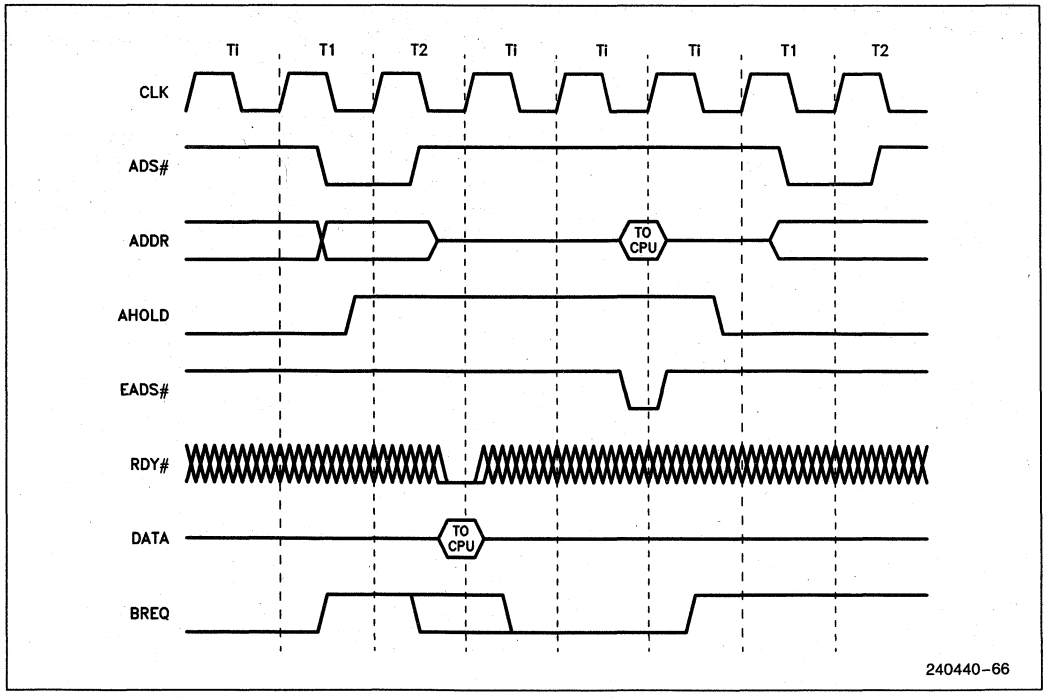


Figure 7.23. Typical Internal Cache Invalidation Cycle

7.2.8.1 Rate of Invalidate Cycles

The 486 microprocessor can accept one invalidate per clock except in the last clock of a line fill. One invalidate per clock is possible as long as EADS# is negated in ONE or BOTH of the following cases:

1. In the clock RDY# or BRDY# is returned for the last time.
2. In the clock following RDY# or BRDY# being returned for the last time.

This definition allows two system designs. Simple designs can restrict invalidates to one every other clock. The simple design need not track bus activity. Alternatively, systems can request one invalidate per clock provided that the bus is monitored.

7.2.8.2 Running Invalidate Cycles Concurrently with Line Fills

Precautions are necessary to avoid caching stale data in the 486 microprocessor's cache in a system with a second level cache. An example of a system with a second level cache is shown in Figure 7.24. An external device can be writing to main memory over the system bus while the 486 microprocessor is retrieving data from the second level cache. The 486 microprocessor will need to invalidate a line in its internal cache if the external device is writing to a main memory address also contained in the 486 microprocessor's cache.

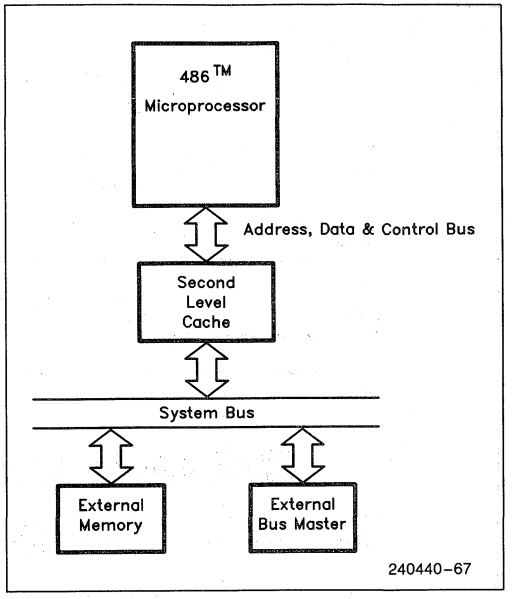
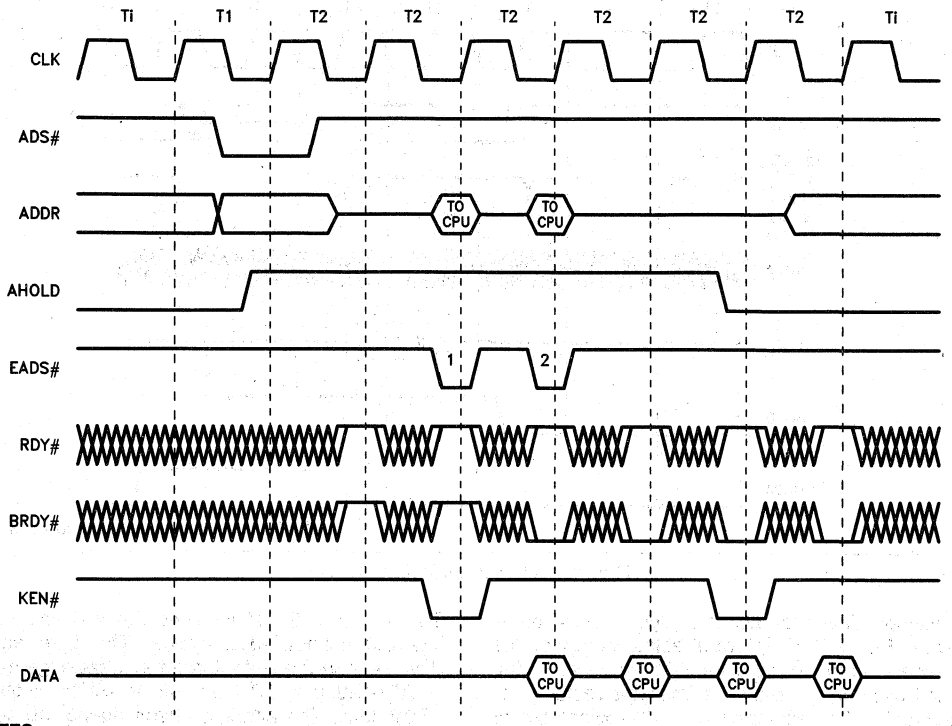


Figure 7.24. System with Second Level Cache

A potential problem exists if the external device is writing to an address in external memory, and at the same time the 486 microprocessor is reading data from the same address in the second level cache. The system must force an invalidation cycle to invalidate the data that the 486 microprocessor has requested during the line fill.

If the system asserts EADS# before the first data in the line fill is returned to the 486 microprocessor, the system must return data consistent with the new data in the external memory upon resumption of the line fill after the invalidation cycle. This is illustrated by the asserted EADS# signal labeled 1 in Figure 7.25.



NOTES:
1. Data returned must be consistent if its address equals the invalidation address in this clock
2. Data returned will not be cached if its address equals the invalidation address in this clock

240440-68

Figure 7.25. Cache Invalidation Cycle Concurrent with Line Fill

If the system asserts EADS# at the same time or after the first data in the line fill is returned (in the same clock that the first RDY# or BRDY# is returned or any subsequent clock in the line fill) the data will be read into the 486 microprocessors input buffers but it will not be stored in the on-chip cache. This is illustrated by asserted EADS# signal labeled 2 in Figure 7.25. The stale data will be used to satisfy the request that initiated the cache fill cycle.

7.2.9 BUS HOLD

The 486 microprocessor provides a bus hold, hold acknowledge protocol using the bus hold request (HOLD) and bus hold acknowledge (HLDA) pins. Asserting the HOLD input indicates that another bus master desires control of the 486 microprocessor's bus. The processor will respond by floating its bus and driving HLDA active when the current bus cycle, or sequence of locked cycles is complete. An example of a HOLD/HLDA transaction is shown in Figure 7.26. Unlike the 386 microprocessor, the 486 microprocessor can respond to HOLD by floating its bus and asserting HLDA while RESET is asserted.

Note that HOLD will be recognized during un-aligned writes (less than or equal to 32-bits) with BLAST# being active for each write. For greater than 32-bit or un-aligned write, HOLD# recognition is prevented by PLOCK# getting asserted.

The pins floated during bus hold are: BE0#–BE3#, PCD, PWT, W/R#, D/C#, M/IO#, LOCK#, PLOCK#, ADS#, BLAST#, D0–D31, A2–A31, DP0–DP3.

7.2.10 INTERRUPT ACKNOWLEDGE

The 486 microprocessor generates interrupt acknowledge cycles in response to maskable interrupt requests generated on the interrupt request input (INTR) pin. Interrupt acknowledge cycles have a unique cycle type generated on the cycle type pins.

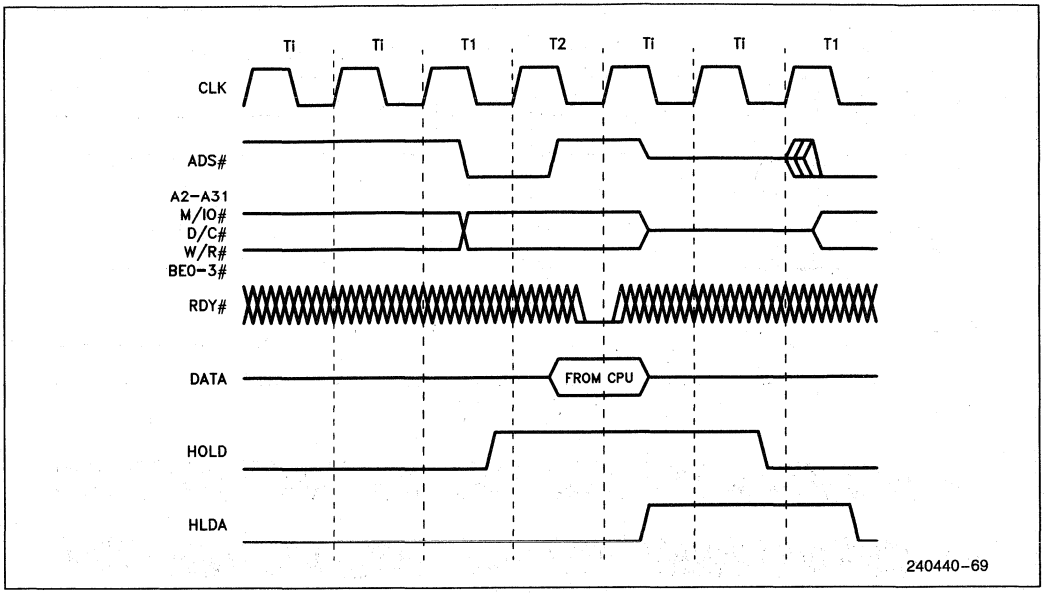


Figure 7.26. HOLD/HLDA Cycles

An example interrupt acknowledge transaction is shown in Figure 7.27. Interrupt acknowledge cycles are generated in locked pairs. Data returned during the first cycle is ignored. The interrupt vector is returned during the second cycle on the lower 8 bits of the data bus. The 486 microprocessor has 256 possible interrupt vectors.

The state of A2 distinguishes the first and second interrupt acknowledge cycles. The byte address driven during the first interrupt acknowledge cycle is 4 (A31–A3 low, A2 high, BE3#–BE1# high, and BE0# low). The address driven during the second interrupt acknowledge cycle is 0 (A31–A2 low, BE3#–BE1# high, BE0# low).

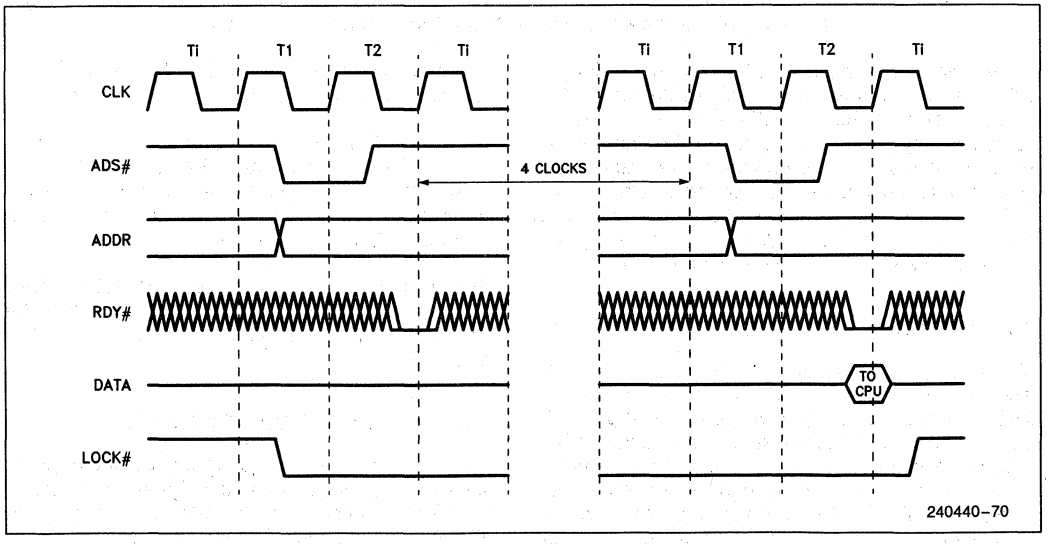


Figure 7.27. Interrupt Acknowledge Cycles

Each of the interrupt acknowledge cycles are terminated when the external system returns RDY# or BRDY#. Wait states can be added by withholding RDY# or BRDY#. The 486 microprocessor automatically generates four idle clocks between the first and second cycles to allow for 8259A recovery time.

7.2.11 SPECIAL BUS CYCLES

The 486 microprocessor provides four special bus cycles to indicate that certain instructions have been executed, or certain conditions have occurred internally. The special bus cycles in Table 7.8 are defined when the bus cycle definition pins are in the following state: M/IO# = 0, D/C# = 0 and W/R# = 1. During these cycles the address bus is driven low while the data bus is undefined.

Two of the special cycles indicate halt or shutdown. Another special cycle is generated when the 486 microprocessor executes an INVD (invalidate data cache) instruction and could be used to flush an external cache. The Write Back cycle is generated when the 486 microprocessor executes the WBINVD (write-back invalidate data cache) instruction and could be used to synchronize an external write-back cache.

The external hardware must acknowledge these special bus cycles by returning RDY# or BRDY#.

Table 7.8. Special Bus Cycle Encoding

BE3#	BE2#	BE1#	BE0#	Special Bus Cycle
1	1	1	0	Shutdown
1	1	0	1	Flush
1	0	1	1	Halt
0	1	1	1	Write Back

7.2.11.1 Halt Indication Cycle

The i486 Microprocessor halts as a result of executing a HALT instruction. Signaling its entrance into the halt state, a halt indication cycle is performed. The halt indication cycle is identified by the bus definition signals in special bus cycle state and a byte address of 2. BE0# and BE2# are the only signals distinguishing halt indication from shutdown indication, which drives an address of 0. During the halt cycle undefined data is driven on D0–D31. The halt indication cycle must be acknowledged by READY# asserted.

A halted i486 Microprocessor resumes execution when INTR (if interrupts are enabled) or NMI or RESET is asserted.

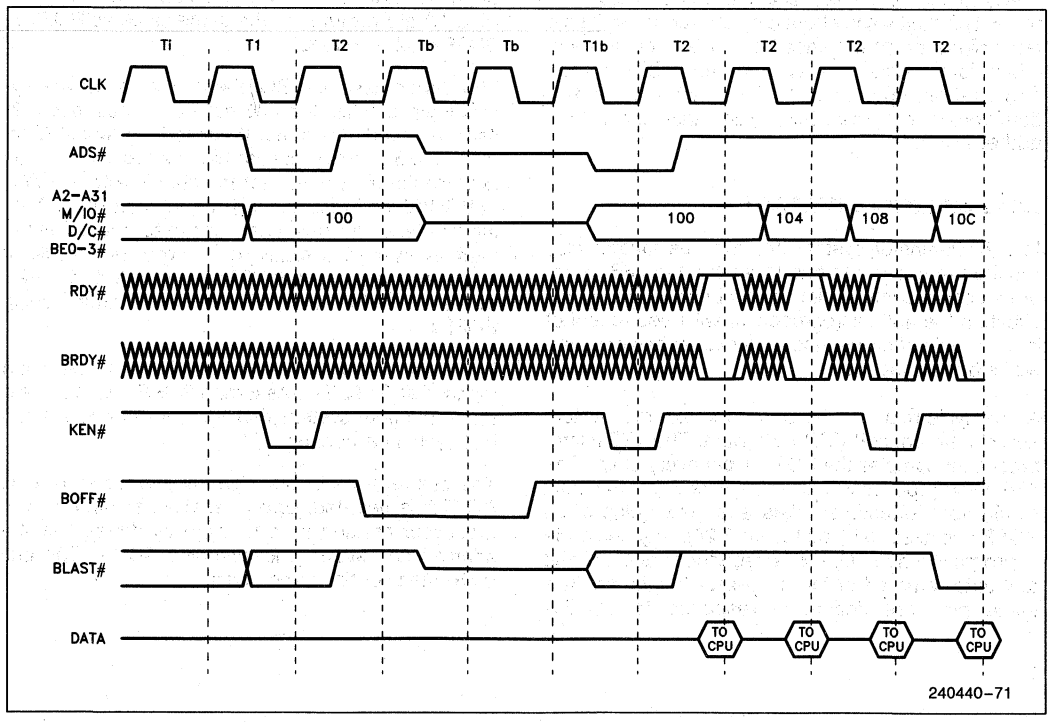


Figure 7.28. Restarted Read Cycle

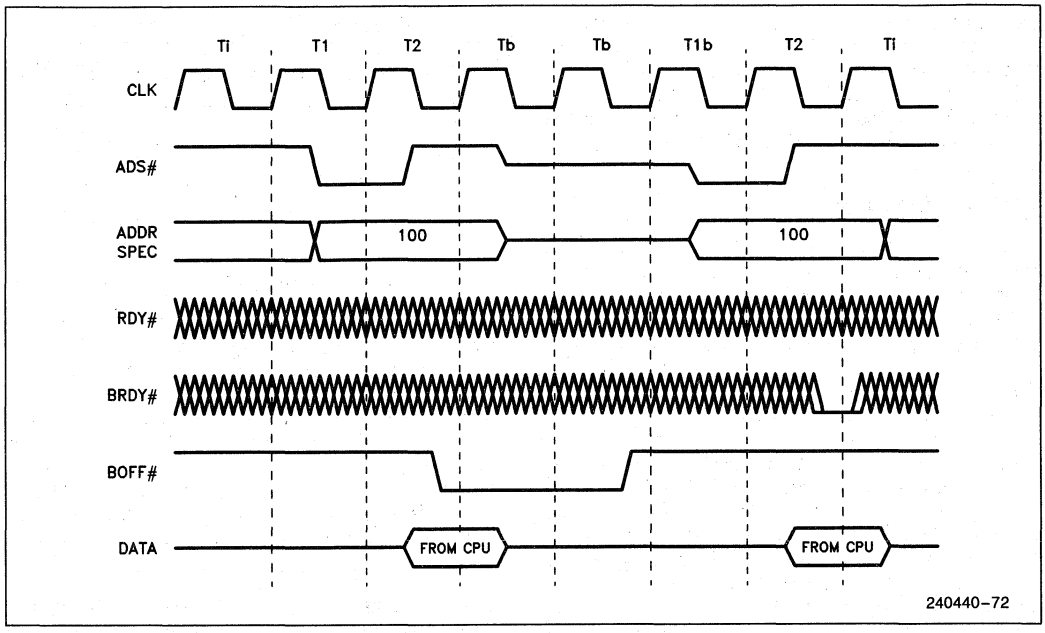


Figure 7.29. Restarted Write Cycle

7.2.11.2 Shutdown Indication Cycle

The i486 Microprocessor shuts down as a result of a protection fault while attempting to process a double fault. Signaling its entrance into the shutdown state, a shutdown indication cycle is performed. The shutdown indication cycle is identified by the bus definition signals in special bus cycle state and a byte address of 0.

7.2.12 BUS CYCLE RESTART

In a multi-master system another bus master may require the use of the bus to enable the 486 microprocessor to complete its current bus request. In this situation the 486 microprocessor will need to restart its bus cycle after the other bus master has completed its bus transaction.

A bus cycle may be restarted if the external system asserts the backoff (BOFF#) input. The 486 microprocessor samples the BOFF# pin every clock. The 486 microprocessor will immediately (in the next clock) float its address, data and status pins when BOFF# is asserted (see Figure 7.28). Any bus cycle in progress when BOFF# is asserted is aborted and any data returned to the processor is ignored. The same pins are floated in response to BOFF# as are floated in response to HOLD. HLDA is not generated in response to BOFF#. BOFF# has higher priority than RDY# or BRDY#. If either RDY# or BRDY# are returned in the same clock as BOFF#, BOFF# takes effect.

The device asserting BOFF# is free to run any cycles it wants while the 486 microprocessor bus is in its high impedance state. If backoff is requested after the 486 microprocessor has started a cycle, the new master should wait for memory to return RDY# or BRDY# before assuming control of the bus. Waiting for ready provides a handshake to insure that the memory system is ready to accept a new cycle. If the bus is idle when BOFF# is asserted, the new master can start its cycle two clocks after issuing BOFF#.

The external memory can view BOFF# in the same manner as BLAST#. Asserting BOFF# tells the external memory system that the current cycle is the last cycle in a transfer.

The bus remains in the high impedance state until BOFF# is negated. Upon negation, the 486 microprocessor restarts its bus cycle by driving out the address and status and asserting ADS#. The bus cycle then continues as usual.

Asserting BOFF# during a burst, BS8# or BS16# cycle will force the 486 microprocessor to ignore data returned for that cycle only. Data from previous cycles will still be valid. For example, if BOFF# is asserted on the third BRDY# of a burst, the 486 microprocessor assumes the data returned with the first and second BRDY#'s is correct and restarts the burst beginning with the third item. The same rule applies to transfers broken into multiple cycle by BS8# or BS16#.

Asserting BOFF# in the same clock as ADS# will cause the 486 microprocessor to float its bus in the next clock and leave ADS# floating low. Since ADS# is floating low, a peripheral may think that a new bus cycle has begun even-though the cycle was aborted. There are two possible solutions to this problem. The first is to have all devices recognize this condition and ignore ADS# until ready comes back. The second approach is to use a "two clock" backoff: in the first clock AHOLD is asserted, and in the second clock BOFF# is asserted. This guarantees that ADS# will not be floating low. This is only necessary in systems where BOFF# may be asserted in the same clock as ADS#.

7.2.13 BUS STATES

A bus state diagram is shown in Figure 7.30. A description of the signals used in the diagram is given in Table 7.9.

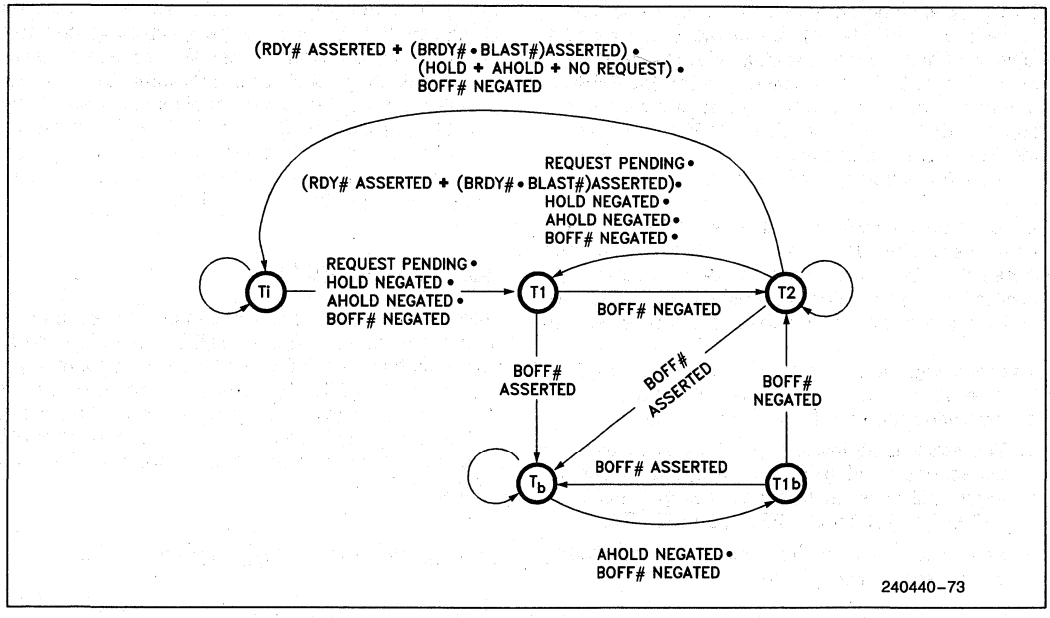


Figure 7.30. Bus State Diagram

Table 7.9. Bus State Description

State	Means
Ti	Bus is idle. Address and status signals may be driven to undefined values, or the bus may be floated to a high impedance state.
T1	First clock cycle of a bus cycle. Valid address and status are driven and ADS# is asserted.
T2	Second and subsequent clock cycles of a bus cycle. Data is driven if the cycle is a write, or data is expected if the cycle is a read. RDY# and BRDY# are sampled.
T1b	First clock cycle of a restarted bus cycle. Valid address and status are driven and ADS# is asserted.
Tb	Second and subsequent clock cycles of an aborted bus cycle.

7.2.14 FLOATING POINT ERROR HANDLING

The 486 microprocessor provides two options for reporting floating point errors. The simplest method is to raise interrupt 16 whenever an unmasked floating point error occurs. This option may be enabled by setting the NE bit in control register 0 (CR0).

The 486 microprocessor also provides the option of allowing external hardware to determine how floating point errors are reported. This option is necessary for compatibility with the error reporting scheme used in DOS based systems. The NE bit must be cleared in CR0 to enable user-defined error reporting. User-defined error reporting is the default condition because the NE bit is cleared on reset.

Two pins, floating point error (FERR#) and ignore numeric error (IGNNE#), are provided to direct the actions of hardware if user-defined error reporting is used. The 486 microprocessor asserts the FERR# output to indicate that a floating point error has occurred. FERR# corresponds to the ERROR# pin on the 387 math coprocessor. However, there is a difference in the behavior of the two.

In some cases FERR# is asserted when the next floating point instruction is encountered and in other cases it is asserted before the next floating point instruction is encountered depending upon the execution state of the instruction causing the exception.

The following class of floating point exceptions drive FERR# at the time the exception occurs (i.e., before encountering the next floating point instruction).

1. The stack fault, invalid operation, and denormal exceptions on all transcendental instructions, integer arithmetic instructions, FSQRT, FSEALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exceptions on store instructions (including integer store instructions).

The following class of floating point exceptions drive FERR# only after encountering the next floating point instruction.

1. Exceptions other than on all transcendental instructions, integer arithmetic instructions, FSQRT, FSCALE, FPREM(1), FXTRACT, FBLD, and FBSTP.
2. Any exception on all basic arithmetic, load, compare, and control instructions (i.e., all other instructions).

For both sets of exceptions above, the 387 Math Coprocessor asserts ERROR# when the error occurs and does not wait for the next floating point instruction to be encountered.

IGNNE# is an input to the 486 microprocessor. When the NE bit in CR0 is cleared, and IGNNE# is asserted, the 486 microprocessor will ignore a user floating point error and continue executing floating point instructions. When IGNNE# is negated, the 486 microprocessor will freeze on floating point instructions which get errors (except for the control instructions FNCLEX, FNINIT, FNSAVE, FNSTENV, FNSTCW, FNSTSW, FNSTSW AX, FNENI, FNDISI and FNSETPM). IGNNE# may be asynchronous to the 486 clock.

In systems with user-defined error reporting, the FERR# pin is connected to the interrupt controller. When an unmasked floating point error occurs, an interrupt is raised. If IGNNE# is high at the time of this interrupt, the 486 microprocessor will freeze (disallowing execution of a subsequent floating point instruction) until the interrupt handler is invoked. By driving the IGNNE# pin low (when clearing the interrupt request), the interrupt handler can allow execution of a floating point instruction, within the interrupt handler, before the error condition is cleared (by FNCLEX, FNINIT, FNSAVE or FNSTENV). If execution of a non-control floating point instruction, within the floating point interrupt handler, is not needed, the IGNNE# pin can be tied HIGH.

8.0 TESTABILITY

Testing in the 486 microprocessor can be divided into two categories: Built-in Self Test (BIST) and external testing. The BIST tests the non-random logic, control ROM (CROM), translation lookaside buffer (TLB) and on-chip cache memory. External tests can be run on the TLB and the on-chip cache. The 486 microprocessor also has a test mode in which all outputs are tristated.

8.1 Built-In Self Test (BIST)

The BIST is initiated by holding the AHOLD (address hold) pin HIGH for 2 CLKs before and 2 CLKs after RESET going from HIGH to LOW as shown in Figure 6.3. The BIST takes approximately 2**20 clocks, or approximately 42 milliseconds with a 25 MHz 486 microprocessor. No bus cycles will be run by the 486 microprocessor until the BIST is concluded. Note that for i486 the RESET must be active for 15 clocks with or without BIST being enabled for warm resets.

The results of BIST is stored in the EAX register. The 486 microprocessor has successfully passed the BIST if the contents of the EAX register are zero. If the results in EAX are not zero then the BIST has detected a flaw in the microprocessor. The microprocessor performs reset and begins normal operation at the completion of the BIST.

The non-random logic, control ROM, on-chip cache and translation lookaside buffer (TLB) are tested during the BIST.

The cache portion of the BIST verifies that the cache is functional and that it is possible to read and write to the cache. The BIST manipulates test registers TR3, TR4 and TR5 while testing the cache. These test registers are described in Section 8.2.

The cache testing algorithm writes a value to each cache entry, reads the value back, and checks that the correct value was read back. The algorithm may be repeated more than once for each of the 512 cache entries using different constants.

The TLB portion of the BIST verifies that the TLB is functional and that it is possible to read and write to the TLB. The BIST manipulates test registers TR6 and TR7 while testing the TLB. TR6 and TR7 are described in Section 8.3.

8.2 On-Chip Cache Testing

The on-chip cache testability hooks are designed to be accessible during the BIST and for assembly language testing of the cache.

The 486 microprocessor contains a cache fill buffer and a cache read buffer. For testability writes, data must be written to the cache fill buffer before it can be written to a location in the cache. Data must be read from a cache location into the cache read buffer before the microprocessor can access the data. The cache fill and cache read buffer are both 128 bits wide.

8.2.1 CACHE TESTING REGISTERS TR3, TR4 AND TR5

Figure 8.1 shows the three cache testing registers: the Cache Data Test Register (TR3), the Cache Status Test Register (TR4) and the Cache Control Test Register (TR5). External access to these registers is provided through MOV reg,TREG and MOV TREG, reg instructions.

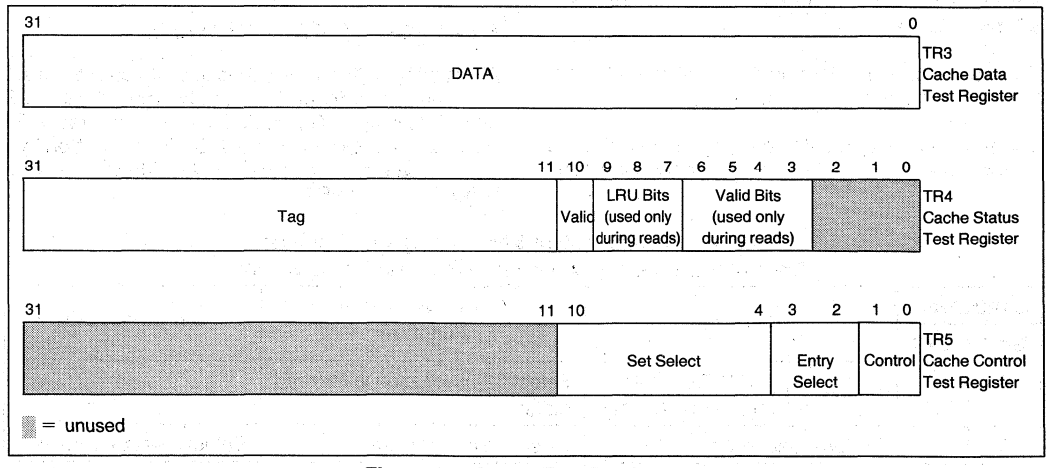


Figure 8.1. Cache Test Registers

Cache Data Test Register: TR3

The cache fill buffer and the cache read buffer can only be accessed through TR3. Data to be written to the cache fill buffer must first be written to TR3. Data read from the cache read buffer must be loaded into TR3.

TR3 is 32 bits wide while the cache fill and read buffers are 128 bits wide. 32 bits of data must be written to TR3 four times to fill the cache fill buffer. 32 bits of data must be read from TR3 four times to empty the cache read buffer. The entry select bits in TR5 determine which 32 bits of data TR3 will access in the buffers.

Cache Status Test Register: TR4

TR4 handles tag, LRU and valid bit information during cache tests. TR4 must be loaded with a tag and a valid bit before a write to the cache. After a read from a cache entry, TR4 contains the tag and valid bit from that entry, and the LRU bits and four valid bits from the accessed set.

Cache Control Test Register: TR5

TR5 specifies which testability operation will be performed and the set and entry within the set which will be accessed.

The seven bit set select field determines which of the 128 sets will be accessed.

The functionality of the two entry select bits depend on the state of the control bits. When the fill or read buffers are being accessed, the entry select bits point to the 32-bit location in the buffer being accessed. When a cache location is specified, the entry select bits point to one of the four entries in a set. Refer to Table 8.1.

Five testability functions can be performed on the cache. The two control bits in TR5 specify the operation to be executed. The five operations are:

1. Write cache fill buffer
2. Perform a cache testability write
3. Perform a cache testability read
4. Read the cache read buffer
5. Perform a cache flush

Table 8.1 shows the encoding of the two control bits in TR5 for the cache testability functions. Table 8.1 also shows the functionality of the entry and set select bits for each control operation.

The cache tests attempt to use as much of the normal operating circuitry as possible. Therefore when cache tests are being performed, the cache must be disabled (the CD and NW bits in control register must be set to 1 to disable the cache. See Section 5).

8.2.2 CACHE TESTABILITY WRITE

A testability write to the cache is a two step process. First the cache fill buffer must be loaded with 128 bits of data and TR4 loaded with the tag and valid bit. Next the contents of the fill buffer are written to a cache location. Sample assembly code to do a write is given in Figure 8.2.

Table 8.1. Cache Control Bit Encoding and Effect of Control Bits on Entry Select and Set Select Functionality

Control Bits		Operation	Entry Select Bits Function	Set Select Bits
Bit 1	Bit 0			
0	0	Enable { Fill Buffer Write / Read Buffer Read	Select 32-bit location in fill/read buffer	—
0	1	Perform Cache Write	Select an entry in set.	Select a set to write to
1	0	Perform Cache Read	Select an entry in set.	Select a set to read from
1	1	Perform Flush Cache	—	—

Sample Assembly Code

An example assembly language sequence to perform a cache write is:

```
;
; eax. ebx. ecx. edx contain the cache line to write
; edi contains the tag information to load
; CR0 already says to enable reads/write to TR5
;
; fill the cache buffer
        mov esi,0         ; set up command
        mov tr5,esi       ; load to TR5
        mov tr3,eax       ; load data into cache fill buffer
        mov esi,4
        mov tr5,esi
        mov tr3,ebx
        mov esi,8
        mov tr5,esi
        mov tr3,ecx
        mov esi,0ch
        mov tr5,esi
        mov tr3,edx
;
; load the Cache Status Register
;
        mov tr4,edi       ; load 21-bit tag and valid bit
;
; perform the cache write
;
        mov esi,1
        mov tr5,esi       ; write the cache (set 0, entry 0)
```

An example assembly language sequence to perform a cache read is:

```
;
; data into eax, ebx, ecx, edx; status into edi
;
; read the cache line back
;
        mov esi,2
        mov tr5,esi       ; do cache testability read (set 0, entry 0)
;
; read the data from the read buffer
;
        mov esi,0
        mov tr5,esi
        mov eax,tr3
        mov esi,4
        mov tr5,esi
        mov ebx,tr3
        mov esi,8
        mov tr5,esi
        mov ecx,tr3
        mov esi,0ch
        mov tr5,esi
        mov edx,tr3
;
; read the status from TR4
;
        mov edi,tr4
```

Figure 8.2 Sample Assembly Code for Cache Testing

Loading the fill buffer is accomplished by first writing to the entry select bits in TR5 and setting the control bits in TR5 to 00. The entry select bits identify one of four 32-bit locations in the cache fill buffer to put 32 bits of data. Following the write to TR5, TR3 is written with 32 bits of data which are immediately placed in the cache fill buffer. Writing to TR3 initiates the write to the cache fill buffer. The cache fill buffer is loaded with 128 bits of data by writing to TR5 and TR3 four times using a different entry select location each time.

TR4 must be loaded with the 21-bit tag and valid bit (bit 10 in TR4) before the contents of the fill buffer are written to a cache location.

The contents of the cache fill buffer are written to a cache location by writing TR5 with a control field of 01 along with the set select and entry select fields. The set select and entry select field indicate the location in the cache to be written. The normal cache LRU update circuitry updates the internal LRU bits for the selected set.

Note that a cache testability write can only be done when the cache is disabled for replaces (the CD bit is control register 0 is reset to 1). Also note that care must be taken when directly writing to entries in the cache. If the entry is set to overlap an area of memory that is being used in external memory, that cache entry could inadvertently be used instead of the external memory. Of course, this is exactly the type of operation that one would desire if the cache were to be used as a high speed RAM.

8.2.3 CACHE TESTABILITY READ

A cache testability read is a two step process. First the contents of the cache location are read into the cache read buffer. Next the data is examined by reading it out of the read buffer. Sample assembly code to do a testability read is given in Figure 8.2.

Reading the contents of a cache location into the cache read buffer is initiated by writing TR5 with the control bits set to 10 and the desired seven-bit set select and two-bit entry select. In response to the write to TR5, TR4 is loaded with the 21-bit tag field and the single valid bit from the cache entry read. TR4 is also loaded with the three LRU bits and four valid bits corresponding to the cache set that was accessed. The cache read buffer is filled with the 128-bit value which was found in the data array at the specified location.

The contents of the read buffer are examined by performing four reads of TR3. Before reading TR3 the entry select bits in TR5 must loaded to indicate which of the four 32-bit words in the read buffer to transfer into TR3 and the control bits in TR5 must be loaded with 00. The register read of TR3 will initiate the transfer of the 32-bit value from the read buffer to the specified general purpose register.

Note that it is very important that the entire 128-bit quantity from the read buffer and also the information from TR4 be read before any memory references are allowed to occur. If memory operations are allowed to happen, the contents of the read buffer will be corrupted. This is because the testability operations use hardware that is used in normal memory accesses for the 486 microprocessor whether the cache is enabled or not.

8.2.4 FLUSH CACHE

The control bits in TR5 must be written with 11 to flush the cache. None of the other bits in TR5 have any meaning when 11 is written to the control bits. Flushing the cache will reset the LRU bits and the valid bits to 0, but will not change the cache tag or data arrays.

When the cache is flushed by writing to TR5 the special bus cycle indicating a cache flush to the external system is not run (see Section 7.2.11, Special Bus Cycles). The cache should be flushed with the instruction INVD (Invalidate Data Cache) instruction or the WBINVD (Write-back and Invalidate Data Cache) instruction.

8.3 Translation Lookaside Buffer (TLB) Testing

The 486 microprocessor TLB testability hooks are similar to those in the 386 microprocessor. The testability hooks have been enhanced to provide added test features and to include new features in the 486 microprocessor. The TLB testability hooks are designed to be accessible during the BIST and for assembly language testing of the TLB.

8.3.1 TRANSLATION LOOKASIDE BUFFER ORGANIZATION

The 486 microprocessors TLB is 4-way set associative and has space for 32 entries. The TLB is logically split into three blocks shown in Figure 8.3.

The data block is physically split into four arrays, each with space for eight entries. An entry in the data block is 22 bits wide containing a 20-bit physical address and two bits for the page attributes. The page attributes are the PCD (page cache disable) bit and the PWT (page write-through) bit. Refer to Section 4.5.4 for a discussion of the PCD and PWT bits.

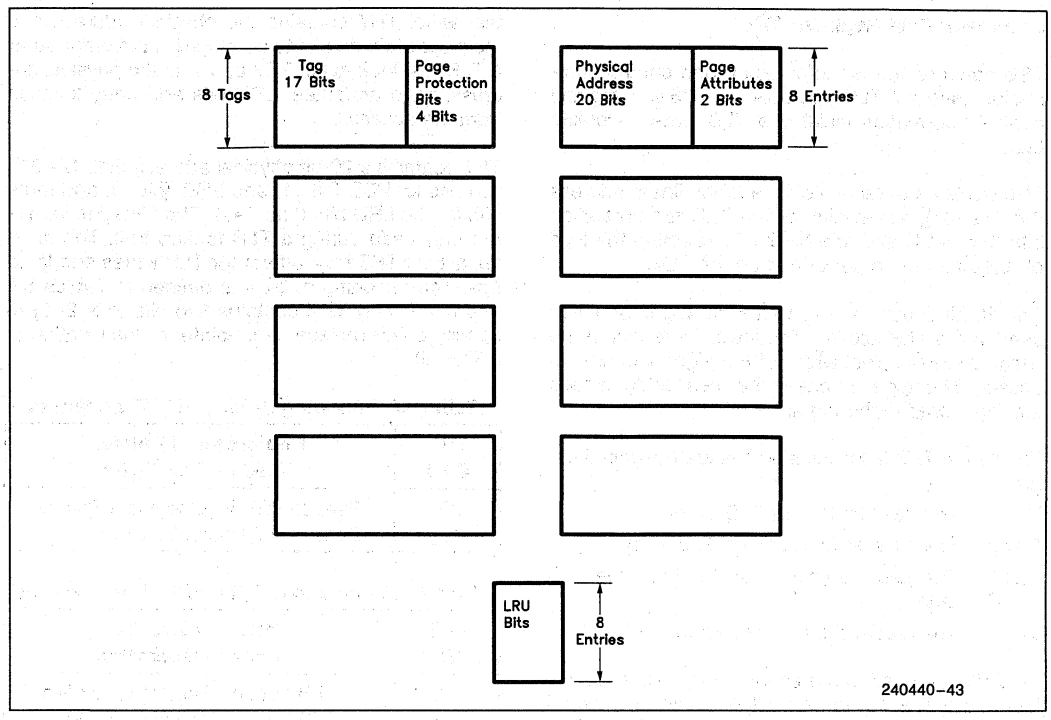


Figure 8.3. TLB Organization

The tag block is also split into four arrays, one for each of the data arrays. A tag entry is 21 bits wide containing a 17-bit linear address and four protection bits. The protection bits are valid (V), user/supervisor (U/S), read/write (R/W) and dirty (D).

The third block contains eight three bit quantities used in the pseudo least recently used (LRU) replacement algorithm. These bits are called the LRU bits. The LRU replacement algorithm used in the TLB is the same as used by the on-chip cache. For a description of this algorithm refer to Section 5.5.

8.3.2 TLB TEST REGISTERS TR6 AND TR7

The two TLB test registers are shown in Figure 8.4. TR6 is the command test register and TR7 is the data test register. External access to these registers is provided through MOV reg,TREG and MOV TREG,reg instructions.

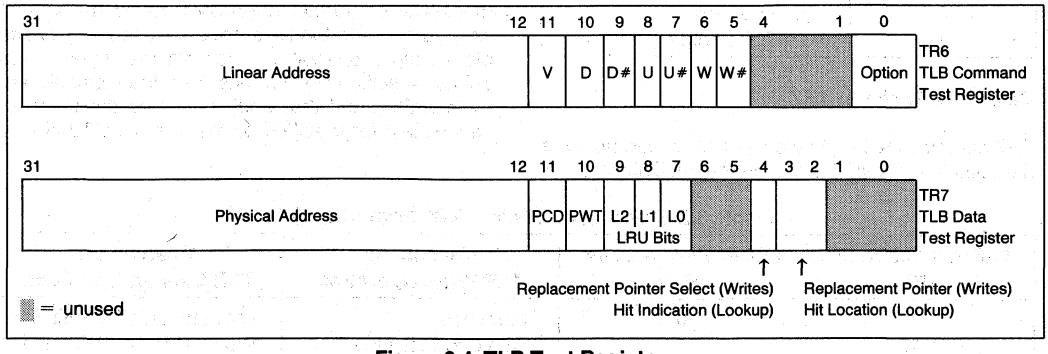


Figure 8.4. TLB Test Registers

Command Test Register: TR6

TR6 contains the tag information and control information used in a TLB test. Loading TR6 with tag and control information initiates a TLB write or lookup test.

TR6 contains three bit fields, a 20-bit linear address (bits 12–31), seven bits for the TLB tag protection bits (bits 5-11) and one bit (bit 0) to define the type of operation to be performed on the TLB.

The 20-bit linear address forms the tag information used in the TLB access. The lower three bits of the linear address select which of the eight sets are accessed. The upper 17 bits of the linear address form the tag stored in the tag array.

The seven TLB tag protection bits are described below.

V: The valid bit for this TLB entry

D,D#: The dirty bit for/from the TLB entry

U,U#: The user/supervisor bit for/from the TLB entry

W,W#: The read/write bit for/from the TLB entry

Two bits are used to represent the D, U/S and R/W bits in the TLB tag to permit the option of a forced miss or hit during a TLB lookup operation. The forced miss or hit will occur regardless of the state of the actual bit in the TLB. The meaning of these pairs of bits is given in Table 8.2.

The operation bit in TR6 determines if the TLB test operation will be a write or a lookup. The function of the operation bit is given in Table 8.3.

Table 8.3. TR6 Operation Bit Encoding

TR6 Bit 0	TLB Operation to Be Performed
0	TLB Write
1	TLB Lookup

Data Test Register: TR7

TR7 contains the information stored or read from the data block during a TLB test operation. Before a TLB test write, TR7 contains the physical address and the page attribute bits to be stored in the entry. After a TLB test lookup hit, TR7 contains the physical address, page attributes, LRU bits and entry location from the access.

TR7 contains a 20-bit physical address (bits 12–31), two bits for PCD (bit 11) and PWT (bit 10) and three bits for the LRU bits (bits 7–9). The LRU bits in TR7 are only used during a TLB lookup test. The functionality of TR7 bit 4 differs for TLB writes and lookups. The encoding of bit 4 is defined in Tables 8.4 and 8.5. Finally TR7 contains two bits (bits 2–3) to specify a TLB replacement pointer or the location of a TLB hit.

Table 8.4. Encoding of Bit 4 of TR7 on Writes

TR7 Bit 4	Replacement Pointer Used on TLB Write
0	Pseudo-LRU Replacement Pointer
1	Data Test Register Bits 3:2

Table 8.5. Encoding of Bit 4 of TR7 on Lookups

TR7 Bit 4	Meaning after TLB Lookup Operation
0	TLB Lookup Resulted in a Miss
1	TLB Lookup Resulted in a Hit

A replacement pointer is used during a TLB write. The pointer indicates which of the four entries in an accessed set is to be written. The replacement pointer can be specified to be the internal LRU bits or bits 2–3 in TR7. The source of the replacement pointer is specified by TR7 bit 4. The encoding of bit 4 during a write is given by Table 8.4.

Note that both testability writes and lookups affect the state of the internal LRU bits regardless of the replacement pointer used. All TLB write operations (testability or normal operation) cause the written entry to become the most recently used. For example, during a testability write with the replacement pointer specified by TR7 bits 2–3, the indicated entry is written and that entry becomes the most recently used as specified by the internal LRU bits.

Table 8.2. Meaning of a Pair of TR6 Protection Bits

TR6 Protection Bit (B)	TR6 Protection Bit# (B#)	Meaning on TLB Write Operation	Meaning on TLB Lookup Operation
0	0	Undefined	Miss any TLB TAG Bit B
0	1	Write 0 to TLB TAG Bit B	Match TLB TAG Bit B if 0
1	0	Write 1 to TLB TAG Bit B	Match TLB TAG Bit B if 1
1	1	Undefined	Match any TLB TAG Bit B

There are two TLB testing operations: write entries into the TLB, and perform TLB lookups. One major enhancement over TLB testing in the 386 microprocessor is that paging need not be disabled while executing testability writes or lookups.

Note that any time one TLB set contains the same linear address in more than one of its entries, looking up that linear address will not result in a hit. Therefore a single linear address should not be written to one TLB set more than once.

8.3.3 TLB WRITE TEST

To perform a TLB write TR7 must be loaded followed by a TR6 load. The register operations must be performed in this order since the TLB operation is triggered by the write to TR6.

TR7 is loaded with a 20-bit physical address and values for PCD and PWT to be written to the data portion of the TLB. In addition, bit 4 of TR7 must be loaded to indicate whether to use TR7 bits 3-2 or the internal LRU bits as the replacement pointer on the TLB write operation. Note that the LRU bits in TR7 are not used in a write test.

TR6 must be written to initiate the TLB write operation. Bit 0 in TR6 must be reset to zero to indicate a TLB write. The 20-bit linear address and the seven page protection bits must also be written in TR6 to specify the tag portion of the TLB entry. Note that the three least significant bits of the linear address specify which of the eight sets in the data block will be loaded with the physical address data. Thus only 17 of the linear address bits are stored in the tag array.

8.3.4 TLB LOOKUP TEST

To perform a TLB lookup it is only necessary to write the proper tags and control information into TR6. Bit 0 in TR6 must be set to 1 to indicate a TLB lookup. TR6 must be loaded with a 20-bit linear address and the seven protection bits. To force misses and matches of the individual protection bits on TLB lookups, set the seven protection bits as specified in Table 8.2.

A TLB lookup operation is initiated by the write to TR6. TR7 will indicate the result of the lookup operation following the write to TR6. The hit/miss indication can be found in TR7 bit 4 (see Table 8.5).

TR7 will contain the following information if bit 4 indicated that the lookup test resulted in a hit. Bits 2–3 will indicate in which set the match occurred. The 22 most significant bits in TR7 will contain the physical address and page attributes contained in the entry. Bits 9–7 will contain the LRU bits associated with the accessed set. The state of the LRU bits is previous to their being updated for the current lookup.

If bit 4 in TR7 indicated that the lookup test resulted in a miss the remaining bits in TR7 are undefined.

Again it should be noted that a TLB testability lookup operation affects the state of the LRU bits. The LRU bits will be updated if a hit occurred. The entry which was hit will become the most recently used.

8.4 Tristate Output Test Mode

The 486 microprocessor provides the ability to float all its outputs and bidirectional pins. This includes all pins floated during bus hold as well as pins which are never floated in normal operation of the chip (HLDA, BREQ, FERR# and PCHK#). When the 486 microprocessor is in the tri-state output test mode external testing can be used to test board connections.

The tri-state test mode is invoked by driving FLUSH# low for 2 clocks before and 2 clocks after RESET going low. The outputs are guaranteed to tristate no later than 10 clocks after RESET goes low (see Figure 6.4). The 486 microprocessor remains in the tristate test mode until the next RESET.

9.0 DEBUGGING SUPPORT

The 486 Microprocessor provides several features which simplify the debugging process. The three categories of on-chip debugging aids are:

1) the code execution breakpoint opcode (0CCH),
2) the single-step capability provided by the TF bit in the flag register, and
3) the code and data breakpoint capability provided by the Debug Registers DR0–3, DR6, and DR7.

9.1 Breakpoint Instruction

A single-byte-opcode breakpoint instruction is available for use by software debuggers. The breakpoint opcode is 0CCH, and generates an exception 3 trap when executed. In typical use, a debugger program can "plant" the breakpoint instruction at all desired code execution breakpoints. The single-byte breakpoint opcode is an alias for the two-byte general software interrupt instruction, INT n, where n=3. The only difference between INT 3 (0CCh) and INT n is that INT 3 is never IOPL-sensitive but INT n is IOPL-sensitive in Protected Mode and Virtual 8086 Mode.

9.2 Single-Step Trap

If the single-step flag (TF, bit 8) in the EFLAG register is found to be set at the end of an instruction, a single-step exception occurs. The single-step exception is auto vectored to exception number 1. Precisely, exception 1 occurs as a trap after the instruction following the instruction which set TF. In typical practice, a debugger sets the TF bit of a flag register image on the debugger's stack. It then typically transfers control to the user program and loads the flag image with a signal instruction, the IRET instruction. The single-step trap occurs after executing one instruction of the user program.

Since the exception 1 occurs as a trap (that is, it occurs after the instruction has already executed), the CS:EIP pushed onto the debugger's stack points to the next unexecuted instruction of the program being debugged. An exception 1 handler, merely by ending with an IRET instruction, can therefore efficiently support single-stepping through a user program.

9.3 Debug Registers

The Debug Registers are an advanced debugging feature of the 486 Microprocessor. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution breakpoint can be placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT3 breakpoint opcode.

The 486 Microprocessor contains six Debug Registers, providing the ability to specify up to four distinct breakpoints addresses, breakpoint control options, and read breakpoint status. Initially after reset, breakpoints are in the disabled state. Therefore, no breakpoints will occur unless the debug registers are programmed. Breakpoints set up in the Debug Registers are autovectored to exception number 1.

9.3.1 LINEAR ADDRESS BREAKPOINT REGISTERS (DR0–DR3)

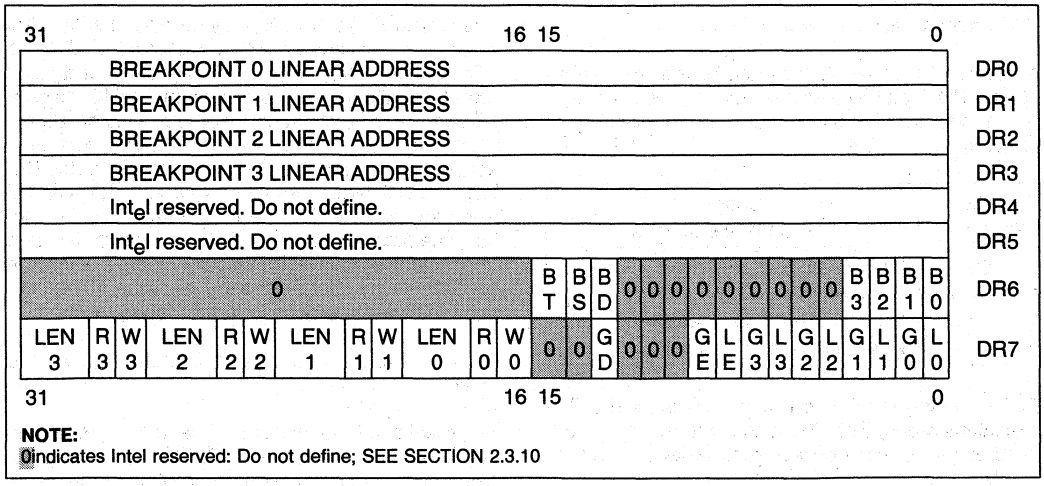
Up to four breakpoint addresses can be specified by writing into Debug Registers DR0–DR3, shown in Figure 9.1. The breakpoint addresses specified are 32-bit linear addresses. 486 Microprocessor hardware continuously compares the linear breakpoint addresses in DR0–DR3 with the linear addresses generated by executing software (a linear address is the result of computing the effective address and adding the 32-bit segment base address). Note that if paging is not enabled the linear address equals the physical address. If paging is enabled, the linear address is translated to a physical 32-bit address by the on-chip paging unit. Regardless of whether paging is enabled or not, however, the breakpoint registers hold linear addresses.

9.3.2 DEBUG CONTROL REGISTER (DR7)

A Debug Control Register, DR7 shown in Figure 9.1, allows several debug control functions such as enabling the breakpoints and setting up other control options for the breakpoints. The fields within the Debug Control Register, DR7, are as follows:

LENi (breakpoint length specification bits)

A 2-bit LEN field exists for each of the four breakpoints. LEN specifies the length of the associated breakpoint field. The choices for data breakpoints are: 1 byte, 2 bytes, and 4 bytes. Instruction execution breakpoints must have a length of 1 (LENi = 00). Encoding of the LENi field is as follows:



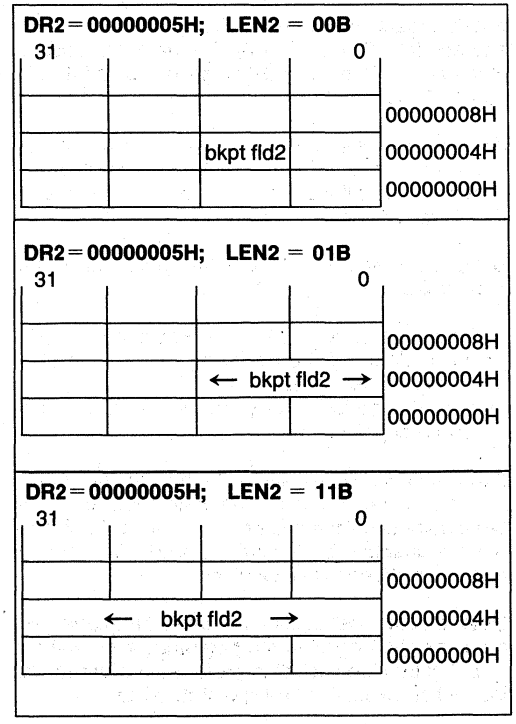
NOTE:
0indicates Intel reserved: Do not define; SEE SECTION 2.3.10

Figure 9.1. Debug Registers

LENi Encoding	Breakpoint Field Width	Usage of Least Significant Bits in Breakpoint Address Register i, (i=0–3)
00	1 byte	All 32-bits used to specify a single-byte breakpoint field.
01	2 bytes	A1–A31 used to specify a two-byte, word-aligned breakpoint field. A0 in Breakpoint Address Register is not used.
10	Undefined—do not use this encoding	
11	4 bytes	A2–A31 used to specify a four-byte, dword-aligned breakpoint field. A0 and A1 in Breakpoint Address Register are not used.

The LENi field controls the size of breakpoint field i by controlling whether all low-order linear address bits in the breakpoint address register are used to detect the breakpoint event. Therefore, all breakpoint fields are aligned; 2-byte breakpoint fields begin on Word boundaries, and 4-byte breakpoint fields begin on Dword boundaries.

The following is an example of various size breakpoint fields. Assume the breakpoint linear address in DR2 is 00000005H. In that situation, the following illustration indicates the region of the breakpoint field for lengths of 1, 2, or 4 bytes.



5

RWi (memory access qualifier bits)

A 2-bit RW field exists for each of the four breakpoints. The 2-bit RW field specifies the type of usage which must occur in order to activate the associated breakpoint.

RW Encoding	Usage Causing Breakpoint
00	Instruction execution only
01	Data writes only
10	Undefined—do not use this encoding
11	Data reads and writes only

RW encoding 00 is used to set up an instruction execution breakpoint. RW encodings 01 or 11 are used to set up write-only or read/write data breakpoints.

Note that **instruction execution breakpoints are taken as faults** (i.e., before the instruction executes), but **data breakpoints are taken as traps** (i.e., after the data transfer takes place).

Using LENi and RWi to Set Data Breakpoint i

A data breakpoint can be set up by writing the linear address into DRi (i = 0–3). For data breakpoints, RWi can = 01 (write-only) or 11 (write/read). LEN can = 00, 01, or 11.

If a data access entirely or partly falls within the data breakpoint field, the data breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 trap will occur.

Using LENi and RWi to Set Instruction Execution Breakpoint i

An instruction execution breakpoint can be set up by writing address of the beginning of the instruction (including prefixes if any) into DRi (i = 0–3). RWi must = 00 and LEN must = 00 for instruction execution breakpoints.

If the instruction beginning at the breakpoint address is about to be executed, the instruction execution breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 fault will occur before the instruction is executed.

Note that an instruction execution breakpoint address must be equal to the **beginning** byte address of an instruction (including prefixes) in order for the instruction execution breakpoint to occur.

GD (Global Debug Register access detect)

The Debug Registers can only be accessed in Real Mode or at privilege level 0 in Protected Mode. The GD bit, when set, provides extra protection against **any** Debug Register access even in Real Mode or at privilege level 0 in Protected Mode. This additional protection feature is provided to guarantee that a software debugger can have full control over the Debug Register resources when required. The GD bit, when set, causes an exception 1 fault if an instruction attempts to read or write any Debug Register. The GD bit is then automatically cleared when the exception 1 handler is invoked, allowing the exception 1 handler free access to the debug registers.

GE and LE (Exact data breakpoint match, global and local)

The breakpoint mechanism of the 486 Microprocessor differs from that of the 386. The 486 Microprocessor always does exact data breakpoint matching, regardless of GE/LE bit settings. Any data breakpoint trap will be reported exactly after completion of the instruction that caused the operand transfer. Exact reporting is provided by forcing the 486 Microprocessor execution unit to wait for completion of data operand transfers before beginning execution of the next instruction.

When the 486 Microprocessor performs a task switch, the LE bit is cleared. Thus, the LE bit supports fast task switching out of tasks, that have enabled the exact data breakpoint match for their task-local breakpoints. The LE bit is cleared by the processor during a task switch, to avoid having exact data breakpoint match enabled in the new task. Note that exact data breakpoint match must be re-enabled under software control.

The 486 Microprocessor GE bit is unaffected during a task switch. The GE bit supports exact data breakpoint match that is to remain enabled during all tasks executing in the system.

Note that **instruction execution** breakpoints are always reported exactly.

Gi and Li (breakpoint enable, global and local)

If either Gi or Li is set then the associated breakpoint (as defined by the linear address in DRi, the length in LENi and the usage criteria in RWi) is enabled. If either Gi or Li is set, and the 486 Microprocessor detects the ith breakpoint condition, then the exception 1 handler is invoked.

When the 486 Microprocessor performs a task switch to a new Task State Segment (TSS), all Li bits are cleared. Thus, the Li bits support fast task switching out of tasks that use some task-local

breakpoint registers. The Li bits are cleared by the processor during a task switch, to avoid spurious exceptions in the new task. Note that the breakpoints must be re-enabled under software control.

All 486 Microprocessor Gi bits are unaffected during a task switch. The Gi bits support breakpoints that are active in all tasks executing in the system.

9.3.3 DEBUG STATUS REGISTER (DR6)

A Debug Status Register, DR6 shown in Figure 9.1, allows the exception 1 handler to easily determine why it was invoked. Note the exception 1 handler can be invoked as a result of one of several events:

1) DR0 Breakpoint fault/trap.
2) DR1 Breakpoint fault/trap.
3) DR2 Breakpoint fault/trap.
4) DR3 Breakpoint fault/trap.
5) Single-step (TF) trap.
6) Task switch trap.
7) Fault due to attempted debug register access when GD=1.

The Debug Status Register contains single-bit flags for each of the possible events invoking exception 1. Note below that some of these events are faults (exception taken before the instruction is executed), while other events are traps (exception taken after the debug events occurred).

The flags in DR6 are set by the hardware but never cleared by hardware. Exception 1 handler software should clear DR6 before returning to the user program to avoid future confusion in identifying the source of exception 1.

The fields within the Debug Status Register, DR6, are as follows:

Bi (debug fault/trap due to breakpoint 0–3)

Four breakpoint indicator flags, B0–B3, correspond one-to-one with the breakpoint registers in DR0–DR3. A flag Bi is set when the condition described by DRi, LENi, and RWi occurs.

If Gi or Li is set, and if the ith breakpoint is detected, the processor will invoke the exception 1 handler. The exception is handled as a fault if an instruction execution breakpoint occurred, or as a trap if a data breakpoint occurred.

IMPORTANT NOTE: A flag Bi is set whenever the hardware detects a match condition on **enabled** breakpoint i. Whenever a match is detected on at least one **enabled** breakpoint i, the hardware immediately sets all Bi bits corresponding to breakpoint conditions matching at that instant, whether enabled **or not.** Therefore, the exception 1 handler may see that multiple Bi bits are set, but only set Bi bits corresponding to **enabled** breakpoints (Li or Gi set) are **true** indications of why the exception 1 handler was invoked.

BD (debug fault due to attempted register access when GD bit set)

This bit is set if the exception 1 handler was invoked due to an instruction attempting to read or write to the debug registers when GD bit was set. If such an event occurs, then the GD bit is automatically cleared when the exception 1 handler is invoked, allowing handler access to the debug registers.

BS (debug trap due to single-step)

This bit is set if the exception 1 handler was invoked due to the TF bit in the flag register being set (for single-stepping).

BT (debug trap due to task switch)

This bit is set if the exception 1 handler was invoked due to a task switch occurring to a task having a 486 Microprocessor TSS with the T bit set. Note the task switch into the new task occurs normally, but before the first instruction of the task is executed, the exception 1 handler is invoked. With respect to the task switch operation, the operation is considered to be a trap.

9.3.4 USE OF RESUME FLAG (RF) IN FLAG REGISTER

The Resume Flag (RF) in the flag word can suppress an instruction execution breakpoint when the exception 1 handler returns to a user program at a user address which is also an instruction execution breakpoint.

10.0 INSTRUCTION SET SUMMARY

This section describes the 486 microprocessor instruction set. Tables 10.1 through 10.3 list all instructions along with instruction encoding diagrams and clock counts. Further details of the instruction encoding are then provided in Section 10.2, which completely describes the encoding structure and the definition of all fields occurring within the 486 microprocessor instructions.

10.1 i486™ Microprocessor Instruction Encoding and Clock Count Summary

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in Tables 10.1 through 10.3 by the processor clock period (e.g., 40 ns for a 25 MHz 486 microprocessor).

For more detailed information on the encodings of instructions, refer to Section 10.2 Instruction Encodings. Section 10.2 explains the general structure of instruction encodings, and defines exactly the encodings of all fields contained within the instruction.

INSTRUCTION CLOCK COUNT ASSUMPTIONS

The 486 microprocessor instruction clock count tables give clock counts assuming data and instruction accesses hit in the cache. A separate penalty column defines clocks to add if a data access misses in the cache. The combined instruction and data cache hit rate is over 90%.

A cache miss will force the 486 microprocessor to run an external bus cycle. The 486 microprocessor 32-bit burst bus is defined as r−b−w.

Where:

r = The number of clocks in the first cycle of a burst read or the number of clocks per data cycle in a non-burst read.

b = The number of clocks for the second and subsequent cycles in a burst read.

w = The number of clocks for a write.

The fastest bus the 486 microprocessor can support is 2−1−2 assuming 0 wait states. The clock counts in the cache miss penalty column assume a 2−1−2 bus. For slower busses add r−2 clocks to the cache miss penalty for the first dword accessed. Other factors also affect instruction clock counts.

Instruction Clock Count Assumptions

1. The external bus is available for reads or writes at all times. Else add clocks to reads until the bus is available.
2. Accesses are aligned. Add three clocks to each misaligned access.
3. Cache fills complete before subsequent accesses to the same line. If a read misses the cache during a cache fill due to a previous read or pre-fetch, the read must wait for the cache fill to complete. If a read or write accesses a cache line still being filled, it must wait for the fill to complete.
4. If an effective address is calculated, the base register is not the destination register of the preceding instruction. If the base register is the destination register of the preceding instruction add 1 to the clock counts shown. Back-to-back PUSH and POP instructions are not affected by this rule.
5. An effective address calculation uses one base register and does not use an index register. However, if the effective address calculation uses an index register, 1 clock **may** be added to the clock count shown.
6. The target of a jump is in the cache. If not, add r clocks for accessing the destination instruction of a jump. If the destination instruction is not completely contained in the first dword read, add a maximum of 3b clocks. If the destination instruction is not completely contained in the first 16 byte burst, add a maximum of another r+3b clocks.
7. If no write buffer delay, w clocks are added only in the case in which all write buffers are full. Typically, this case rarely occurs.
8. Displacement and immediate not used together. If displacement and immediate used together, 1 clock **may** be added to the clock count shown.
9. No invalidate cycles. Add a delay of 1 clock for each invalidate cycle if the invalidate cycle contends for the internal cache/external bus when the 486 CPU needs to use it.
10. Page translation hits in TLB. A TLB miss will add 13, 21 or 28 clocks to the instruction depending on whether the Accessed and/or Dirty bit in neither, one or both of the page entries needs to be set in memory. This assumes that neither page entry is in the data cache and a page fault does not occur on the address translation.
11. No exceptions are detected during instruction execution. Refer to Interrupt Clock Counts Table for extra clocks if an interrupt is detected.
12. Instructions that read multiple consecutive data items (i.e. task switch, POPA, etc.) and miss the cache are assumed to start the first access on a 16-byte boundary. If not, an extra cache line fill may be necessary which may add up to (r+3b) clocks to the cache miss penalty.

Table 10.1. i486™ Microprocessor Integer Clock Count Summary

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS				
MOV = Move:				
reg1 to reg2	1000100W | 11 reg1 reg2	1		
reg2 to reg1	1000101w | 11 reg1 reg2	1		
memory to reg	1000101w | mod reg r/m	1	2	
reg to memory	1000100w | mod reg r/m	1		
Immediate to reg	1100011w | 11000 reg | immediate data	1		
or	1011w reg | immediate data	1		
Immediate to Memory	1100011w | mod 000 r/m | displacement immediate	1		
Memory to Accumulator	1010000w | full displacement	1	2	
Accumulator to Memory	1010001w | full displacement	1		
MOVSX/MOVZX = Move with Sign/Zero Extension				
reg2 to reg1	00001111 | 1011z11w | 11 reg1 reg2	3		
memory to reg	00001111 | 1011z11w | mod reg r/m	3	2	
z instruction 0 MOVZX 1 MOVSX				
PUSH = Push				
reg	11111111 | 11 110 reg	4		
or	01010 reg	1		
memory	11111111 | mod 110 r/m	4	1	1
immediate	011010s0 | immediate data	1		
PUSHA = Push All	01100000	11		
POP = Pop				
reg	10001111 | 11 000 reg	4	1	
or	01011 reg	1	2	
memory	10001111 | mod 000 r/m	5	2	1
POPA = Pop All	01100001	9	7/15	16/32
XCHG = Exchange				
reg1 with reg2	1000011w | 11 reg1 reg2	3		2
Accumulator with reg	10010 reg	3		2
Memory with reg	1000011w | mod reg r/m	5		2
NOP = No Operation	10010000	1		
LEA = Load EA to Register	10001101 | mod reg r/m			
no index register		1		
with index register		2		

5

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
Instruction / **TTT**: ADD = Add 000; ADC = Add with Carry 010; AND = Logical AND 100; OR = Logical OR 001; SUB = Subtract 101; SBB = Subtract with Borrow 011; XOR = Logical Exclusive OR 110				
reg1 to reg2	00TTT00w | 11 reg1 reg2	1		
reg2 to reg1	00TTT01w | 11 reg1 reg2	1		
memory to register	00TTT01w | mod reg r/m	2	2	
register to memory	00TTT00w | mod reg r/m	3	6/2	U/L
immediate to register	100000sw | 11 TTT reg | immediate register	1		
immediate to accumulator	00TTT10w | immediate data	1		
immediate to memory	100000sw | mod TTT r/m | immediate data	3	6/2	U/L
Instruction / **TTT**: INC = Increment 000; DEC = Decrement 001				
reg	1111111w | 11 TTT reg	1		
or	01TTT reg	1		
memory	1111111w | mod TTT r/m	3	6/2	U/L
Instruction / **TTT**: NOT = Logical Complement 010; NEG = Negate 011				
reg	1111011w | 11 TTT reg	1		
memory	1111011w | mod TTT r/m	3	6/2	U/L
CMP = Compare				
reg1 with reg2	0011100w | 11 reg1 reg2	1		
reg2 with reg1	0011101w | 11 reg1 reg2	1		
memory with register	0011100w | mod reg r/m	2	2	
register with memory	0011101w | mod reg r/m	2	2	
immediate with register	100000sw | 11 111 reg | immediate data	1		
immediate with acc.	0011110w | immediate data	1		
immediate with memory	100000sw | mod 111 r/m | immediate data	2	2	
TEST – Logical Compare				
reg1 and reg2	1000010w | 11 reg1 reg2	1		
memory and register	1000010w | mod reg r/m	2	2	
immediate and register	1111011w | 11 000 reg | immediate data	1		
immediate and acc.	1010100w | immediate data	1		
immediate and memory	1111011w | mod 000 r/m | immediate data	2	2	

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
MUL = Multiply (unsigned)				
acc. with register	1111011w | 11 100 reg			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
acc. with memory	1111011w | mod 100 r/m			
Multiplier-Byte		13/18	1	MN/MX, 3
Word		13/26	1	MN/MX, 3
Dword		13/42	1	MN/MX, 3
IMUL = Integer Multiply (signed)				
acc. with register	1111011w | 11 101 reg			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
acc. with memory	1111011w | mod 101 r/m			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
reg1 with reg2	00001111 | 10101111 | 11 reg1 reg2			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
register with memory	00001111 | 10101111 | mod reg r/m			
Multiplier-Byte		13/18	1	MN/MX, 3
Word		13/26	1	MN/MX, 3
Dword		13/42	1	MN/MX, 3
reg1 with imm. to reg2	011010s1 | 11 reg1 reg2 | immediate data			
Multiplier-Byte		13/18		MN/MX, 3
Word		13/26		MN/MX, 3
Dword		13/42		MN/MX, 3
mem. with imm. to reg.	011010s1 | mod reg r/m | immediate data			
Multiplier-Byte		13/18	2	MN/MX, 3
Word		13/26	2	MN/MX, 3
Dword		13/42	2	MN/MX, 3
DIV = Divide (unsigned)				
acc. by register	1111011w | 11 110 reg			
Divisor-Byte		16		
Word		24		
Dword		40		
acc. by memory	1111011w | mod 110 r/m			
Divisor-Byte		16		
Word		24		
Dword		40		
IDIV = Integer Divide (signed)				
acc. by register	1111011w | 11 111 reg			
Divisor-Byte		19		
Word		27		
Dword		43		

5

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTEGER OPERATIONS (Continued)				
acc. by memory	1111011w | mod 111 r/m			
Divisor-Byte		20		
Word		28		
Dword		44		
CBW = Convert Byte to Word	10011000	3		
CWD = Convert Word to Dword	10011001	3		
Not Through Carry (ROL, ROR, SAL, SAR, SHL, and SHR)				
reg by 1	1101000w | 11 TTT reg	3		
memory by 1	1101000w | mod TTT r/m	4	6	
reg by CL	1101001w | 11 TTT reg	3		
memory by CL	1101001w | mod TTT r/m	4	6	
reg by immediate count	1100000w | 11 TTT reg | immediate 8-bit data	2		
mem by immediate count	1100000w | mod TTT r/m | immediate 8-bit data	4	6	
Through Carry (RCL and RCR)				
reg by 1	1101000w | 11 TTT reg	3		
memory by 1	1101000w | mod TTT r/m	4	6	
reg by CL	1101001w | 11 TTT reg	8/30		MN/MX, 4
memory by CL	1101001w | mod TTT r/m	9/31		MN/MX, 5
reg by immediate count	1100000w | 11 TTT reg | immediate 8-bit data	8/30		MN/MX, 4
mem by immediate count	1100000w | mod TTT r/m | immediate 8-bit data	9/31		MN/MX, 5
register with immediate	00001111 | 10TTT100 | 11 reg2 reg1 | imm 8-bit data	2		
memory by immediate	00001111 | 10TTT100 | mod reg r/m | imm 8-bit data	3	6	
register by CL	00001111 | 10TTT101 | 11 reg2 reg1	3		
memory by CL	00001111 | 10TTT101 | mod reg r/m	4	5	
BSWAP = Byte Swap	00001111 | 11001 reg	1		
XADD = Exchange and Add				
reg1, reg2	00001111 | 1100000w | 11 reg2 reg1	3		
memory, reg	00001111 | 1100000w | mod reg r/m	4	6/2	U/L
CMPXCHG = Compare and Exchange				
reg1, reg2	00001111 | 1011000w | 11 reg2 reg1	6		
memory, reg	00001111 | 1011000w | mod reg r/m	7/10	2	6

Instruction	TTT
ROL = Rotate Left	000
ROR = Rotate Right	001
RCL = Rotate through Carry Left	010
RCR = Rotate through Carry Right	011
SHL/SAL = Shift Logical/Arithmetic Left	100
SHR = Shift Logical Right	101
SAR = Shift Arithmetic Right	111

Instruction	TTT
SHLD = Shift Left Double	100
SHRD = Shift Right Double	101

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
CONTROL TRANSFER (within segment)				
NOTE: Times are jump taken/not taken				
Jccc = Jump on ccc				
8-bit displacement	0111tttn | 8-bit disp.	3/1		T/NT, 23
full displacement	00001111 | 1000tttn | full displacement	3/1		T/NT, 23
NOTE: Times are jump taken/not taken				
SETcccc = Set Byte on cccc (Times are cccc true/false)				
reg	00001111 | 1001tttn | 11 000 reg	4/3		
memory	00001111 | 1001tttn | mod 000 r/m	3/4		
LOOP = LOOP CX Times	11100010 | 8-bit disp.	7/6		L/NL, 23
LOOPZ/LOOPE = Loop with Zero/Equal	11100001 | 8-bit disp.	9/6		L/NL, 23
LOOPNZ/LOOPNE = Loop while Not Zero	11100000 | 8-bit disp.	9/6		L/NL, 23
JCXZ = Jump on CX Zero	11100011 | 8-bit disp.	8/5		T/NT, 23
JECXZ = Jump on ECX Zero	11100011 | 8-bit disp.	8/5		T/NT, 23
(Address Size Prefix Differentiates JCXZ for JECXZ)				
JMP = Unconditional Jump (within segment)				
Short	11101011 | 8-bit disp.	3		7, 23
Direct	11101001 | full displacement	3		7, 23
Register Indirect	11111111 | 11 100 reg	5		7, 23
Memory Indirect	11111111 | mod 100 r/m	5	5	7
CALL = Call (within segment)				
Direct	11101000 | full displacement	3		7, 23
Register Indirect	11111111 | 11 010 reg	5		7, 23
Memory Indirect	11111111 | mod 010 r/m	5	5	7
RET = Return from CALL (within segment)				
	11000011	5	5	
Adding Immediate to SP	11000010 | 16-bit disp.	5	5	

Mnemonic cccc	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less Than or Equal/Greater Than	1111

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
CONTROL TRANSFER (within segment) (Continued)				
ENTER = Enter Procedure	11001000 | 16-bit disp., 8-bit level			
Level = 0		14		
Level = 1		17		
Level (L) > 1		17+3L		8
LEAVE = Leave Procedure	11001001	5	1	
MULTIPLE-SEGMENT INSTRUCTIONS				
MOV = Move				
reg. to segment reg.	10001110 | 11 sreg3 reg	3/9	0/3	RV/P, 9
memory to segment reg.	10001110 | mod sreg3 r/m	3/9	2/5	RV/P, 9
segment reg. to reg.	10001100 | 11 sreg3 reg	3		
segment reg. to memory	10001100 | mod sreg3 r/m	3		
PUSH = Push				
segment reg. (ES, CS, SS, or DS)	000sreg2110	3		
segment reg. (FS or GS)	00001111 | 10 sreg3000	3		
POP = Pop				
segment reg. (ES, SS, or DS)	000sreg2111	3/9	2/5	RV/P, 9
segment reg. (FS or GS)	00001111 | 10 sreg3001	3/9	2/5	RV/P, 9
LDS = Load Pointer to DS	11000101 | mod reg r/m	6/12	7/10	RV/P, 9
LES = Load Pointer to ES	11000100 | mod reg r/m	6/12	7/10	RV/P, 9
LFS = Load Pointer to FS	00001111 | 10110100 | mod reg r/m	6/12	7/10	RV/P, 9
LGS = Load Pointer to GS	00001111 | 10110101 | mod reg r/m	6/12	7/10	RV/P, 9
LSS = Load Pointer to SS	00001111 | 10110010 | mod reg r/m	6/12	7/10	RV/P, 9
CALL = Call				
Direct intersegment	10011010 | unsigned full offset, selector	18	2	R, 7, 22
to same level		20	3	P, 9
thru Gate to same level		35	6	P, 9
to inner level, no parameters		69	17	P, 9
to inner level, x parameter (d) words		77+4X	17+n	P, 11, 9
to TSS		37+TS	3	P, 10, 9
thru Task Gate		38+TS	3	P, 10, 9
Indirect intersegment	11111111 | mod 011 r/m	17	8	R, 7
to same level		20	10	P, 9
thru Gate to same level		35	13	P, 9
to inner level, no parameters		69	24	P, 9
to inner level, x parameter (d) words		77+4X	24+n	P, 11, 9
to TSS		37+TS	10	P, 10, 9
thru Task Gate		38+TS	10	P, 10, 9
RET = Return from CALL				
intersegment	11001011	13	8	R, 7
to same level		17	9	P, 9
to outer level		35	12	P, 9
intersegment adding imm. to SP	11001010 | 16-bit disp.			
		14	8	R, 7
to same level		18	9	P, 9
to outer level		36	12	P, 9

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
MULTIPLE-SEGMENT INSTRUCTIONS (Continued)				
JMP = Unconditional Jump				
Direct intersegment	11101010 unsigned full offset, selector	17	2	R, 7, 22
to same level		19	3	P, 9
thru Call Gate to same level		32	6	P, 9
thru TSS		42+TS	3	P, 10, 9
thru Task Gate		43+TS	3	P, 10, 9
Indirect intersegment	11111111 | mod 101 r/m	13	9	R, 7, 9
to same level		18	10	P, 9
thru Call Gate to same level		31	13	P, 9
thru TSS		41+TS	10	P, 10, 9
thru Task Gate		42+TS	10	P, 10, 9
BIT MANIPULATION				
BT = Test bit				
register, immediate	00001111 | 10111010 | 11 100 reg | imm. 8-bit data	3		
memory, immediate	00001111 | 10111010 | mod 100 r/m | imm. 8-bit data	3	1	
reg1, reg2	00001111 | 10100011 | 11 reg2 reg1	3		
memory, reg	00001111 | 10100011 | mod reg r/m	8	2	
register, immediate	00001111 | 10111010 | 11 TTT reg | imm. 8-bit data	6		
memory, immediate	00001111 | 10111010 | mod TTT r/m | imm. 8-bit data	8	2/0	U/L
reg1, reg2	00001111 | 10TTT011 | 11 reg2 reg1	6		
memory, reg	00001111 | 10TTT011 | mod reg r/m	13	3/1	U/L
BSF = Scan Bit Forward				
reg1, reg2	00001111 | 10111100 | 11 reg2 reg1	6/42		MN/MX, 12
memory, reg	00001111 | 10111100 | mod reg r/m	7/43	2	MN/MX, 13
BSR = Scan Bit Reverse				
reg1, reg2	00001111 | 10111101 | 11 reg2 reg1	6/103		MN/MX, 14
memory, reg	00001111 | 10111101 | mod reg r/m	7/104	1	MN/MX, 15
STRING INSTRUCTIONS				
CMPS = Compare Byte Word	1010011w	8	6	16
LODS = Load Byte/Word to AL/AX/EAX	1010110w	5	2	
MOVS = Move Byte/Word	1010010w	7	2	16
SCAS = Scan Byte/Word	1010111w	6	2	
STOS = Store Byte/Word from AL/AX/EX	1010101w	5		
XLAT = Translate String	11010111	4	2	

Instruction	TTT
BTS = Test Bit and Set	101
BTR = Test Bit and Reset	110
BTC = Test Bit and Compliment	111

5

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
REPEATED STRING INSTRUCTIONS				
Repeated by Count in CX or ECX (C = Count in CX or ECX)				
REPE CMPS = Compare String (Find Non-Match)	11110011 1010011w			
C = 0		5		
C > 0		7+7c		16, 17
REPNE CMPS = Compare String (Find Match)	11110010 1010011w			
C = 0		5		
C > 0		7+7c		16, 17
REP LODS = Load String	11110010 1010110w			
C = 0		5		
C > 0		7+4c		16, 18
REP MOVS = Move String	11110010 1010010w			
C = 0		5		
C = 1		13	1	16
C > 1		12+3c		16, 19
REPE SCAS = Scan String (Find Non-AL/AX/EAX)	11110011 1010111w			
C = 0		5		
C > 0		7+5c		20
REPNE SCAS = Scan String (Find AL/AX/EAX)	11110010 1010111w			
C = 0		5		
C > 0		7+5c		20
REP STOS = Store String	11110010 1010101w			
C = 0		5		
C > 0		7+4c		
FLAG CONTROL				
CLC = Clear Carry Flag	11111000	2		
STC = Set Carry Flag	11111001	2		
CMC = Complement Carry Flag	11110101	2		
CLD = Clear Direction Flag	11111100	2		
STD = Set Direction Flag	11111101	2		
CLI = Clear Interrupt Enable Flag	11111010	5		
STI = Set Interrupt Enable Flag	11111011	5		
LAHF = Load AH into Flag	10011111	3		
SAHF = Store AH into Flags	10011110	2		
PUSHF = Push Flags	10011100	4/3		RV/P
POPF = Pop Flags	10011101	9/6		RV/P
DECIMAL ARITHMETIC				
AAA = ASCII Adjust for Add	00110111	3		
AAS = ASCII Adjust for Subtract	00111111	3		
AAM = ASCII Adjust for Multiply	11010100 00001010	15		

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
DECIMAL ARITHMETIC (Continued)				
AAD = ASCII Adjust for Divide	11010101 00001010	14		
DAA = Decimal Adjust for Add	00100111	2		
DAS = Decimal Adjust for Subtract	00101111	2		
PROCESSOR CONTROL INSTRUCTIONS				
HLT = Halt	11110100	4		
MOV = Move To and From Control/Debug/Test Registers				
CR0 from register	00001111 00100010 11 000 reg	17	2	
CR2/CR3 from register	00001111 00100010 11 eee reg	4		
Reg from CR0–3	00001111 00100000 11 eee reg	4		
DR0–3 from register	00001111 00100011 11 eee reg	10		
DR6–7 from register	00001111 00100011 11 eee reg	10		
Register from DR6–7	00001111 00100001 11 eee reg	9		
Register from DR0–3	00001111 00100001 11 eee reg	9		
TR3 from register	00001111 00100110 11 011 reg	4		
TR4–7 from register	00001111 00100110 11 eee reg	4		
Register from TR3	00001111 00100100 11 011 reg	3		
Register from TR4–7	00001111 00100100 11 eee reg	4		
CLTS = Clear Task Switched Flag	00001111 00000110	7	2	
INVD = Invalidate Data Cache	00001111 00001000	4		
WBINVD = Write-Back and Invalidate Data Cache	00001111 00001001	5		
INVLPG = Invalidate TLB Entry				
INVLPG memory	00001111 00000001 mod 111 r/m	12/11		H/NH
PREFIX BYTES				
Address Size Prefix	01100111	1		
LOCK = Bus Lock Prefix	11110000	1		
Operand Size Prefix	01100110	1		
Segment Override Prefix				
CS:	00101110	1		
DS:	00111110	1		
ES:	00100110	1		
FS:	01100100	1		
GS:	01100101	1		
SS:	00110110	1		

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
PROTECTION CONTROL				
ARPL = Adjust Requested Privilege Level				
From register	01100011 | 11 reg1 reg2	9		
From memory	01100011 | mod reg r/m	9		
LAR = Load Access Rights				
From register	00001111 | 00000010 | 11 reg1 reg2	11	3	
From memory	00001111 | 00000010 | mod reg r/m	11	5	
LGDT = Load Global Descriptor				
Table register	00001111 | 00000001 | mod 010 r/m	12	5	
LIDT = Load Interrupt Descriptor				
Table register	00001111 | 00000001 | mod 011 r/m	12	5	
LLDT = Load Local Descriptor				
Table register from reg.	00001111 | 00000000 | 11 010 reg	11	3	
Table register from mem.	00001111 | 00000000 | mod 010 r/m	11	6	
LMSW = Load Machine Status Word				
From register	00001111 | 00000001 | 11 110 reg	13		
From memory	00001111 | 00000001 | mod 110 r/m	13	1	
LSL = Load Segment Limit				
From register	00001111 | 00000011 | 11 reg1 reg2	10	3	
From memory	00001111 | 00000011 | mod reg r/m	10	6	
LTR = Load Task Register				
From Register	00001111 | 00000000 | 11 011 reg	20		
From Memory	00001111 | 00000000 | mod 011 r/m	20		
SGDT = Store Global Descriptor Table				
	00001111 | 00000001 | mod 000 r/m	10		
SIDT = Store Interrupt Descriptor Table				
	00001111 | 00000001 | mod 001 r/m	10		
SLDT = Store Local Descriptor Table				
To register	00001111 | 00000000 | 11 000 reg	2		
To memory	00001111 | 00000000 | mod 000 r/m	3		
SMSW = Store Machine Status Word				
To register	00001111 | 00000001 | 11 100 reg	2		
To memory	00001111 | 00000001 | mod 100 r/m	3		
STR = Store Task Register				
To register	00001111 | 00000000 | 11 001 reg	2		
To memory	00001111 | 00000000 | mod 001 r/m	3		
VERR = Verify Read Access				
Register	00001111 | 00000000 | 11 100 r/m	11	3	
Memory	00001111 | 00000000 | mod 100 r/m	11	7	
VERW = Verify Write Access				
To register	00001111 | 00000000 | 11 101 reg	11	3	
To memory	00001111 | 00000000 | mod 101 r/m	11	7	

Table 10.1. i486™ Microprocessor Integer Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit	Penalty if Cache Miss	Notes
INTERRUPT INSTRUCTIONS				
INT n = Interrupt Type n	11001101 | type	INT+4/0		RV/P, 21
INT 3 = Interrupt Type 3	11001100	INT+0		21
INTO = Interrupt 4 if Overflow Flag Set	11001110			
Taken		INT+2		21
Not Taken		3		21
BOUND = Interrupt 5 if Detect Value Out Range	01100010 | mod reg r/m			
If in range		7	7	21
If out of range		INT+24	7	21
IRET = Interrupt Return	11001111			
Real Mode/Virtual Mode		15	8	
Protected Mode				
To same level		20	11	9
To outer level		36	19	9
To nested task (EFLAGS.NT = 1)		TS+32	4	9, 10
External Interrupt		INT+11		21
NMI = Non-Maskable Interrupt		INT+3		21
Page Fault		INT+24		21
VM86 Exceptions				
CLI		INT+8		21
STI		INT+8		21
INT n		INT+9		
PUSHF		INT+9		21
POPF		INT+8		21
IRET		INT+9		
IN				
Fixed Port		INT+50		21
Variable Port		INT+51		21
OUT				
Fixed Port		INT+50		21
Variable Port		INT+51		21
INS		INT+50		21
OUTS		INT+50		21
REP INS		INT+51		21
REP OUTS		INT+51		21

Task Switch Clock Counts Table		
Method	**Value for TS**	
	Cache Hit	**Miss Penalty**
VM/486 CPU/286 TSS To 486 CPU TSS	162	55
VM/486 CPU/286 TSS To 286 TSS	143	31
VM/486 CPU/286 TSS To VM TSS	140	37

Interrupt Clock Counts Table			
Method	**Value for INT**		
	Cache Hit	**Miss Penalty**	**Notes**
Real Mode	26	2	
Protected Mode			
Interrupt/Trap gate, same level	44	6	9
Interrupt/Trap gate, different level	71	17	9
Task Gate	37 + TS	3	9, 10
Virtual Mode			
Interrupt/Trap gate, different level	82	17	
Task gate	37 + TS	3	10

Abbreviations	**Definition**
16/32	16/32 bit modes
U/L	unlocked/locked
MN/MX	minimum/maximum
L/NL	loop/no loop
RV/P	real and virtual mode/protected mode
R	real mode
P	protected mode
T/NT	taken/not taken
H/NH	hit/no hit

NOTES:
1. Assuming that the operand address and stack address fall in different cache sets.
2. Always locked, no cache hit case.
3. Clocks = 10 + max($\log_2$(|m|),n)
   m = multiplier value (min clocks for m=0)
   n = 3/5 for ±m
4. Clocks = {quotient(count/operand length)}*7+9
   = 8 if count ≤ operand length (8/16/32)
5. Clocks = {quotient(count/operand length)}*7+9
   = 9 if count ≤ operand length (8/16/32)
6. Equal/not equal cases (penalty is the same regardless of lock).
7. Assuming that addresses for memory read (for indirection), stack push/pop, and branch fall in different cache sets.
8. Penalty for cache miss: add 6 clocks for every 16 bytes copied to new stack frame.
9. Add 11 clocks for each unaccessed descriptor load.
10. Refer to task switch clock counts table for value of TS.
11. Add 4 extra clocks to the cache miss penalty for each 16 bytes.

For notes 12–13: (b = 0–3, non-zero byte number);
(i = 0–1, non-zero nibble number);
(n = 0–3, non bit number in nibble);

12. Clocks = 8+4 (b+1) + 3(i+1) + 3(n+1)
    = 6 if second operand = 0
13. Clocks = 9+4(b+1) + 3(i+1) + 3(n+1)
    = 7 if second operand = 0

For notes 14–15: (n = bit position 0–31)

14. Clocks = 7 + 3(32−n)
    6 if second operand = 0
15. Clocks = 8 + 3(32−n)
    7 if second operand = 0
16. Assuming that the two string addresses fall in different cache sets.
17. Cache miss penalty: add 6 clocks for every 16 bytes compared. Entire penalty on first compare.
18. Cache miss penalty: add 2 clocks for every 16 bytes of data. Entire penalty on first load.
19. Cache miss penalty: add 4 clocks for every 16 bytes moved.
    (1 clock for the first operation and 3 for the second)
20. Cache miss penalty: add 4 clocks for every 16 bytes scanned.
    (2 clocks each for first and second operations)
21. Refer to interrupt clock counts table for value of INT
22. Clock count includes one clock for using both displacement and immediate.
23. Refer to assumption 6 in the case of a cache miss.

Table 10.2. i486™ Microprocessor I/O Instructions Clock Count Summary

INSTRUCTION	FORMAT	Real Mode	Protected Mode (CPL≤IOPL)	Protected Mode (CPL>IOPL)	Virtual 86 Mode	Notes
I/O INSTRUCTIONS						
IN = Input from:						
Fixed Port	1110010w port number	14	9	29	27	
Variable Port	1110110w	14	8	28	27	
OUT = Output to:						
Fixed Port	1110011w port number	16	11	31	29	
Variable Port	1110111w	16	10	30	29	
INS = Input Byte/Word from DX Port	0110110w	17	10	32	30	
OUTS = Output Byte/Word to DX Port	0110111w	17	10	32	30	1
REP INS = Input String	11110010 0110110w	16+8c	10+8c	30+8c	29+8c	2
REP OUTS = Output String	11110010 0110111w	17+5c	11+5c	31+5c	30+5c	3

NOTES:
1. Two clock cache miss penalty in all cases.
2. c = count in CX or ECX.
3. Cache miss penalty in all modes: Add 2 clocks for every 16 bytes. Entire penalty on second operation.

5

Table 10.3. i486™ Microprocessor Floating Point Clock Count Summary

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range . . . Upper Range)	Penalty if Cache Miss	Concurrent Execution Avg (Lower Range . . . Upper Range)	Notes
DATA TRANSFER					
FLD = Real Load to ST(0)					
32-bit memory	11011 001 | mod 000 r/m | s-i-b/disp.	3	2		
64-bit memory	11011 101 | mod 000 r/m | s-i-b/disp.	3	3		
80-bit memory	11011 011 | mod 101 r/m | s-i-b/disp.	6	4		
ST(i)	11011 001 | 11000 ST(i)	4			
FILD = Integer Load to ST(0)					
16-bit memory	11011 111 | mod 000 r/m | s-i-b/disp.	14.5(13–16)	2	4	
32-bit memory	11011 011 | mod 000 r/m | s-i-b/disp.	11.5(9–12)	2	4(2–4)	
64-bit memory	11011 111 | mod 101 r/m | s-i-b/disp.	16.8(10–18)	3	7.8(2–8)	
FBLD = BCD Load to ST(0)	11011 111 | mod 100 r/m | s-i-b/disp.	75(70–103)	4	7.7(2–8)	
FST = Store Real from ST(0)					
32-bit memory	11011 001 | mod 010 r/m | s-i-b/disp.	7			1
64-bit memory	11011 101 | mod 010 r/m | s-i-b/disp.	8			2
ST(i)	11011 101 | 11010 ST(i)	3			
FSTP = Store Real from ST(0) and Pop					
32-bit memory	11011 011 | mod 011 r/m | s-i-b/disp.	7			1
64-bit memory	11011 101 | mod 011 r/m | s-i-b/disp.	8			2
80-bit memory	11011 011 | mod 111 r/m | s-i-b/disp.	6			
ST(i)	11011 101 | 11001 ST(i)	3			
FIST = Store Integer from ST(0)					
16-bit memory	11011 111 | mod 010 r/m | s-i-b/disp.	33.4(29–34)			
32-bit memory	11011 011 | mod 010 r/m | s-i-b/disp.	32.4(28–34)			
FISTP = Store Integer from ST(0) and Pop					
16-bit memory	11011 111 | mod 011 r/m | s-i-b/disp.	33.4(29–34)			
32-bit memory	11011 011 | mod 011 r/m | s-i-b/disp.	33.4(29–34)			
64-bit memory	11011 111 | mod 111 r/m | s-i-b/disp.	33.4(29–34)			
FBSTP = Store BCD from ST(0) and Pop	11011 111 | mod 110 r/m | s-i-b/disp.	175(172–176)			
FXCH = Exchange ST(0) and ST(i)	11011 001 | 11001 ST(i)	4			
COMPARISON INSTRUCTIONS					
FCOM = Compare ST(0) with Real					
32-bit memory	11011 000 | mod 010 r/m | s-i-b/disp.	4	2	1	
64-bit memory	11011 100 | mod 010 r/m | s-i-b/disp.	4	3	1	
ST(i)	11011 000 | 11010 ST(i)	4		1	
FCOMP = Compare ST(0) with Real and Pop					
32-bit memory	11011 000 | mod 011 r/m | s-i-b/disp.	4	2	1	
64-bit memory	11011 100 | mod 011 r/m | s-i-b/disp.	4	3	1	
ST(i)	11011 000 | 11011 ST(i)	4		1	

Table 10.3. i486™ Microprocessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range ... Upper Range)	Penalty if Cache Miss	Concurrent Execution Avg (Lower Range ... Upper Range)	Notes
COMPARISON INSTRUCTIONS (Continued)					
FCOMPP = Compare ST(0) with ST(1) and Pop Twice	11011 110 | 1101 1001	5		1	
FICOM = Compare ST(0) with Integer					
16-bit memory	11011 110 | mod 010 r/m | s-i-b/disp.	18(16–20)	2	1	
32-bit memory	11011 010 | mod 010 r/m | s-i-b/disp.	16.5(15–17)	2	1	
FICOMP = Compare ST(0) with Integer					
16-bit memory	11011 110 | mod 011 r/m | s-i-b/disp.	18(16–20)	2	1	
32-bit memory	11011 010 | mod 011 r/m | s-i-b/disp.	16.5(15–17)	2	1	
FTST = Compare ST(0) with 0.0	1,1011 001 | 1110 0100	4		1	
FUCOM = Unordered compare ST(0) with ST(i)	11011 101 | 11100 ST(i)	4		1	
FUCOMP = Unordered compare ST(0) with ST(i) and Pop	11011 101 | 11101 ST(i)	4		1	
FUCOMPP = Unordered compare ST(0) with ST(i) and Pop Twice	11011 101 | 11101 1001	5		1	
FXAM = Examine ST(0)	11011 001 | 1110 0101	8			
CONSTANTS					
FLDZ = Load +0.0 into ST(0)	11011 001 | 1110 1110	4			
FLD1 = Load +1.0 into ST(0)	11011 001 | 1110 1000	4			
FLDPI = Load π into ST(0)	11011 001 | 1110 1011	8		2	
FLDL2T = Load $\log_2(10)$ into ST(0)	11011 001 | 1110 1001	8		2	
FLDL2E = Load $\log_2(e)$ into ST(0)	11011 001 | 1110 1010	8		2	
FLDLG2 = Load $\log_{10}(2)$ into ST(0)	11011 001 | 1110 1100	8		2	
FLDLN2 = Load $\log_e(2)$ into ST(0)	11011 001 | 1110 1101	8		2	
ARITHMETIC					
FADD = Add Real with ST(0)					
ST(0) ← ST(0) + 32-bit memory	11011 000 | mod 000 r/m | s-i-b/disp.	10(8–20)	2	7(5–17)	
ST(0) ← ST(0) + 64-bit memory	11011 100 | mod 000 r/m | s-i-b/disp.	10(8–20)	3	7(5–17)	
ST(d) ← ST(0) + ST(i)	11011 d00 | 11000 ST(i)	10(8–20)		7(5–17)	
FADDP = Add real with ST(0) and Pop (ST(i) ← ST(0) + ST(i))	11011 110 | 11000 ST(i)	10(8–20)		7(5–17)	
FSUB = Subtract real from ST(0)					
ST(0) ← ST(0) − 32-bit memory	11011 000 | mod 100 r/m | s-i-b/disp.	10(8–20)	2	7(5–17)	
ST(0) ← ST(0) − 64-bit memory	11011 100 | mod 100 r/m | s-i-b/disp.	10(8-20)	3	7(5–17)	
ST(d) ← ST(0) − ST(i)	11011 d00 | 11101 ST(i)	10(8–20)		7(5–17)	
FSUBP = Subtract real from ST(0) and Pop (ST(i) ← ST(0) − ST(i))	11011 110 | 11101 ST(i)	10(8–20)		7(5–17)	

Table 10.3. i486™ Microprocessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range ... Upper Range)	Penalty if Cache Miss	Concurrent Execution Avg (Lower Range ... Upper Range)	Notes
ARITHMETIC (Continued)					
FSUBR = Subtract real reversed (Subtract ST(0) from real)					
ST(0) ← 32-bit memory − ST(0)	11011 000 | mod 101 r/m | s-i-b/disp.	10(8–20)	2	7(5–17)	
ST(0) ← 64-bit memory − ST(0)	11011 100 | mod 101 r/m | s-i-b/disp.	10(8–20)	3	7(5–17)	
ST(d) ← ST(i) − ST(0)	11011 d00 | 11100 ST(i)	10(8–20)		7(5–17)	
FSUBRP = Subtract real reversed and Pop (ST(i) ← ST(i) − ST(0))	11011 110 | 11100 ST(i)	10(8–20)		7(5–17)	
FMUL = Multiply real with ST(0)					
ST(0) ← ST(0) × 32-bit memory	11011 000 | mod 001 r/m | s-i-b/disp.	11	2	8	
ST(0) ← ST(0) × 64-bit memory	11011 100 | mod 001 r/m | s-i-b/disp.	14	3	11	
ST(d) ← ST(0) × ST(i)	11011 d00 | 11001 ST(i)	16		13	
FMULP = Multiply ST(0) with ST(i) and Pop (ST(i) ← ST(0) × ST(i))	11011 110 | 11001 ST(i)	16		13	
FDIV = Divide ST(0) by Real					
ST(0) ← ST(0)/32-bit memory	11011 000 | mod 110 r/m | s-i-b/disp.	73	2	70	3
ST(0) ← ST(0)/64-bit memory	11011 100 | mod 100 r/m | s-i-b/disp.	73	3	70	3
ST(d) ← ST(0)/ST(i)	11011 d00 | 11111 ST(i)	73		70	3
FDIVP = Divide ST(0) by ST(i) and Pop (ST(i) ← ST(0)/ST(i))	11011 110 | 11111 ST(i)	73		70	3
FDIVR = Divide real reversed (Real/ST(0))					
ST(0) ← 32-bit memory/ST(0)	11011 000 | mod 111 r/m | s-i-b/disp.	73	2	70	3
ST(0) ← 64-bit memory/ST(0)	11011 100 | mod 111 r/m | s-i-b/disp.	73	3	70	3
ST(d) ← ST(i)/ST(0)	11011 d00 | 11110 ST(i)	73		70	3
FDIVRP = Divide real reversed and Pop (ST(i) ← ST(i)/ST(0))	11011 110 | 11110 ST(i)	73		70	3
FIADD = Add Integer to ST(0)					
ST(0) ← ST(0) + 16-bit memory	11011 110 | mod 000 r/m | s-i-b/disp.	24(20–35)	2	7(5–17)	
ST(0) ← ST(0) + 32-bit memory	11011 010 | mod 000 r/m | s-i-b/disp.	22.5(19–32)	2	7(5–17)	
FISUB = Subtract Integer from ST(0)					
ST(0) ← ST(0) − 16-bit memory	11011 110 | mod 100 r/m | s-i-b/disp.	24(20–35)	2	7(5–17)	
ST(0) ← ST(0) − 32-bit memory	11011 010 | mod 100 r/m | s-i-b/disp.	22.5(19–32)	2	7(5–17)	
FISUBR = Integer Subtract Reversed					
ST(0) ← 16-bit memory − ST(0)	11011 110 | mod 101 r/m | s-i-b/disp.	24(20–35)	2	7(5–17)	
ST(0) ← 32-bit memory − ST(0)	11011 010 | mod 101 r/m | s-i-b/disp.	22.5(19–32)	2	7(5–17)	
FIMUL = Multiply Integer with ST(0)					
ST(0) ← ST(0) × 16-bit memory	11011 110 | mod 001 r/m | s-i-b/disp.	25(23–27)	2	8	
ST(0) ← ST(0) × 32-bit memory	11011 010 | mod 001 r/m | s-i-b/disp.	23.5(22–24)	2	8	
FIDIV = Integer Divide					
ST(0) ← ST(0)/16-bit memory	11011 110 | mod 110 r/m | s-i-b/disp.	87(85–89)	2	70	3
ST(0) ← ST(0)/32-bit memory	11011 010 | mod 110 r/m | s-i-b/disp.	85.5(84–86)	2	70	3

Table 10.3. i486™ Microprocessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range ... Upper Range)	Penalty if Cache Miss	Concurrent Execution Avg (Lower Range ... Upper Range)	Notes
ARITHMETIC (Continued)					
FIDIVR = Integer Divide Reversed					
ST(0) ← 16-bit memory/ST(0)	11011 110 | mod 111 r/m | s-i-b/disp.	87(85–89)	2	70	3
ST(0) ← 32-bit memory/ST(0)	11011 010 | mod 111 r/m | s-i-b/disp.	85.5(84–86)	2	70	3
FSQRT = Square Root	11011 001 | 1111 1010	85.5(83–87)		70	
FSCALE = Scale ST(0) by ST(1)	11011 001 | 1111 1101	31(30–32)		2	
FXTRACT = Extract components of ST(0)	11011 001 | 1111 0100	19(16–20)		4(2–4)	
FPREM = Partial Reminder	11011 001 | 1111 1000	84(70–138)		2(2–8)	
FPREM1 = Partial Reminder (IEEE)	11011 001 | 1111 0101	94.5(72–167)		5.5(2–18)	
FRNDINT = Round ST(0) to integer	11011 001 | 1111 1100	29.1(21–30)		7.4(2–8)	
FABS = Absolute value of ST(0)	11011 001 | 1110 0001	3			
FCHS = Change sign of ST(0)	11011 001 | 1110 0000	6			
TRANSCENDENTAL					
FCOS = Cosine of ST(0)	11011 001 | 1111 1111	241(193–279)		2	6, 7
FPTAN = Partial tangent of ST(0)	11011 001 | 1111 0010	244(200–273)		70	6, 7
FPATAN = Partial arctangent	11011 001 | 1111 0011	289(218–303)		5(2–17)	6
FSIN = Sine of ST(0)	11011 001 | 1111 1110	241(193–279)		2	6, 7
FSINCOS = Sine and cosine of ST(0)	11011 001 | 1111 1011	291(243–329)		2	6, 7
F2XM1 = $2^{ST(0)} - 1$	11011 001 | 1111 0000	242(140–279)		2	6
FYL2X = ST(1) × $\log_2$(ST(0))	11011 001 | 1111 0001	311(196–329)		13	6
FYL2XP1 = ST(1) × $\log_2$(ST(0) + 1.0)	11011 001 | 1111 1001	313(171–326)		13	6
PROCESSOR CONTROL					
FINIT = Initialize FPU	11011 011 | 1110 0011	17			4
FSTSW AX = Store status word into AX	11011 111 | 1110 0000	3			5
FSTSW = Store status word into memory	11011 101 | mod 111 r/m | s-i-b/disp.	3			5
FLDCW = Load control word	11011 001 | mod 101 r/m | s-i-b/disp.	4	2		
FSTCW = Store control word	11011 001 | mod 111 r/m | s-i-b/disp.	3			5
FCLEX = Clear exceptions	11011 011 | 1110 0010	7			4
FSTENV = Store environment	11011 001 | mod 110 r/m | s-i-b/disp.				
Real and Virtual modes 16-bit Address		67			4
Real and Virtual modes 32-bit Address		67			4
Protected mode 16-bit Address		56			4
Protected mode 32-bit Address		56			4
FLDENV = Load environment	11011 001 | mod 100 r/m | s-i-b/disp.				
Real and Virtual modes 16-bit Address		44	2		
Real and Virtual modes 32-bit Address		44	2		
Protected mode 16-bit Address		34	2		
Protected mode 32-bit Address		34	2		

Table 10.3. i486™ Microprocessor Floating Point Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Cache Hit Avg (Lower Range ... Upper Range)	Penalty if Cache Miss	Concurrent Execution Avg (Lower Range ... Upper Range)	Notes
PROCESSOR CONTROL (Continued)					
FSAVE = Save state	11011 101 | mod 110 r/m | s-i-b/disp.				
Real and Virtual modes 16-bit Address		154			4
Real and Virtual modes 32-bit Address		154			4
Protected mode 16-bit Address		143			4
Protected mode 32-bit Address		143			4
FRSTOR = Restore state	11011 101 | mod 100 r/m | s-i-b/				
Real and Virtual modes 16-bit Address		131	23		
Real and Virtual modes 32-bit Address		131	27		
Protected mode 16-bit Address		120	23		
Protected mode 32-bit Address		120	27		
FINCSTP = Increment Stack Pointer	11011 001 | 1111 0111	3			
FDECSTP = Decrement Stack Pointer	11011 001 | 1111 0110	3			
FFREE = Free ST(i)	11011 101 | 11000 ST(i)	3			
FNOP = No operations	11011 001 | 1101 0000	3			
WAIT = Wait until FPU ready	10011011				
(Minimum/Maximum)		1/3			

NOTES:
1. If operand is 0 clock counts = 27.
2. If operand is 0 clock counts = 28.
3. If CW.PC indicates 24 bit precision then subtract 38 clocks.
   If CW.PC indicates 53 bit precision then subtract 11 clocks.
4. If there is a numeric error pending from a previous instruction add 17 clocks.
5. If there is a numeric error pending from a previous instruction add 18 clocks.
6. The INT pin is polled several times while this instruction is executing to assure short interrupt latency.
7. If ABS(operand) is greater than $\pi/4$ then add n clocks. Where n = (operand/(π/4)).

10.2 Instruction Encoding

10.2.1 OVERVIEW

All instruction encodings are subsets of the general instruction format shown in Figure 10.1. Instructions consist of one or two primary opcode bytes, possibly an address specifier consisting of the "mod r/m" byte and "scaled index" byte, a displacement if required, and an immediate data field if required.

Within the primary opcode or opcodes, smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary opcode byte(s). This byte, the mod r/m byte, specifies the address mode to be used. Certain encodings of the mod r/m byte indicate a second addressing byte, the scale-index-base byte, follows the mod r/m byte to fully specify the addressing mode.

Addressing modes can include a displacement immediately following the mod r/m byte, or scaled index byte. If a displacement is present, the possible sizes are 8, 16 or 32 bits.

If the instruction specifies an immediate operand, the immediate operand follows any displacement bytes. The immediate operand, if specified, is always the last field of the instruction.

Figure 10.1 illustrates several of the fields that can appear in an instruction, such as the mod field and the r/m field, but the Figure does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the opcode bytes themselves. Table 10.4 is a complete list of all fields appearing in the 486 Microprocessor instruction set. Further ahead, following Table 10.4, are detailed tables for each field.

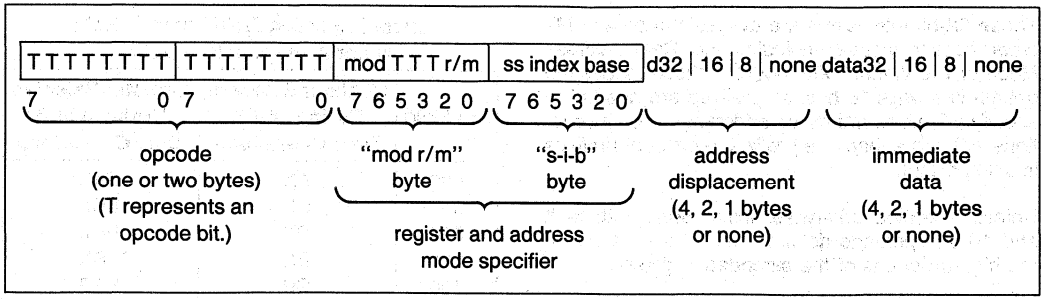


Figure 10.1. General Instruction Format

Table 10.4. Fields within i486™ Microprocessor Instructions

Field Name	Description	Number of Bits
w	Specifies if Data is Byte or Full Size (Full Size is either 16 or 32 Bits	1
d	Specifies Direction of Data Operation	1
s	Specifies if an Immediate Data Field Must be Sign-Extended	1
reg	General Register Specifier	3
mod r/m	Address Mode Specifier (Effective Address can be a General Register)	2 for mod; 3 for r/m
ss	Scale Factor for Scaled Index Address Mode	2
index	General Register to be used as Index Register	3
base	General Register to be used as Base Register	3
sreg2	Segment Register Specifier for CS, SS, DS, ES	2
sreg3	Segment Register Specifier for CS, SS, DS, ES, FS, GS	3
tttn	For Conditional Instructions, Specifies a Condition Asserted or a Condition Negated	4

NOTE:
Tables 10.1–10.3 show encoding of individual instructions.

10.2.2 32-BIT EXTENSIONS OF THE INSTRUCTION SET

With the 486 Microprocessor, the 8086/80186/80286 instruction set is extended in two orthogonal directions: 32-bit forms of all 16-bit instructions are added to support the 32-bit data types, and 32-bit addressing modes are made available for all instructions referencing memory. This orthogonal instruction set extension is accomplished having a Default (D) bit in the code segment descriptor, and by having 2 prefixes to the instruction set.

Whether the instruction defaults to operations of 16 bits or 32 bits depends on the setting of the D bit in the code segment descriptor, which gives the default length (either 32 bits or 16 bits) for both operands and effective addresses when executing that code segment. In the Real Address Mode or Virtual 8086 Mode, no code segment descriptors are used, but a D value of 0 is assumed internally by the 486 Microprocessor when operating in those modes (for 16-bit default sizes compatible with the 8086/80186/80286).

Two prefixes, the Operand Size Prefix and the Effective Address Size Prefix, allow overriding individually the Default selection of operand size and effective address size. These prefixes may precede any opcode bytes and affect only the instruction they precede. If necessary, one or both of the prefixes may be placed before the opcode bytes. The presence of the Operand Size Prefix and the Effective Address Prefix will toggle the operand size or the effective address size, respectively, to the value "opposite" from the Default setting. For example, if the default operand size is for 32-bit data operations, then presence of the Operand Size Prefix toggles the instruction to 16-bit data operation. As another example, if the default effective address size is 16 bits, presence of the Effective Address Size prefix toggles the instruction to use 32-bit effective address computations.

These 32-bit extensions are available in all 486 Microprocessor modes, including the Real Address Mode or the Virtual 8086 Mode. In these modes the default is always 16 bits, so prefixes are needed to specify 32-bit operands or addresses. For instructions with more than one prefix, the order of prefixes is unimportant.

Unless specified otherwise, instructions with 8-bit and 16-bit operands do not affect the contents of the high-order bits of the extended registers.

10.2.3 ENCODING OF INTEGER INSTRUCTION FIELDS

Within the instruction are several fields indicating register selection, addressing mode and so on. The exact encodings of these fields are defined immediately ahead.

10.2.3.1 Encoding of Operand Length (w) Field

For any given instruction performing a data operation, the instruction is executing as a 32-bit operation or a 16-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or the full operation size, as shown in the table below.

w Field	Operand Size During 16-Bit Data Operations	Operand Size During 32-Bit Data Operations
0	8 Bits	8 Bits
1	16 Bits	32 Bits

10.2.3.2 Encoding of the General Register (reg) Field

The general register is specified by the reg field, which may appear in the primary opcode bytes, or as the reg field of the "mod r/m" byte, or as the r/m field of the "mod r/m" byte.

Encoding of reg Field When w Field is not Present in Instruction

reg Field	Register Selected During 16-Bit Data Operations	Register Selected During 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
110	SI	ESI
111	DI	EDI

Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field During 16-Bit Data Operations:		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Register Specified by reg Field During 32-Bit Data Operations		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

10.2.3.3 Encoding of the Segment Register (sreg) Field

The sreg field in certain instructions is a 2-bit field allowing one of the four 80286 segment registers to be specified. The sreg field in other instructions is a 3-bit field, allowing the 486 Microprocessor FS and GS segment registers to be specified.

2-Bit sreg2 Field

2-Bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

3-Bit sreg3 Field

3-Bit sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	do not use
111	do not use

10.2.3.4 Encoding of Address Mode

Except for special instructions, such as PUSH or POP, where the addressing mode is pre-determined, the addressing mode for the current instruction is specified by addressing bytes following the primary opcode. The primary addressing byte is the "mod r/m" byte, and a second byte of addressing information, the "s-i-b" (scale-index-base) byte, can be specified.

The s-i-b byte (scale-index-base byte) is specified when using 32-bit addressing mode and the "mod r/m" byte has r/m = 100 and mod = 00, 01 or 10. When the sib byte is present, the 32-bit addressing mode is a function of the mod, ss, index, and base fields.

The primary addressing byte, the "mod r/m" byte, also contains three bits (shown as TTT in Figure 10.1) sometimes used as an extension of the primary opcode. The three bits, however, may also be used as a register field (reg).

When calculating an effective address, either 16-bit addressing or 32-bit addressing is used. 16-bit addressing uses 16-bit address components to calculate the effective address while 32-bit addressing uses 32-bit address components to calculate the effective address. When 16-bit addressing is used, the "mod r/m" byte is interpreted as a 16-bit addressing mode specifier. When 32-bit addressing is used, the "mod r/m" byte is interpreted as a 32-bit addressing mode specifier.

Tables on the following three pages define all encodings of all 16-bit addressing modes and 32-bit addressing modes.

Encoding of 16-bit Address Mode with "mod r/m" Byte

mod r/m	Effective Address
00 000	DS:[BX + SI]
00 001	DS:[BX + DI]
00 010	SS:[BP + SI]
00 011	SS:[BP + DI]
00 100	DS:[SI]
00 101	DS:[DI]
00 110	DS:d16
00 111	DS:[BX]
01 000	DS:[BX + SI + d8]
01 001	DS:[BX + DI + d8]
01 010	SS:[BP + SI + d8]
01 011	SS:[BP + DI + d8]
01 100	DS:[SI + d8]
01 101	DS:[DI + d8]
01 110	SS:[BP + d8]
01 111	DS:[BX + d8]

mod r/m	Effective Address
10 000	DS:[BX + SI + d16]
10 001	DS:[BX + DI + d16]
10 010	SS:[BP + SI + d16]
10 011	SS:[BP + DI + d16]
10 100	DS:[SI + d16]
10 101	DS:[DI + d16]
10 110	SS:[BP + d16]
10 111	DS:[BX + d16]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by r/m During 16-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by r/m During 32-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode with "mod r/m" byte (no "s-i-b" byte present):

mod r/m	Effective Address
00 000	DS:[EAX]
00 001	DS:[ECX]
00 010	DS:[EDX]
00 011	DS:[EBX]
00 100	s-i-b is present
00 101	DS:d32
00 110	DS:[ESI]
00 111	DS:[EDI]
01 000	DS:[EAX + d8]
01 001	DS:[ECX + d8]
01 010	DS:[EDX + d8]
01 011	DS:[EBX + d8]
01 100	s-i-b is present
01 101	SS:[EBP + d8]
01 110	DS:[ESI + d8]
01 111	DS:[EDI + d8]

mod r/m	Effective Address
10 000	DS:[EAX + d32]
10 001	DS:[ECX + d32]
10 010	DS:[EDX + d32]
10 011	DS:[EBX + d32]
10 100	s-i-b is present
10 101	SS:[EBP + d32]
10 110	DS:[ESI + d32]
10 111	DS:[EDI + d32]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by reg or r/m during 16-Bit Data Operations:		
mod r/m	Function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by reg or r/m during 32-Bit Data Operations:		
mod r/m	Function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode ("mod r/m" byte and "s-i-b" byte present):

mod base	Effective Address
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

NOTE:
Mod field in "mod r/m" byte; ss, index, base fields in "s-i-b" byte.

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8

index	Index Register
000	EAX
001	ECX
010	EDX
011	EBX
100	no index reg**
101	EBP
110	ESI
111	EDI

****IMPORTANT NOTE:**
When index field is 100, indicating "no index register," then ss field MUST equal 00. If index is 100 and ss does not equal 00, the effective address is undefined.

10.2.3.5 Encoding of Operation Direction (d) Field

In many two-operand instructions the d field is present to indicate which operand is considered the source and which is the destination.

d	Direction of Operation
0	Register/Memory <- - Register "reg" Field Indicates Source Operand; "mod r/m" or "mod ss index base" Indicates Destination Operand
1	Register <- - Register/Memory "reg" Field Indicates Destination Operand; "mod r/m" or "mod ss index base" Indicates Source Operand

10.2.3.6 Encoding of Sign-Extend (s) Field

The s field occurs primarily to instructions with immediate data fields. The s field has an effect only if the size of the immediate data is 8 bits and is being placed in a 16-bit or 32-bit destination.

s	Effect on Immediate Data8	Effect on Immediate Data 16|32
0	None	None
1	Sign-Extend Data8 to Fill 16-Bit or 32-Bit Destination	None

10.2.3.7 Encoding of Conditional Test (tttn) Field

For the conditional instructions (conditional jumps and set on condition), tttn is encoded with n indicating to use the condition (n = 0) or its negation (n = 1), and ttt giving the condition to test.

Mnemonic	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less or Equal/Greater Than	1111

10.2.3.8 Encoding of Control or Debug or Test Register (eee) Field

For the loading and storing of the Control, Debug and Test registers.

When Interpreted as Control Register Field

eee Code	Reg Name
000	CR0
010	CR2
011	CR3
Do not use any other encoding	

When Interpreted as Debug Register Field

eee Code	Reg Name
000	DR0
001	DR1
010	DR2
011	DR3
110	DR6
111	DR7
Do not use any other encoding	

When Interpreted as Test Register Field

eee Code	Reg Name
011	TR3
100	TR4
101	TR5
110	TR6
111	TR7
Do not use any other encoding	

	Instruction								Optional Fields	
	First Byte				Second Byte					
1	11011	OPA		1	mod		1 OPB	r/m	s-i-b	disp
2	11011	MF		OPA	mod		OPB	r/m	s-i-b	disp
3	11011	d	P	OPA	1	1	OPB	ST(i)		
4	11011	0	0	1	1	1	1	OP		
5	11011	0	1	1	1	1	1	OP		
	15–11	10	9	8	7	6	5	4 3 2 1 0		

10.2.4 ENCODING OF FLOATING POINT INSTRUCTION FIELDS

Instructions for the FPU assume one of the five forms shown in the following table. In all cases, instructions are at least two bytes long and begin with the bit pattern 11011B.

OP = Instruction opcode, possible split into two fields OPA and OPB

MF = Memory Format
- 00—32-bit real
- 01—32-bit integer
- 10—64-bit real
- 11—16-bit integer

P = Pop
- 0—Do not pop stack
- 1—Pop stack after operation

d = Destination
- 0—Destination is ST(0)
- 1—Destination is ST(i)

R XOR d = 0—Destination (op) Source
R XOR d = 1—Source (op) Destination

ST(i) = Register stack element *i*
- 000 = Stack top
- 001 = Second stack element
- •
- •
- •
- 111 = Eighth stack element

mod (Mode field) and r/m (Register/Memory specifier) have the same interpretation as the corresponding fields of the integer instructions.

s-i-b (Scale Index Base) byte and disp (displacement) are optionally present in instructions that have mod and r/m fields. Their presence depends on the values of mod and r/m, as for integer instructions.

11.0 DIFFERENCES BETWEEN THE i486™ MICROPROCESSOR AND THE 386™ MICROPROCESSOR PLUS THE 387™ MATH COPROCESSOR EXTENSION

The differences between the 486 microprocessor and the 386 microprocessor are due to performance enhancements. The differences between the microprocessors are listed below.

1. Instruction clock counts have been reduced to achieve higher performance. See Section 10.
2. The 486 microprocessor bus is significantly faster than the 386 microprocessor bus. Differences include a 1X clock, parity support, burst cycles, cacheable cycles, cache invalidate cycles and 8-bit bus support. The Hardware Interface and Bus Operation Sections (Sections 6 and 7) of the data sheet should be carefully read to understand the 486 microprocessor bus functionality.
3. To support the on-chip cache new bits have been added to control register 0 (CD and NW) (Section 2.1.2.1), new pins have been added to the bus (Section 6) and new bus cycle types have been added (Section 7). The on-chip cache needs to be enabled after reset by clearing the CD and NW bit in CR0.
4. The complete 387 math coprocessor instruction set and register set have been added. No I/O cycles are performed during Floating Point instructions. The instruction and data pointers are set to 0 after FINIT/FSAVE. Interrupt 9 can no longer occur, interrupt 13 occurs instead.
5. The 486 microprocessor supports new floating point error reporting modes to guarantee DOS compatibility. These new modes required a new bit in control register 0 (NE) (Section 2.1.2.1) and new pins (FERR# and IGNNE#) (Section 6.2.13 and 7.2.14).
6. In some cases FERR# is asserted when the next floating point instruction is encountered and in other cases it is asserted before the next floating point instruction is encountered, depending upon the execution state the instruction causing exception (see Sections 6.2.13 and 7.2.14). For

both of these cases, the 387 Math Coprocessor asserts ERROR# when the error occurs and does not wait for the next floating point instruction to be encountered.

7. Six new instructions have been added:
   Byte Swap (BSWAP)
   Exchange-and-Add (XADD)
   Compare and Exchange (CMPXCHG)
   Invalidate Data Cache (INVD)
   Write-back and Invalidate Data Cache (WBINVD)
   Invalidate TLB Entry (INVLPG)
8. There are two new bits defined in control register 3, the page table entries and page directory entries (PCD and PWT) (Section 4.5.2.5).
9. A new page protection feature has been added. This feature required a new bit in control register 0 (WP) (Section 2.1.2.1 and 4.5.3).
10. A new Alignment Check feature has been added. This feature required a new bit in the flags register (AC) (Section 2.1.1.3) and a new bit in control register 0 (AM) (Section 2.1.2.1).
11. The replacement algorithm for the translation lookaside buffer has been changed from a random algorithm to a pseudo least recently used algorithm like that used by the on-chip cache. See Section 5.5 for a description of the algorithm.
12. Three new testability registers, TR3, TR4 and TR5, have been added for testing the on-chip cache. TLB testability has been enhanced. See Section 8.
13. The prefetch queue has been increased from 16 bytes to 32 bytes. A jump always needs to execute after modifying code to guarantee correct execution of the new instruction.
14. After reset, the ID in the upper byte of the DX register is 04. The contents of the base registers including the floating point registers may be different after reset.

12.0 ELECTRICAL DATA

The following sections describe recommended electrical connections for the 486 microprocessor, and its electrical specifications.

12.1 Power and Grounding

12.1.1 POWER CONNECTIONS

The 486 microprocessor is implemented in CHMOS IV technology and has modest power requirements. However, its high clock frequency output buffers can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 24 V_{CC} and 28 V_{SS} pins feed the 486 microprocessor.

Power and ground connections must be made to all external V_{CC} and GND pins of the 486 microprocessor. On the circuit board, all V_{CC} pins must be connected on a V_{CC} plane. All V_{SS} pins must be likewise connected on a GND plane.

12.1.2 POWER DECOUPLING RECOMMENDATIONS

Liberal decoupling capacitance should be placed near the 486 microprocessor. The 486 microprocessor driving its 32-bit parallel address and data busses at high frequencies can cause transient power surges, particularly when driving large capacitive loads.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the 486 microprocessor and decoupling capacitors as much as possible. Capacitors specifically for PGA packages are also commercially available.

12.1.3 OTHER CONNECTION RECOMMENDATIONS

N.C. pins should always remain unconnected.

For reliable operation, always connect unused inputs to an appropriate signal level. Active LOW inputs should be connected to V_{CC} through a pullup resistor. Pullups in the range of 20 KΩ are recommended. Active HIGH inputs should be connected to GND.

12.2 Maximum Ratings

Table 12.1 is a stress rating only, and functional operation at the maximums is not guaranteed. Function operating conditions are given in 12.3 D.C. Specifications and 12.4 A.C. Specifications.

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the 486 microprocessor contains protective circuitry to resist damage from static electric discharge, always take precautions to avoid high static voltages or electric fields.

Table 12.1. Absolute Maximum Ratings

Case Temperature under Bias . . . −65°C to +110°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin with
Respect to Ground . . . . . −0.5 to V_{CC} + 0.5V
Supply Voltage with
Respect to V_{SS} . . . . . −0.5V to +6.5V

12.3 D.C. Specifications

Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C

Table 12.2. DC Parametric Values

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC} +0.3	V	
V_{OL}	Output Low Voltage		0.45	V	(Note 1)
V_{OH}	Output High Voltage	2.4		V	(Note 2)
I_{CC}	Power Supply Current (25 MHz)		700	mA	(Note 3)
	Power Supply Current (33 MHz)		900		
I_{LI}	Input Leakage Current		±15	μA	(Note 4)
I_{IH}	Input Leakage Current		200	μA	(Note 5)
I_{IL}	Input Leakage Current		−400	μA	(Note 6)
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance		20	pF	F_C = 1 MHz (Note 7)
C_O	I/O or Output Capacitance		20	pF	F_C = 1 MHz (Note 7)
C_{CLK}	CLK Capacitance		20	pF	F_C = 1 MHz (Note 7)

* PRELIMINARY SEE INTEL FOR DESIGN INFORMATION

NOTES:
1. This parameter is measured at:
   Address, Data, BEn 4.0 mA
   Definition, Control 5.0 mA
2. This parameter is measured at:
   Address, Data, BEn −1.0 mA
   Definition, Control −0.9 mA
3. Typical supply current:
   550 mA @ 25 MHz
   700 mA @ 33 MHz
4. This parameter is for inputs without internal pullups or pulldowns and $0 \le V_{IN} \le V_{CC}$.
5. This parameter is for inputs with internal pulldowns and V_{IH} = 2.4V.
6. This parameter is for inputs with internal pullups and V_{IL} = 0.45V.
7. Not 100% tested.

12.4 A.C. Specifications

The A.C. specifications, given in Table 12.3, consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the rising edge of the CLK signal.

A.C. specifications measurement is defined by Figures 12.1−12.3. Inputs must be driven to the voltage levels indicated by Figure 12.3 when A.C. specifications are measured. 486 microprocessor output delays are specified with minimum and maximum limits, measured as shown. The minimum 486 microprocessor delay times are hold times provided to external circuitry. 486 microprocessor input setup and hold times are specified as minimums, defining the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct 486 microprocessor operation.

Table 12.3. 25 MHz i486 Microprocessor A.C. Characteristics

V_{CC} = 5V ±5%; T_{case} = 0°C to +85°C; C_l = 50 pF unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	25	MHz		1X Clock Driven to 486
t_1	CLK Period	40	125	ns	12.1	
t_{1a}	CLK Period Stability		0.1%	Δ		Adjacent Clocks
t_2	CLK High Time	14		ns	12.1	at 2V
t_3	CLK Low Time	14		ns	12.1	at 0.8V
t_4	CLK Fall Time		4	ns	12.1	
t_5	CLK Rise Time		4	ns	12.1	
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	22	ns	12.5	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		30	ns	12.6	Note 1
t_8	PCHK# Valid Delay	3	27	ns	12.4	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	27	ns	12.5	
t_9	BLAST#, PLOCK# Float Delay		30	ns	12.6	Note 1
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	22	ns	12.5	
t_{11}	D0–D31, DP0–3 Write Data Float Delay		30	ns	12.6	Note 1
t_{12}	EADS# Setup Time	8		ns	12.2	
t_{13}	EADS# Hold Time	3		ns	12.2	
t_{14}	KEN#, BS16#, BS8# Setup Time	8		ns	12.2	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	12.2	
t_{16}	RDY#, BRDY# Setup Time	8		ns	12.3	
t_{17}	RDY#, BRDY# Hold Time	3		ns	12.3	
t_{18}	HOLD, AHOLD, BOFF# Setup Time	10		ns	12.2	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	12.2	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	10		ns	12.2	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	3		ns	12.2	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	5		ns	12.2	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	3		ns	12.2	

PRELIMINARY
SEE INTEL FOR DESIGN-IN
INFORMATION

NOTE:
1. Not 100% tested. Guaranteed by design characterization.

Table 12.3. 33 MHz i486 Microprocessor A.C. Characteristics

V_{CC} = 5V ±5%; T_{case} = 0°C to +85°C; C_l = 50 pF unless otherwise specified

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Frequency	8	33	MHz		1X Clock Driven to 486
t_1	CLK Period	30	125	ns	12.1	
t_{1a}	CLK Period Stability		0.1%	Δ		Adjacent Clocks
t_2	CLK High Time	11		ns	12.1	at 2V
t_3	CLK Low Time	11		ns	12.1	at 0.8V
t_4	CLK Fall Time		3	ns	12.1	
t_5	CLK Rise Time		3	ns	12.1	
t_6	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK#, FERR#, BREQ, HLDA Valid Delay	3	16	ns	12.5	
t_7	A2–A31, PWT, PCD, BE0–3#, M/IO#, D/C#, W/R#, ADS#, LOCK# Float Delay		20	ns	12.6	Note 1
t_8	PCHK# Valid Delay	3	22	ns	12.4	
t_{8a}	BLAST#, PLOCK# Valid Delay	3	20	ns	12.5	
t_9	BLAST#, PLOCK# Float Delay		20	ns	12.6	Note 1
t_{10}	D0–D31, DP0–3 Write Data Valid Delay	3	18	ns	12.5	
t_{11}	D0–D31, DP0–3 Write Data Float Delay		20	ns	12.6	Note 1
t_{12}	EADS# Setup Time	5		ns	12.2	
t_{13}	EADS# Hold Time	3		ns	12.2	
t_{14}	KEN#, BS16#, BS8# Setup Time	5		ns	12.2	
t_{15}	KEN#, BS16#, BS8# Hold Time	3		ns	12.2	
t_{16}	RDY#, BRDY# Setup Time	5		ns	12.3	
t_{17}	RDY#, BRDY# Hold Time	3		ns	12.3	
t_{18}	HOLD, AHOLD, Setup Time	6		ns	12.2	
t_{18a}	BOFF# Setup Time	8		ns	12.2	
t_{19}	HOLD, AHOLD, BOFF# Hold Time	3		ns	12.2	
t_{20}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Setup Time	5		ns	12.2	
t_{21}	RESET, FLUSH#, A20M#, NMI, INTR, IGNNE# Hold Time	3		ns	12.2	
t_{22}	D0–D31, DP0–3, A4–A31 Read Setup Time	5		ns	12.2	
t_{23}	D0–D31, DP0–3, A4–A31 Read Hold Time	3		ns	12.2	

PRELIMINARY
INTEL FOR DESIGN-IN
INFORMATION
SEE

NOTE:
1. Not 100% tested. Guaranteed by design characterization.

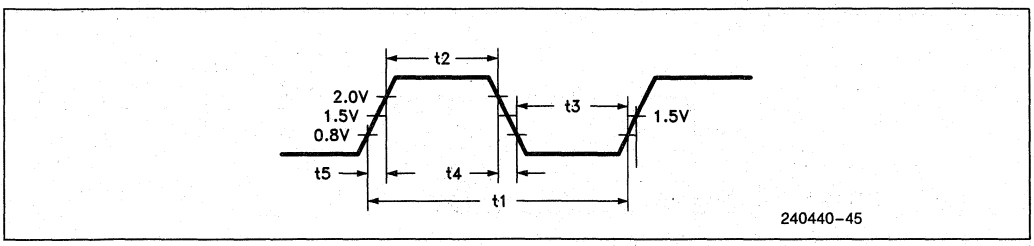


Figure 12.1. CLK Waveforms

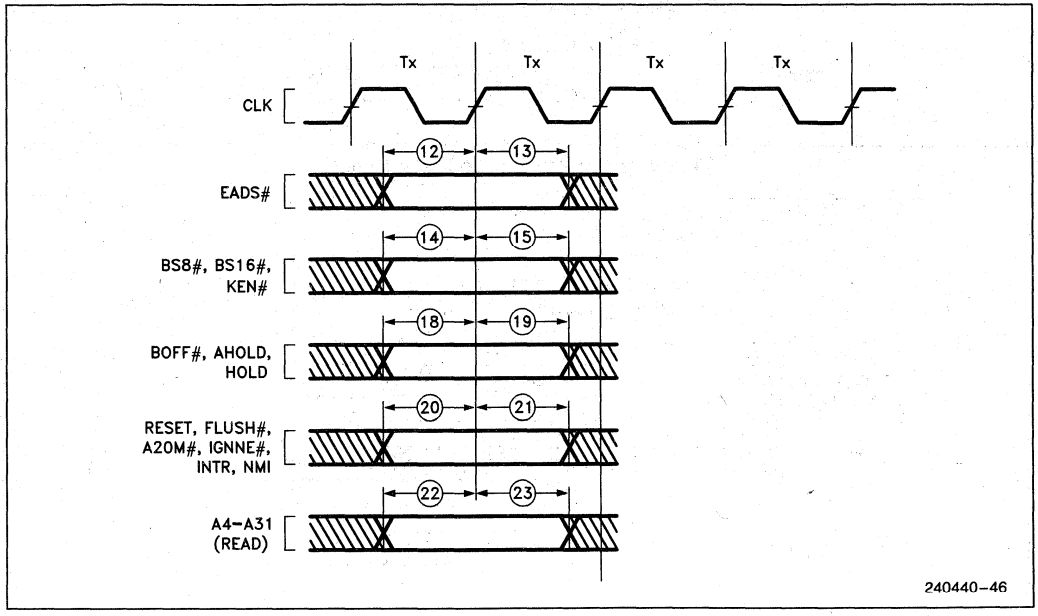


Figure 12.2. Input Setup and Hold Timing

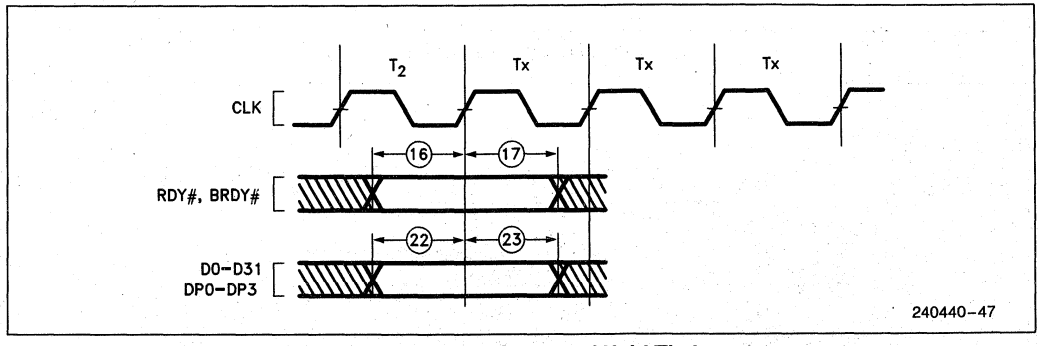


Figure 12.3. Input Setup and Hold Timing

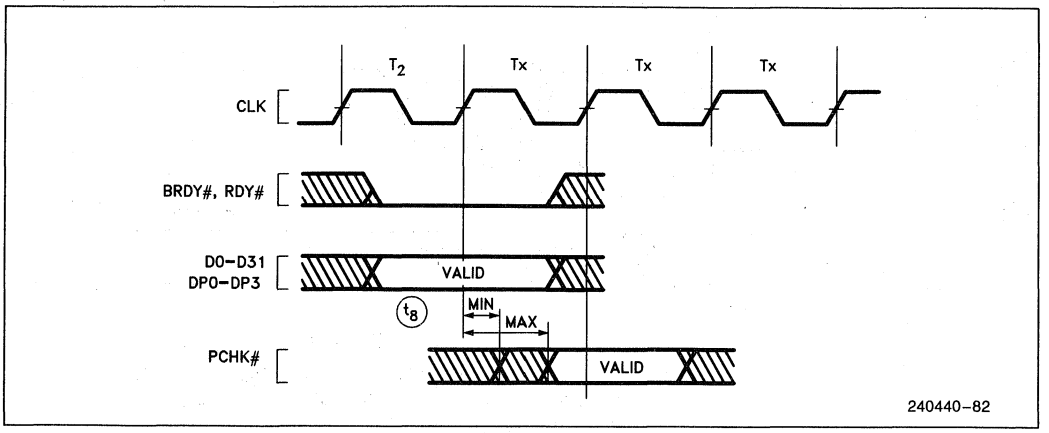


Figure 12.4. PCHK# Valid Delay Timing

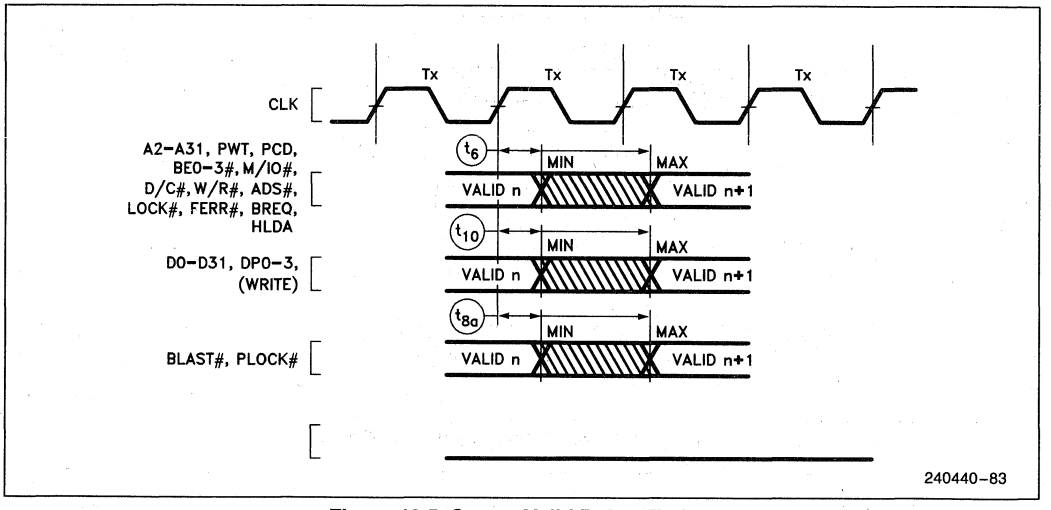


Figure 12.5. Output Valid Delay Timing

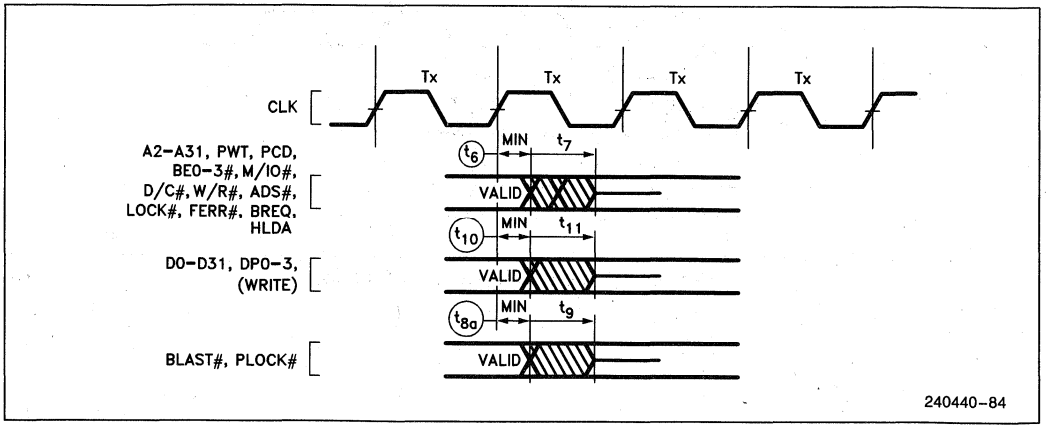
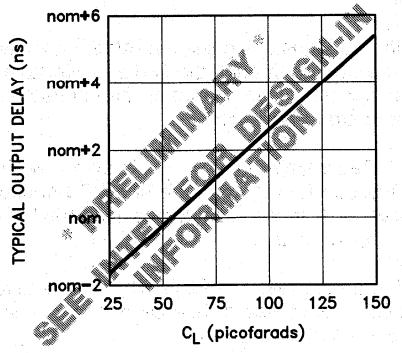


Figure 12.6. Maximum Float Delay Timing

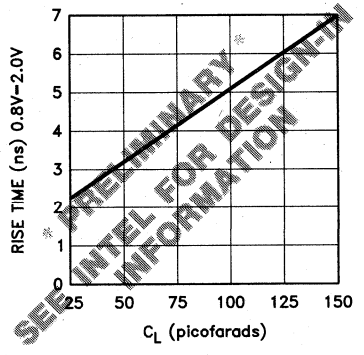
12.4.1 Typical Output Valid Delay versus Load Capacitance Under Worst Case Conditions



240440-75

NOTE:
This graph will not be linear outside of the C_L range shown. nom = nominal value given in A.C. Characteristics table.

12.4.2 Typical Output Rise Time versus Load Capacitance Under Worst-Case Conditions



240440-76

NOTE:
This graph will not be linear outside of the C_L range shown.

12.5 Designing for ICD-486 (Advance Information)

The ICD-486 (In-Circuit Debugger) is a hardware assisted debugger for the 486 CPU. To use the ICD-486, the 486 CPU component must be removed from its socket replaced with the ICD-486 module. Because of the high operating frequency of 486 CPU systems, there is no buffering of signals between the 486 CPU in the ICD-486 and the target system. A direct result of the non-buffered interconnect is that the ICD-486 shares the address and data bus of the target system. In order for the ICD-486 to function properly (without the Optional Isolation Board installed), the design of the target system must meet the following restrictions:

1. The bus controller must only enable data transceivers onto the data bus during valid read cycles of the 486 CPU, other local devices, or other bus masters.
2. Before another bus master drives the local processor address bus, the other bus master must gain access to the address bus through the use of HOLD-HLDA, AHOLD, or BOFF#.

In addition to the above restrictions, the ICD-486 has several electrical and mechanical characteristics that should be taken into consideration when designing the 486 CPU system.

Capacitive Loading: ICD-486 adds up to 30 pF to the CLK signal, and up to 20 pF to each of the other 486 CPU signals.

DC Loading: ICD-486 adds ±15 μA loading to the CLK and data bus signals and ±5 μA loading to the address and control signals.

Power Requirements: For noise immunity and CMOS latch-up protection the ICD-486 is powered by the target system through the power and ground pins of the 486 CPU socket. The circuitry on the ICD-486 draws up to 1.3A excluding the 486 CPU I_{CC}.

No Connects: Pins specified as N.C. in the 486 CPU pin description must be left unconnected. Connection of any of these pins to power, ground, or any other signal may cause the processor or the ICD-486 to malfunction.

486 CPU Location and Orientation: The ICD-486 may require lateral clearance. Figure 12.4 shows the clearance requirements of the ICD-486.

Optional Isolation Board (OIB)

Due to its unbuffered design, the ICD-486 is susceptible to errors on the target system's bus. The OIB installs between the ICD-486 and 486 CPU socket in the target system and allows the ICD-486 to function in systems with faults (i.e., shorted signals). After electrical verification the OIB may be removed. The OIB has the following electrical and mechanical characteristics:

Buffer Characteristics: The OIB buffers the address and data busses as well as the byte enables, ADS#, W/R#, M/IO#, BLAST#, and HLDA. The buffers are advanced CMOS devices and have the following DC drive specifications: $I_{OH} = -15$ mA, $I_{OL} = 64$ mA. The propagation delay of each buffer is 5 ns max driving a 50 pF load. To guarantee proper operation with the OIB, the clock period should be increased by the round trip buffer delay (10 ns) unless the target system design already has enough timing margin.

Unbuffered Signals: Signals not listed above as buffered are passed through the OIB and will have additional capacitive loading due to the connectors and circuit board of up to 10 pF.

Power Requirements: The OIB is also powered by the target system through the 486 CPU socket and requires 0.5A in addition to the ICD-486 and 486 CPU requirements.

OIB Clearance Requirements: The OIB requires an extra 0.55″ of vertical clearance in the target system above the 486 CPU socket.

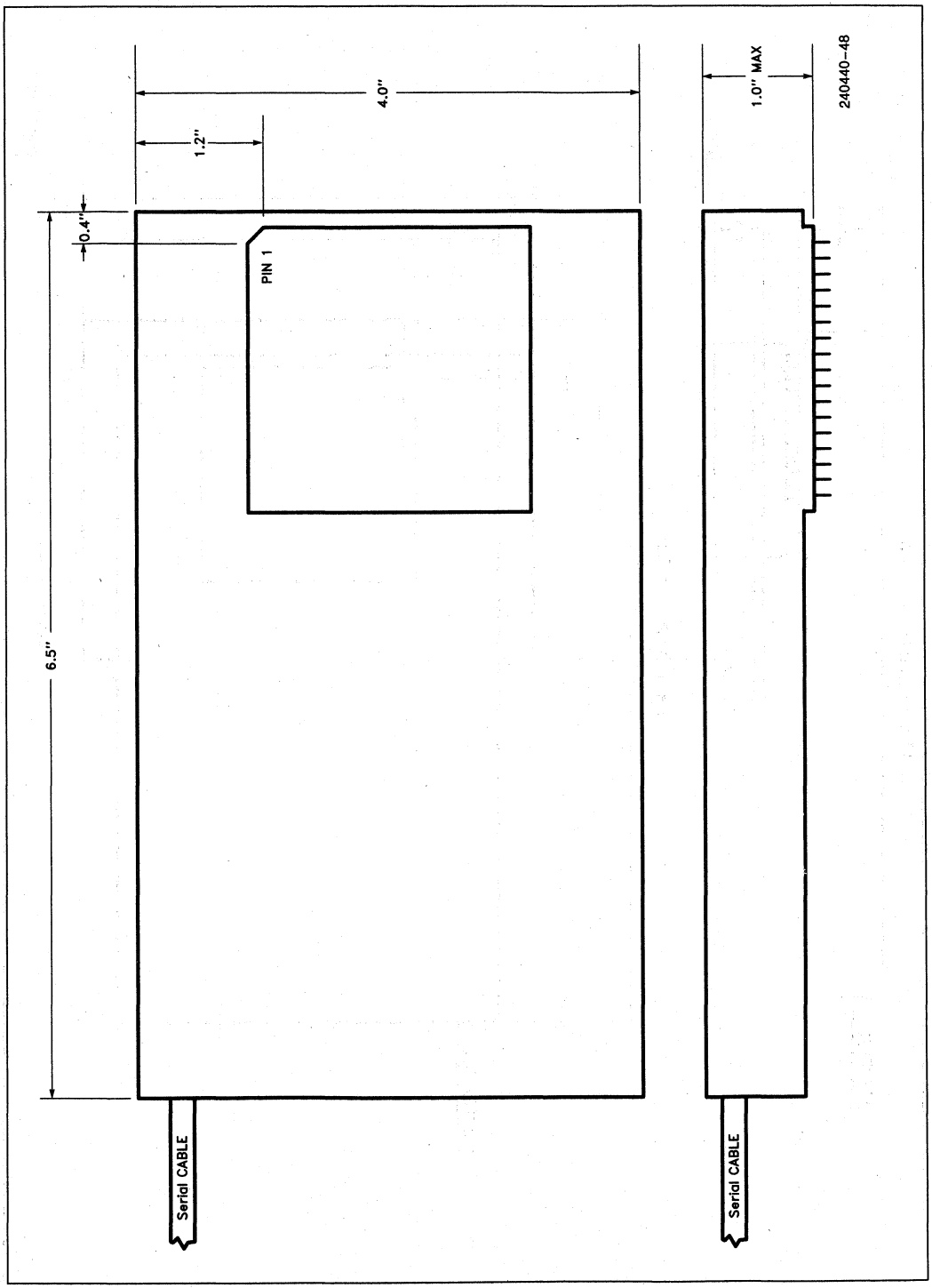


Figure 12.4a. ICD-486™ Probe Dimensions

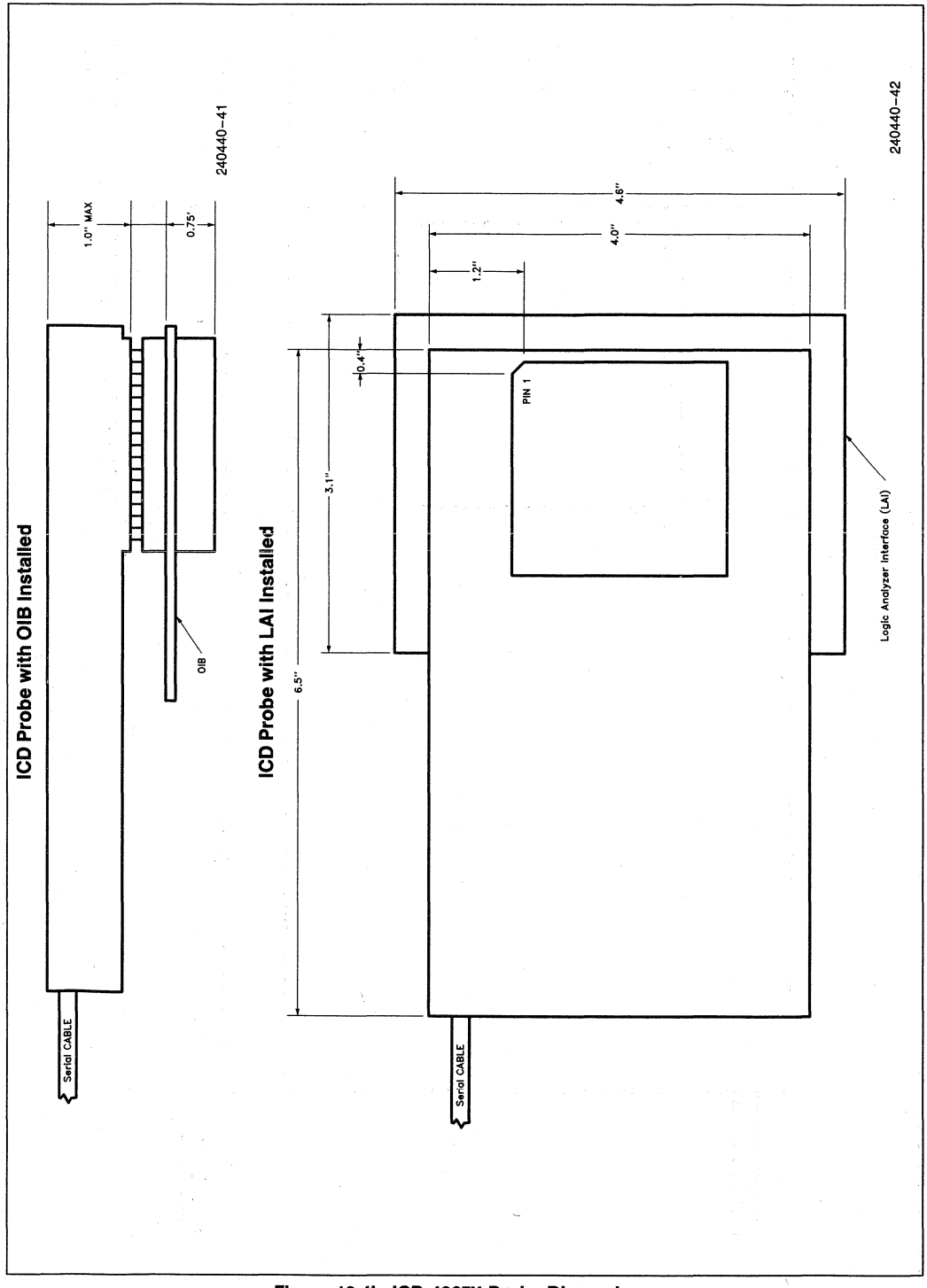


Figure 12.4b. ICD-486™ Probe Dimensions

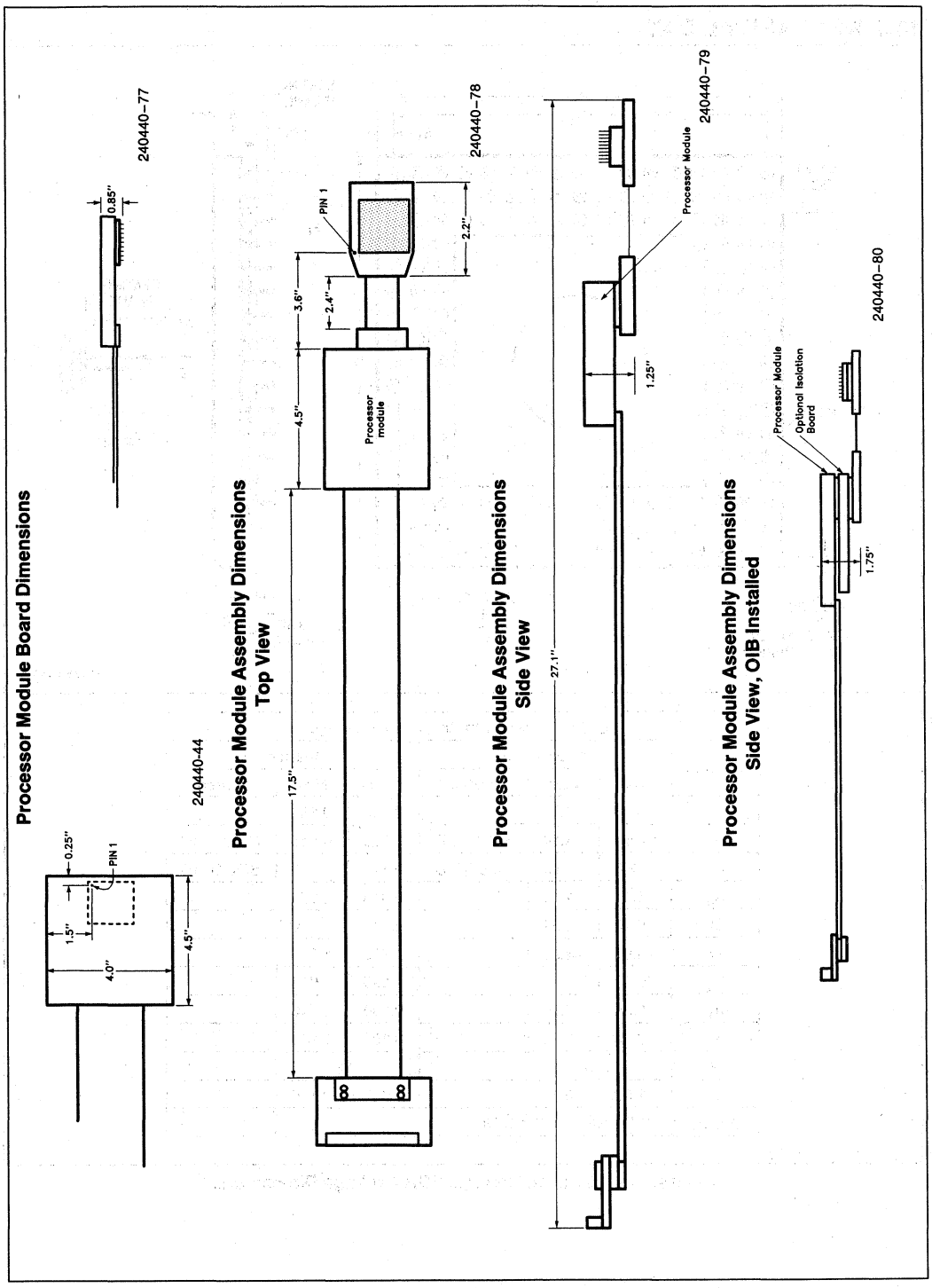
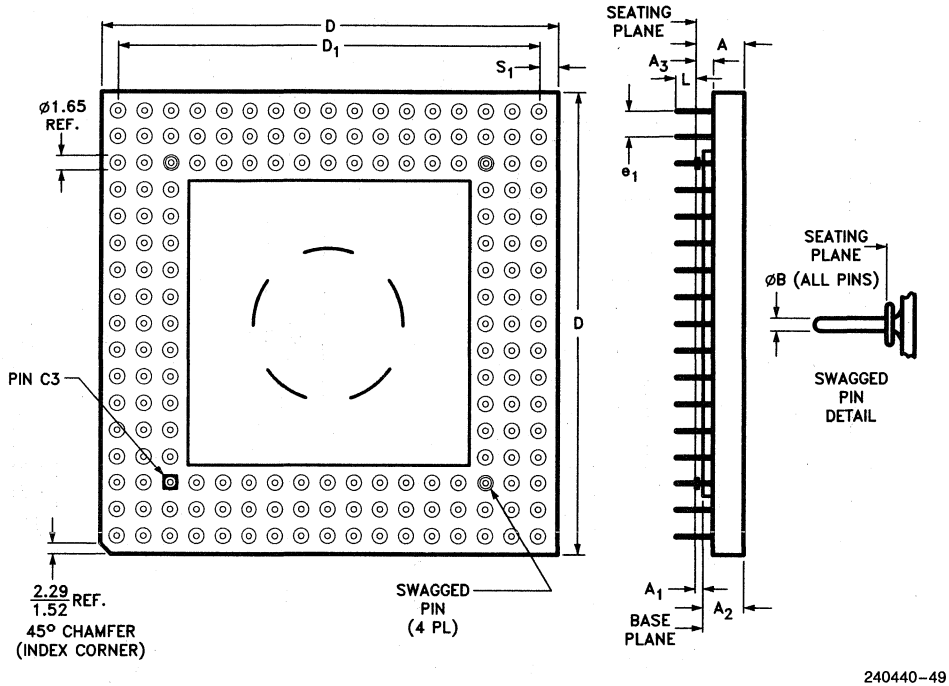


Figure 12.4c. ICD-486™ Probe Dimensions

13.0 MECHANICAL DATA



Family: Ceramic Pin Grid Array Package						
Symbol	Millimeters			Inches		
	Min	Max	Notes	Min	Max	Notes
A	3.56	4.57		0.140	0.180	
A_1	0.64	1.14	SOLID LID	0.025	0.045	SOLID LID
A_2	2.8	3.5	SOLID LID	0.110	0.140	SOLID LID
A_3	1.14	1.40		0.045	0.055	
B	0.43	0.51		0.017	0.020	
D	44.07	44.83		1.735	1.765	
D_1	40.51	40.77		1.595	1.605	
e_1	2.29	2.79		0.090	0.110	
L	2.54	3.30		0.100	0.130	
N	168			168		
S_1	1.52	2.54		0.060	0.100	
ISSUE	IWS REV X 7/15/88					

Figure 13.1. 168 Lead Ceramic PGA Package Dimensions

Table 13.1 Ceramic PGA Package Dimension Symbols

Letter or Symbol	Description of Dimensions
A	Distance from seating plane to highest point of body
A_1	Distance between seating plane and base plane (lid)
A_2	Distance from base plane to highest point of body
A_3	Distance from seating plane to bottom of body
B	Diameter of terminal lead pin
D	Largest overall package dimension of length
D_1	A body length dimension, outer lead center to outer lead center
e_1	Linear spacing between true lead position centerlines
L	Distance from seating plane to end of lead
S_1	Other body dimension, outer lead center to edge of body

NOTES:
1. Controlling dimension: millimeter.
2. Dimension "e_1" ("e") is non-cumulative.
3. Seating plane (standoff) is defined by P.C. board hole size: 0.0415–0.0430 inch.
4. Dimensions "B", "B_1" and "C" are nominal.
5. Details of Pin 1 identifier are optional.

13.1 Package Thermal Specifications

The 486 microprocessor is specified for operation when T_C (the case temperature) is within the range of 0°C–85°C. T_C may be measured in any environment to determine whether the 486 microprocessor is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

The ambient temperature (T_A) is guaranteed as long as T_C is not violated. The ambient temperature can be calculated from θ_{JC} and θ_{JA} from the following equations.

$$T_J = T_C + P * \theta_{JC}$$
$$T_A = T_J + P * \theta_{JA}$$
$$T_C = T_A + P * [\theta_{JA} - \theta_{JC}]$$

where T_J, T_A, T_C = Junction, Ambient and Case Temperature respectively. θ_{JC}, θ_{JA} = Junction-to-Case and Junction-to-Ambient Thermal Resistance, respectively.

P = Maximum Power Consumption

The values for θ_{JA} and θ_{JC} are given in Table 13.2 for the 1.75 sq. in., 168-pin, ceramic PGA.

Table 13.3 shows the T_A allowable (without exceeding T_C) at various airflows and operating frequencies (f_{CLK}).

Note that T_A is greatly improved by attaching "fins" or a "heat sink" to the package. P (the maximum power consumption) is calculated by using the maximum I_{CC} at 5V as tabulated in the *DC Characteristics* of Section 12.

Table 13.2. Thermal Resistance (°C/W) θ_{JC} and θ_{JA}

	θ_{JC}	θ_{JA} vs Airflow—ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
With Heat Sink*	2.0	13	8.0	6.0	5.0	4.5	4.25
Without Heat Sink	1.5	17	14.5	12.5	11.0	10.0	9.5

*0.350" high unidirectional heat sink (Al alloy 6063, 40 mil fin width, 155 mil center-to-center fin spacing).

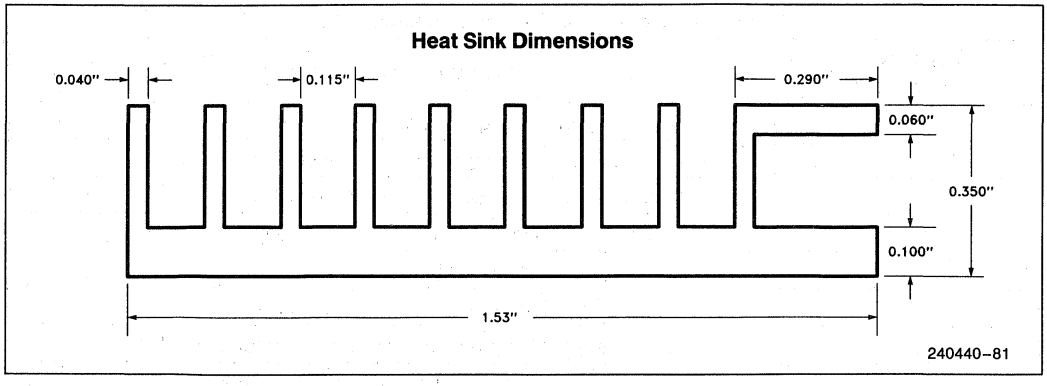


Table 13.3. Maximum T_A at Various Airflows

In °C

		Airflow-ft/min (m/sec)					
	f_{CLK} (MHz)	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
T_A with Heat Sink	25.0	47	64	71	75	76	77
	33.3	36	58	67	72	74	75
T_A without Heat Sink	25.0	31	40	47	52	55	57
	33.3	15	27	36	42	47	49

14.0 SUGGESTED SOURCES FOR i486 ACCESSORIES

Following are some suggested sources of accessories for the i486. They are not an endorsement of any kind, nor a warranty of the performance of any of the listed products and/or companies.

Sockets

1. McKenzie Technology
   44370 Old Palmspring Blvd.
   Fremont, CA 94538
   Tel: (415) 651-2700
2. E-CAM Technology, Inc.
   14455 North Hayden Rd.
   Suite 208
   Scottsdale, AZ 85260
   Tel: (602) 443-1949
3. Augat Inc. (for sockets with decaps)
   Interconnection Products Group
   33 Perry Ave.
   P.O. Box 779
   Attleboro, MA 02703
   Tel: (508) 222-2202

Heat Sinks/Fins

1. Thermalloy Inc.
   2021 West Valley View Lane
   Dallas, TX 75381-0839
   Tel: (214) 243-4321
2. E G & G Division
   60 Audubon Road
   Wakefield, MA 01880
   Tel: (617) 245-5900

TTL Crystals/Oscillators

1. NFL Frequency Controls, Inc.
   357 Beloit Street
   Burlington, WI 53105
   Tel: (414) 763-3591
2. M-Tron
   P.O. Box 630
   Yankton, SD 57078
   Tel: (605) 665-9321

Debugging Tower

1. Emulation Technology
   2344 Walsh Ave., Building F
   Santa Clara, CA 95051
   Tel: (408) 982-0664

15.0 REVISION HISTORY

Revision -003 of the i486 CPU data sheet contains many updates and improvements to the original version. A revision summary of major changes is listed below:

The sections significantly revised since version -001 are:

Section 2.1.2 The polarity and names of the two cache control bits in Control Register 0 (CR0) have been modified. The Cache Enable (CE) and Writes Transparent (WR) have been renamed Cache Disable (CD) and Not Write Through (NW). The value of CR0 after RESET has been changed to reflect the polarity change.

Section 6.2.15 The discussion of A20M# has been clarified. During the falling edge of RESET, A20M# should be high, for proper operation of the CPU.

Section 6.5 The value of CR0 after RESET has been modified.

Section 6.5.1 Figure 6.3, "Pin State during RESET" is added. This Figure is a general reference for Reset issues. Previous Figures 8.1, 8.2, and 8.8 have been deleted, since Figure 6.3 now contains Reset information.

Section 7.2.10 A discussion of addresses and byte enables driven during INTA cycles has been added.

Section 10.1 Clock counts and opcodes have been clarified and corrected.

Section 10.1 The opcode slot for CMPXCHG instruction has been moved from 0FA6/A7 to 0FB0/B1.

Section 12.2 Table 12.1 has been enhanced. The "Case Temperature under Bias" spec was improved. The "Supply Voltage with Respect to V_{SS}" spec was added.

Section 12.3 Maximum I_{CC} values have been improved to 700 mA at 25 MHz and 900 mA at 33 MHz.

Section 12.3 Typical I_{CC} values have been modified to 550 mA at 25 MHz and 700 mA at 33 MHz.

Section 12.3 C_{IN}, C_O, and C_{CLK} values have been changed to 20 pF. Testing parameters and Note 7 were added.

Section 12.4 The A.C. Specifications have been improved. Float delays were improved at both 25 MHz and 33 MHz. Note 1 was added to the float delays. Maximum valid delays were reduced at 33 MHz.

Section 12.5 The ICD section was enhanced.

Section 13.1 Thermal resistance θ_{CA} values of the 168-pin ceramic package have been corrected.

Section 13.1 Maximum ambient temperatures have been corrected to use the max I_{CC} values.

The sections significantly revised since version -002 are:

2.1.2.1 Spec change for PCD and PWT bits.

Table 2.16 Value of Intel Reserved Interrupt Vector assignment corrected to '18-31'.

Section 3.1 Added CMPCHG, XADD instructions in the table.

Section 3.5 Added explanation about NMI not able to bring out the processor from shutdown under certain conditions.

Section 4.4.6 Value of task switching time corrected to 10 ms.

Section 4.5.4 Specification change for PCD and PWT bits.

Section 5.6 Specification change for PCD and PWT bits.

Section 5.7 Cache flushing procedure explained, when FLUSH# applied synchronously or asynchronously.

Section 6.2.5 Specification change for PLOCK cycle.

Section 6.2.8 Added explanation for warm boot-up.

Section 6.2.12 Specification change for PCD and PWT bits.

Section 6.2.13 Explanation added for FERR# behavior.

Section 6.2.14 Explanation added for IGNNE# behavior.

Section 6.2.15 Explanation added for A20M# behavior in protected mode and during RESET.

Section 6.3 Simplified example for read reordering in write buffers.

Section 6.3.1 Corrected REP OUTS instruction.

Section 6.3.2 Added explanation about cache update on read-modify-write cycle.

Section 6.5 Added RESET pulse length requirement with or without BIST

Section 6.5 Added table for i486 revision ID.

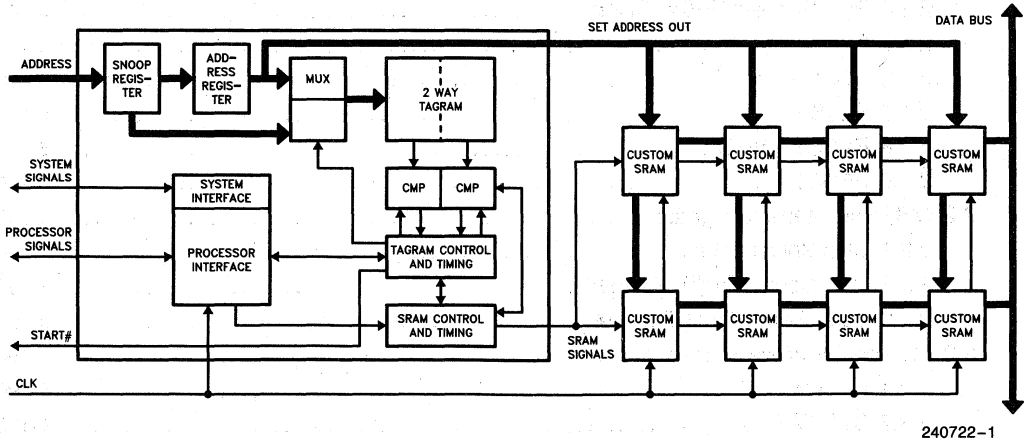
Table 6.2	Corrected CR0 value after Reset.
Figure 6.3	Corrected pin state diagram during RESET. RESET pulse length changed to 15 CLKs.
Section 7.2.2.3	Added explanation to terminate burst cycle.
Section 7.2.3.4	Clarified text on changing KEN# during cache line fill.
Figure 7.12	Corrected timing diagram to show A4–A31, M/IO#, D/C#, W/R# do not change during burst.
Figure 7.13	Corrected timing diagram to show A4–A31, M/IO#, D/C#, W/R# do not change during burst.
Figure 7.14	Corrected timing diagram to show A4–A31, M/IO#, D/C#, W/R# do not change during burst.
Section 7.2.4.2	Added cases that follow burst order.
Section 7.2.6	Added explanation for read-modify-write for un-aligned transfers.
Section 7.2.7	HOLD latency decreased by providing window in PLOCK cycle (specification change).
Section 7.2.8	Added explanation about EADS# timing.
Section 7.2.8	Added the case of invalidation with BOFF or HOLD.
Figure 7.22	Change in Timing Diagram for BREQ.
Figure 7.23	Change in Timing Diagram for BREQ.
Figure 7.25	Change in Timing Diagram for RDY#/BRDY#.
Section 7.2.9	Added explanation about HOLD getting recognized during un-aligned writes.
Section 7.2.11	Added status of address and data busses during special bus cycles.
Section 7.2.11	Added sections on Halt and Shut-down cycles.
Figure 7.30	Corrected state diagram by ANDing BRDY# and BLAST# for the last transfer of the burst cycle.
Section 7.2.14	Difference in FERR# and ERROR# explained.
Section 8.1	Changed Reset width to 15 CLKs.
Section 8.4	Added explanation on tri-state status.
Table 10.1	Corrected value in format.
Section 11.0	Added Note 6 on FERR# and ERROR# difference.
Section 11.0	Added TLB replacement algorithm for 386 DX.
Section 12.3	Corrected values in Note 2.
Section 12.3	Added "internal" for pullup and pull-down resistors.
Figure 12.2 & Figure 12.3	Waveforms for input and output signals have been re-drawn to show details about set-up, hold and float times.
Section 13.1	Added details about T_A calculation from θ_{JC} and θ_{JA}.
Section 14.0	Added new section on suggested sources of i486 accessories like sockets, debugging tower, heat sinks, etc.

485TURBOCACHE MODULE i486™ MICROPROCESSOR CACHE UPGRADE

82485MA (64k Module)
82485MB (128k Module)

- **High Performance**
  - **Zero Waitstate Access**
  - **One Clock Bursting**
  - **Two-Way Set Associative**
  - **BIOS ROM Cacheing**
  - **25/33 MHz Operation**
- **Range Of Price/Performance**
  - **0, 64k, 128k Cache With Single Socket**
  - **Cascadable With Multiple Sockets**
- **High Integration**
  - **Seven Square Inch Area**
  - **Includes Tag, Data, Parity, and Controller**
- **Easy To Use**
  - **Software Transparent**
  - **End User/Dealer Installation**
  - **Write-Through Memory Update**
  - **Same Timing as i486™ CPU**
  - **Same Invalidation Mechanism as i486 CPU**

The 485Turbocache Module is a performance upgrade for 25 MHz or 33 MHz i486™ Microprocessor systems. It provides up to 128k bytes of external cache memory in a single, end-user installable module. Support for the cache module upgrade is provided by a 113 pin socket in the i486 CPU system. A single socket allows three price/performance configurations: no cache, a 64k byte cache, or 128k byte cache. Additional modules may be cascaded for larger cache sizes. No jumpers, configuration software, or BIOS/applications/operating system support is required to get 5-30% (15% average) performance boost after installing the cache. Cache data integrity is monitored by a parity bit per byte.



240722-1

Figure 0.1. 485Turbocache Module Internal Block Diagram

5

i486 and 486 are trademarks of Intel Corporation.

November 1990
Order Number: 240722-002

CONTENTS

PAGE

0.1 PINOUT 5-180

0.2 PIN DESCRIPTION OVERVIEW 5-181

1.0 FUNCTIONAL DESCRIPTION 5-183
1.1 Introduction 5-183
1.2 Base Architecture 5-184
1.3 Cache Operation 5-184
1.3.1 Read Miss 5-184
1.3.2 Read Hit 5-185
1.3.3 Write Cycles 5-185
1.3.4 Invalidation Cycles 5-185
1.3.5 BOFF# Cycles 5-186
1.4 Incompatibilities 5-187

2.0 SYSTEM INTERFACE 5-187
2.1 i486 Microprocessor Signals 5-188
2.1.1 Address Lines A2-A31 5-188
2.1.2 Data Lines D0-D31 and Parity DP0-DP3 5-188
2.1.3 ADS#, W/R#, M/IO# 5-188
2.1.4 Byte Enables BE0#-BE3# 5-188
2.1.5 BLAST# 5-189
2.1.6 BOFF# 5-189
2.1.7 FLUSH# 5-189
2.1.8 EADS#, AHOLD 5-189
2.1.9 RESET 5-189
2.2 CPU Bus Interface Signals 5-189
2.2.1 Chip Select CS# 5-189
2.2.2 CPU Cache Enable CKEN# 5-190
2.2.3 Burst Ready Out BRDYO# 5-190
2.3 Memory Interface Signals 5-190
2.3.1 PRSN# 5-190
2.3.2 START# 5-190
2.3.3 Write Protect WP 5-190
2.3.4 Write Protect Strapping Option WPSTRP# 5-190
2.3.5 System Cache Enable SKEN# 5-190
2.3.6 Cache Ready and Burst Ready CRDY#, CBRDY# 5-191

CONTENTS

PAGE

3.0 SYSTEM CONFIGURATIONS 5-191
3.1 Single Cache 5-191
3.1.1 i486™ Microprocessor Bus Interface 5-191
3.1.2 Memory Bus Interface 5-191
3.1.3 KEN# and SKEN# Generation 5-191
3.1.4 START# Generation 5-191
3.2 Multiple Cache 5-192
3.2.1 Memory Bus Interface 5-192
3.2.2 START# 5-192
3.2.3 KEN# 5-192
3.2.4 SKEN# 5-192
3.2.5 CS# 5-193
3.3 Optional Cache 5-193
3.3.1 Signal Considerations: START#, CKEN#, BRDYO# 5-193
3.3.2 Considerations With Multiple Caches 5-194

4.0 OPERATIONAL/PERFORMANCE CONSIDERATIONS 5-194
4.1 Testing and Data Integrity 5-194
4.2 Sectored vs Non-Sectored Cache 5-194
4.3 Performance Considerations 5-195
4.3.1 SKEN# Assertion 5-195
4.3.2 Invalidation Window 5-195
4.3.3 BOFF# Assertion 5-195
4.3.4 START# Predictability 5-196

5.0 MECHANICAL SPECIFICATIONS 5-198

6.0 ABSOLUTE MAXIMUM RATINGS 5-199

7.0 D.C. CHARACTERISTICS 5-199

8.0 A.C. CHARACTERISTICS 5-200

9.0 WAVEFORMS 5-202

10.0 REVISION HISTORY 5-205

0.1 PINOUT

	1	2	3	4	5
A	GND	RESET		CS#	GND
B	CLK	M/IO#		CRDY#	CKEN#
C	RESV	FLUSH#		CBRDY#	BRDYO#
D	BLAST#	EADS#		V_{CC}	SKEN#
E	BOFF#	V_{CC}		WP	START#
F	ADS#	W/R#		D0	GND
G	GND	WPSTRP#		D2	D1
H	BE0#	BE1#		GND	D3
I	BE2#	BE3#		D5	D4
J	A2	GND		D7	D6
K	V_{CC}	A3		D8	GND
L	A4	A5		D10	D9
M	A6	A7		V_{CC}	D11
N	A9	A8		D13	D12
O	A10	V_{CC}		D15	D14
P	GND	A11		DP0	GND
Q	A31	A12		D16	DP1
R	A14	A13		GND	D17
S	A15	GND		D19	D18
T	A17	A16		D21	D20
U	A19	A18		D22	V_{CC}
V	V_{CC}	A20		D24	D23
W	A22	A21		GND	D25
X	A23	V_{CC}		D27	D26
Y	A25	A24		D29	D28
Z	A27	A26		D30	D31
AA	A29	A28		DP2	DP3
BB	GND	A30	PRSN#	V_{CC}	GND

240722-2

Top Side View

	1	2	3	4	5
BB	GND	A30	PRSN#	V_{CC}	GND
AA	A29	A28		DP2	DP3
Z	A27	A26		D30	D31
Y	A25	A24		D29	D28
X	A23	V_{CC}		D27	D26
W	A22	A21		GND	D25
V	V_{CC}	A20		D24	D23
U	A19	A18		D22	V_{CC}
T	A17	A16		D21	D20
S	A15	GND		D19	D18
R	A14	A13		GND	D17
Q	A31	A12		D16	DP1
P	GND	A11		DP0	GND
O	A10	V_{CC}		D15	D14
N	A9	A8		D13	D12
M	A6	A7		V_{CC}	D11
L	A4	A5		D10	D9
K	V_{CC}	A3		D8	GND
J	A2	GND		D7	D6
I	BE2#	BE3#		D5	D4
H	BE0#	BE1#		GND	D3
G	GND	WPSTRP#		D2	D1
F	ADS#	W/R#		D0	GND
E	BOFF#	V_{CC}		WP	START#
D	BLAST#	EADS#		V_{CC}	SKEN#
C	RESV	FLUSH#		CBRDY#	BRDYO#
B	CLK	M/IO#		CRDY#	CKEN#
A	GND	RESET		CS#	GND

240722-3

Pin Side View

Pin Cross Reference

Pin Name	Location	Pin Name	Location	Pin Name	Location	Pin Name	Location	Pin Name	Location
ADS#	F1	A20	V2	CBRDY#	C4	D10	L4	D29	YA
A2	J1	A21	W2	CKEN#	B5	D11	M5	D30	Z4
A3	K2	A22	W1	CLK	B1	D12	N5	D31	Z5
A4	L1	A23	X1	CRDY#	B4	D13	N4	EADS#	D2
A5	L2	A24	Y2	CS#	A4	D14	O5	FLUSH#	C2
A6	M1	A25	Y1	DP0	P4	D15	O4	M/IO#	B2
A7	M2	A26	Z2	DP1	Q5	D16	Q4	PRSN#	BB3
A8	N2	A27	Z1	DP2	AA4	D17	R5	RESET	A2
A9	N1	A28	AA2	DP3	AA5	D18	S5	RESV	C1
A10	O1	A29	AA1	D0	F4	D19	S4	SKEN#	D5
A11	P2	A30	BB2	D1	G5	D20	T5	START#	E5
A12	Q2	A31	Q1	D2	G4	D21	T4	WP	E4
A13	R2	BE0#	H1	D3	H5	D22	U4	WPSTRP#	G2
A14	R1	BE1#	H2	D4	I5	D23	V5	W/R#	F2
A15	S1	BE2#	I1	D5	I4	D24	V4	GND	A1, G1, P1, BB1,
A16	T2	BE3#	I2	D6	J5	D25	W5		J2, S2, H4, R4, BB5,
A17	T1	BLAST#	D1	D7	J4	D26	X5		W4, A5, F5, K5, P5,
A18	U2	BOFF#	E1	D8	K4	D27	X4	V_{CC}	K1, V1, E2, O2, X2,
A19	U1	BRDYO#	C5	D9	L5	D28	Y5		D4, M4, BB4, U5

Figure 0.2. 485Turbocache Module 64k/128k Pin Configuration

0.2 PIN DESCRIPTION OVERVIEW

Pin Name	Type	Active	Description
CONTROL SIGNALS			
CLK	I	–	**CLOCK** is the timing reference from which the 485Turbocache Module monitors and generates events. CLK must be connected to the i486 CPU CLK pin.
RESET	I	High	**RESET CACHE** forces the 485Turbocache Module to begin execution in a known state and must be connected to the i486 CPU RESET pin. It also causes all cache lines to be invalidated. Setup and hold times t_{23} and t_{24} must be met for recognition in any specific clock.
ADS#	I	Low	**ADDRESS STROBE** is generated by the i486 Microprocessor. It is used to determine that a new cycle has been started. Setup time t_7 must be met for proper operation.
M/IO#	I	–	**MEMORY/IO** is an i486 CPU generated cycle definition signal that indicates a Memory (M/IO# high) or I/O (M/IO# low) access. Setup time t_7 must be met for proper operation.
W/R#	I	–	**WRITE/READ** is an i486 CPU generated cycle definition signal used to indicate a Write (W/R# high) or Read (W/R# low) access. Setup time t_7 must be met for proper operation.
START#	O	Low	**MEMORY START** indicates that a cache read miss or a write has occurred and that the current access must be serviced by the memory system. START# is not activated for I/O cycles, and is not asserted if CS# is inactive.
BRDYO#	O	Low	**BURST READY OUT** is a burst ready signal driven by the 485Turbocache Module to the i486 CPU. It is activated when a read hit occurs to the 485Turbocache Module and should be a term in the BRDY# input to the i486 CPU.
CBRDY#	I	Low	**CACHE BURST READY IN** is the burst ready input from the memory system. It is applied to both the 485Turbocache Module and the i486 CPU BRDY# pin in parallel. CBRDY# is ignored during T1 and idle cycles. BLAST# determines the length of the transfer. All cacheable read cycles are 4 dword transfers. Setup and hold times t_9 and t_{10} must be met for proper operation.
CRDY#	I	Low	**CACHE READY IN** is the non-burst ready input from the system. Like CBRDY#, it is applied to both the cache and i486 CPU RDY# pin in parallel. CRDY# is ignored during T1 and idle cycles. Setup and hold times t_9 and t_{10} must be met for proper operation.
BLAST#	I	Low	**BURST LAST** is output by the i486 CPU and is sampled by the 485Turbocache Module to determine when the end of a cycle occurs. Setup and hold times t_8 and t_{8a} must be met for proper operation.
BOFF#	I	Low	**BACKOFF** is an i486 CPU input sampled by the 485Turbocache Module to indicate that a cycle be immediately terminated. If BOFF# is sampled active, the 485Turbocache Module will float its data bus. The 485Turbocache Module will ignore all cycles, except invalidation cycles, until BOFF# is deactivated. Setup and hold times t_{17} and t_{18} must be met for proper operation.
PRSN#	O	Low	**PRESENCE** is an active low output always asserted by the 485Turbocache Module. It may be used as a 485Turbocache Module presence indicator and should be connected via a 10K pullup resistor.

0.2 PIN DESCRIPTION OVERVIEW (Continued)

Pin Name	Type	Active	Description
ADDRESS SIGNALS			
A2-A31	I	–	**PROCESSOR ADDRESS LINES A2-A31** are the i486 CPU address lines used by the 485Turbocache Module. Address lines A2 and A3 are used as burst address bits. In the 64k 485Turbocache Module, A4-A14 comprise the set address inputs to the 485Turbocache Module and A15-A31 are used as the tag address. In the 128k 485Turbocache Module, A4 becomes a line select input, A5-A15 is the set address input and A16-A31 is used as the tag address. Setup time t_6 must be met for proper operation.
BE0#-BE3#	I	Low	**BYTE ENABLE** inputs are connected to the i486 CPU byte enable outputs. They are specifically used for completing partial reads from and writes to the 485Turbocache Module during hit cycles. During miss cycles, transfers are ignored if all the byte enables are not asserted since the 485Turbocache Module only caches 32-bit transfers. Setup time t_6 must be met for proper operation.
CS#	I	Low	**CHIP SELECT** is used to cascade 485Turbocache Module modules. Address bits may be decoded in order to cascade multiple devices or be decoded to selectively cache portions of memory. Setup and hold times t_{30} and t_{31} must be met for proper operation.
DATA SIGNALS			
D0-D31	I/O	–	**PROCESSOR DATA LINES D0-D31** are connected to the i486 CPU data bus. D0-D7 define the least significant byte while D24-D31 define the most significant byte. Setup and hold times t_{13} and t_{14} must be met for proper operation.
DP0-DP3	I/O	–	**DATA PARITY** are the parity bits associated with the data on the data bus. They are connected to the i486 CPU pins with the same name. Parity is treated by the 485Turbocache Module as additional data bits to be stored. Setup and hold times t_{13} and t_{14} must be met for proper operation.
CACHEABILITY SIGNALS			
CKEN#	O	Low	**CACHE ENABLE TO CPU** is the KEN# term generated by the 485Turbocache Module to the i486 Microprocessor. CKEN# is activated twice; First during T1 to enable a cache line fill, and second on the clock before the last BRDY# or RDY# to validate the line fill. CKEN# is ALWAYS active in T1, but will not validate a line fill if the line fill is a write protected line and WPSTRP# is low, or if the cycle is a read miss.
SKEN#	I	Low	**SYSTEM CACHE ENABLE** is an input from the main memory system to indicate whether the current line fill is cacheable in the 485Turbocache Module. It is sampled by the 485Turbocache Module exactly like KEN# is sampled by the i486 Microprocessor. Setup and hold times t_{11} and t_{12} must be met for proper operation.
FLUSH#	I	Low	**FLUSH CACHE** causes the 485Turbocache Module to invalidate its entire cache contents regardless of CS#. Any line fill in progress will continue, but will be invalidated immediately. The i486 CPU flush instruction does not affect the 485Turbocache Module. Setup and hold times t_{23} and t_{24} must be met for recognition in any specific clock.

0.2 PIN DESCRIPTION OVERVIEW (Continued)

Pin Name	Type	Active	Description
CACHEABILITY SIGNALS (Continued)			
WP	I	High	**WRITE PROTECT** defines a line as write protected. WP is sampled during the third transfer of a line fill and is maintained internally as a state bit. Any subsequent writes to a write protected line will have no effect. Setup and hold times t_{15} and t_{16} must be met for proper operation.
WPSTRP#	I	Low	**WRITE PROTECT STRAPPING OPTION** changes the behavior of CKEN#. CKEN# is always asserted in T1 to indicate a cacheable line transfer but is deasserted on the next clock. During read hit cycles, CKEN# is asserted again for the duration of the transfer to indicate a cacheable line fill. If WPSTRP# is strapped low, and a write protected line is being transferred, CKEN# is not activated again for the transfer. This prevents the i486 CPU from cacheing write protected lines during read hit cycles. WPSTRP# must be valid and not change two clocks before and after the falling edge of RESET.
INVALIDATE SIGNALS			
EADS#	I	Low	**VALID EXTERNAL ADDRESS STROBE** indicates that an invalidation address is present on the i486 CPU address bus. The 485Turbocache Module will invalidate this address, if present, but will only do so if CS# is active. The 485Turbocache Module is capable of accepting an EADS# every other clock. The 485Turbocache Module EADS# should be connected to the i486 CPU EADS# pin. Setup and hold times t_{19} and t_{20} must be met for proper operation.

1.0 FUNCTIONAL DESCRIPTION

1.1 Introduction

The 485Turbocache Module is a complete 2-way set-associative 64k or 128k cache housed in a 113-pin module. It contains 4 or 8 custom data SRAMs and the Intel 82485 cache controller. The cache module was designed to be cascadable to a maximum of 512k with the addition of more modules. The module was also designed so the system may easily detect a cache's presence and reconfigure itself accordingly. The 485Turbocache Module is a plug-in option that is an ideal i486™ Microprocessor cache solution.

The cache module interfaces directly to the i486 Microprocessor. Designing with the cache module is easy because it directly supports the timing of 25 MHz and 33 MHz systems. It is capable of reading and writing data in 0 waitstates, and performing 1 clock bursting. Because the 485Turbocache Module was designed exclusively for the i486 Microprocessor, it recognizes i486 CPU invalidations, use of BOFF#, and prematurely terminated cycles. The cache module is write-through so it supports the same i486 CPU consistency mechanisms, stores data parity, can cache BIOS in modes where the i486 CPU cannot, is software transparent, and may be an end-user installable upgrade.

Below are the order codes for the 485Turbocache Module:

Size	25 MHz	33 MHz
64k	82485MA-25	82485MA-33
128k	82485MB-25	82485MB-33

The following Functional Description describes the cache module's base architecture, its operation, features, and deviations from the i486 CPU specification.

1.2 Base Architecture

The 485Turbocache Module contains an 82485 cache controller and 4 (82485MA) or 8 (82485MB) SRAMs for a complete 64k or 128k cache. In either configuration, the 485Turbocache Module is 2-way set-associative with a 16 byte line size.

Figure 1.1 outlines the 82485 cache controller which is the heart of the 485Turbocache Module. Each WAY contains 2k tags with 17 bits per tag so it may store the complete 4G real address space. The tags also reference 2 valid bits and a write-protect bit. When the 82485 is configured as a 64k cache, as in the 64k 485Turbocache Module, each tag references a single, 16 byte line. When the 82485 is configured as a 128k cache, as in the 128k 485Turbocache Module, each tag is forced to reference two consecutive 16 byte lines; This is called sectoring. A 128k 485Turbocache Module contains 2 sectors per tag. The LS input (address bit A4) determines which sector of each tag is being selected.

The control units of the 82485 are responsible for three main functions: controlling the data SRAMs, controlling the tagram structure, and interfacing to the i486 CPU. Since these are independent units, the 82485 is capable of updating its tagram while data is being bursted into SRAM, or invalidating during a line fill to a different address. Special address registers in the 485Turbocache Module allow the i486 Microprocessor to drop its address in the first T2 (in response to AHOLD) and the system to issue a invalidate address with an i486 CPU hold time.

The 82485 uses the "Least Recently Used" algorithm to determine which tag should be invalidated on cache misses. A single LRU bit per tag is used to point to the tag that will be replaced should a replacement be required.

The data memory portion of the 485Turbocache Module is composed of a set of SRAMs that operate at fast 33 MHz speeds. They are capable of 0 wait-state reads and writes, and single clock bursting, and have minimized capacitive loading on the i486 CPU clock and data lines.

1.3 Cache Operation

To operate at high speeds, the 485Turbocache Module must begin its tag lookup to determine a cache hit or miss as soon as possible. During normal operation, this is done as soon as the i486 CPU generates an address. SRAM reads, SRAM writes, and system signals cannot be generated until a hit or miss has been determined. The following sections will discuss read miss, read hit, write, invalidate, and BOFF# cycles.

1.3.1 READ MISS

Figure 1.2 shows 485Turbocache Module activity during a normal read miss cycles. In T1, the 485Turbocache Module begins its tag lookup to see if the read cycle is a hit. Once it has been determined that the address is not present in the cache (a miss), START# is issued to indicate to the memory system that it must service the current cycle. The cache is then idle until SKEN#, the cache's KEN# input, is seen active. Should SKEN# be inactive and the burst line transfer from memory begin, the line is non-cacheable and is ignored.

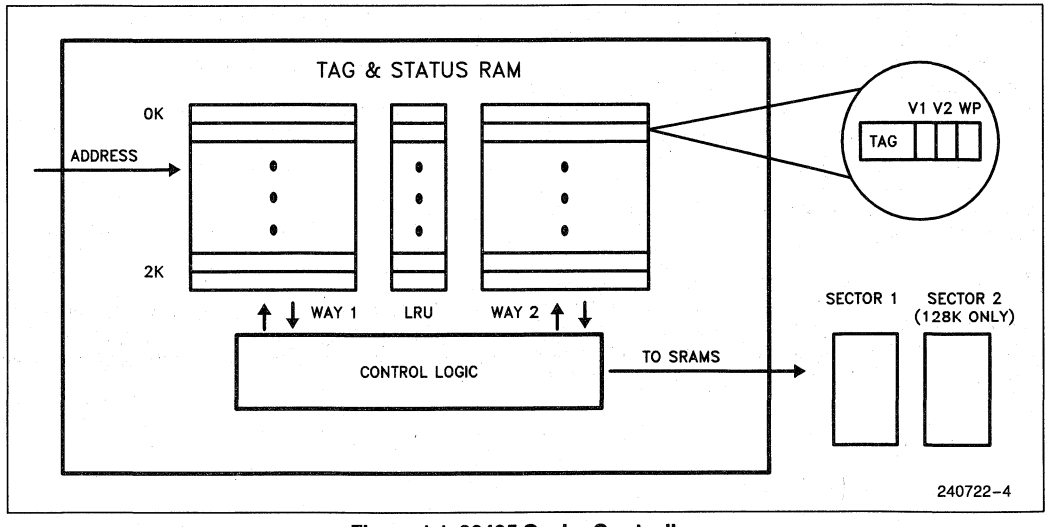


Figure 1.1. 82485 Cache Controller

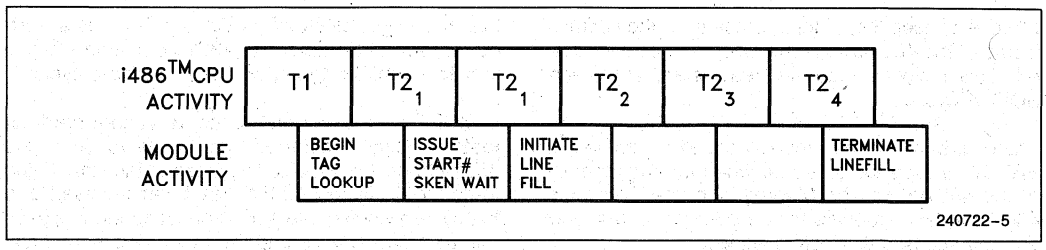


Figure 1.2. Normal Read Miss Cycle

Once SKEN# has been asserted, the 485Turbocache Module invalidates a line in the cache (or chooses a free line) in preparation for the bursted data (see section 4.3.1). The data is bursted into the cache and back to the i486 Microprocessor simultaneously. If an SKEN# preceded the last bursted item, then the line was cacheable, and the 485Turbocache Module updates its valid bit to indicate so. If the line is invalid, or aborted for any reason (BLAST#, BOFF#) the line is left invalid.

During a read miss cycle, the 485Turbocache Module cannot accept the data from memory in zero waitstates. The earliest data may be returned is the clock after START# is sampled active. START# is the signal that indicates that the memory system must complete the current cycle.

The 485Turbocache Module is also capable of handling non-burst and interrupted burst line fills. Refer to the section "4.0. Performance Considerations" for improving 485Turbocache Module performance during line fills. Note that the 485Turbocache Module only caches 32-bit transfers. The 485Turbocache Module does not input the i486 CPU inputs BS#8 or BS#16. All transfers are assumed to be 32-bit transfers with valid data on all 32 data lines.

1.3.2 READ HIT

During Read Hit cycles, the 485Turbocache Module responds directly to the i486 Microprocessor with a line of data in 5 clocks. The 485Turbocache Module asserts CKEN# (its KEN# output to the i486 CPU) in both T1 and the third T2 to indicate this as a cacheable transfer. Should the bursted line be write-protected, AND WPSTRP# is strapped low, CKEN# is high for the third T2 and the line is not cached by the i486 CPU. The only updating the 485Turbocache Module needs to perform during read hit cycles is to update the LRU bit to point to the WAY that was not transferred.

1.3.3 WRITE CYCLES

Since the 485Turbocache Module is a write-through cache, all write cycles are written by the i486 CPU to main memory. Figure 1.3 shows a write hit where the tag lookup in T1 is found to be a hit so the data is updated by the cache in T2. Write misses do not affect cache contents, nor do writes to write protected lines. Write hits will alter the LRU bit in the same way as a read hit.

5

1.3.4 INVALIDATION CYCLES

The 485Turbocache Module allows invalidation cycles to occur at any time by asserting AHOLD and EADS#. Self-invalidations, where AHOLD is not asserted, are allowed at any time except on the clock edge of the last transfer of a line fill. EADS# assertion allows both the CPU cache and 485Turbocache Module to be invalidated at the same time. Regardless of what the 485Turbocache Module is doing,

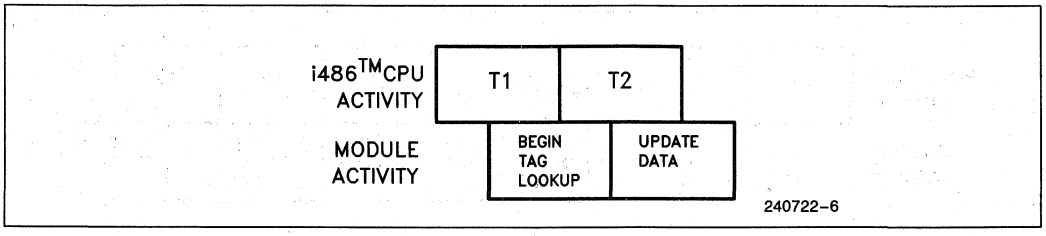


Figure 1.3. Write Hit Cycle

EADS# causes the address present on the address inputs of the 485Turbocache Module to be invalidated. This includes read hit, read miss, write, and BOFF# cycles.

There may be a performance penalty, however, if EADS# is asserted at a time when the tag memory of the 485Turbocache Module is in use. Since the 485Turbocache Module tags are single-ported, only one tag access per clock is allowed.

Figure 1.4 shows a read miss cycle with an invalidation lookup occurring in the third transfer of a line fill. Under normal conditions, the 485Turbocache Module would, on the next clock, validate the current line that is being filled. Since the EADS# occurred, the tagram is occupied on the next clock with a tag lookup to see if the invalidate is a hit, and the current line is not yet validated. If it is a hit, the next cycle is used to perform the actual invalidation. The following clock is spent validating the current line fill. Should the i486 Microprocessor begin a cycle immediately, the 485Turbocache Module is not able to perform its tag lookup until one clock cycle later when the tag memory is free. This causes START# to be delayed, and ultimately a memory read cycle from beginning.

For greatest performance, EADS# should not be issued in the second, third or fourth transfer of a cache line fill.

Self-Invalidations, EADS# asserted without AHOLD, are not allowed at the clock edge of the last T2 of a cycle (the first T1 clock edge of the next cycle). If a self-invalidation occurs in T1, ADS# and EADS# are sampled at the same time, the 485Turbocache Module will invalidate the line and assert START# as in a normal read miss cycle. If EADS# is asserted at any other time, START# is not asserted.

1.3.5 BOFF# CYCLES

When BOFF# is asserted, the 485Turbocache Module, like the i486 Microprocessor, will relinquish the bus in the next clock cycle. While BOFF# is asserted, as any other time, the 485Turbocache Module monitors EADS# to perform any invalidate cycles.

If BOFF# is asserted during a cache read hit (data is being transferred from cache to CPU), the 485Turbocache Module invalidates the line being transferred. Once BOFF# has been released and the cycle resumes, the 485Turbocache Module sees this as a cache miss and the memory system must supply the remaining data. If BOFF# is asserted during a cache read miss (memory is transferring to cache and CPU), the 485Turbocache Module will treat the line fill like an aborted fill, and the line will remain invalid. Once BOFF# is released and the cycle is restarted, the remainder of the line fill is treated like another aborted fill, and remains invalid.

Figure 1.5 is an example of an aborted line fill. Since the line transfer is interrupted before the transfer completes, it stays invalidated. Once the transfer resumes, the 485Turbocache Module sees a new cycle begin with ADS#, but it completes with BLAST# after three transfers. It treats this as an aborted line fill cycle, and the cycle is never validated.

Asserting BOFF# in the same clock as ADS# will cause the i486 CPU to float its bus in the next clock and leave ADS# floating low. Since ADS# is floating low, a peripheral device may think that a new bus cycle has begun even though the cycle was aborted. The 82485 handles this circumstance in most cases since an active ADS# in the clock BOFF# is deasserted is ignored. The only circumstance that must be handled by the system is as follows:

BOFF# is asserted in T1, and before BOFF# is deasserted, HOLD is asserted and remains asserted after BOFF# is deasserted (see Figure 1.6). In this circumstance it is necessary for the system to assure that ADS# is either driven to a valid level or pulled high in the clock after BOFF# is deasserted (meeting the 82485 ADS# setup time).

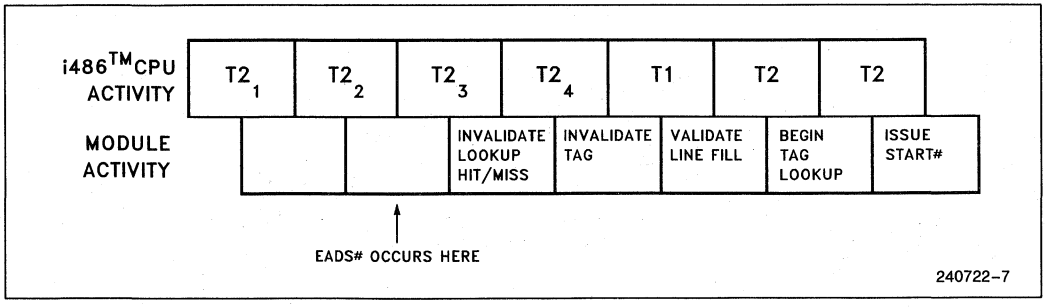


Figure 1.4. Invalidation During Read Miss

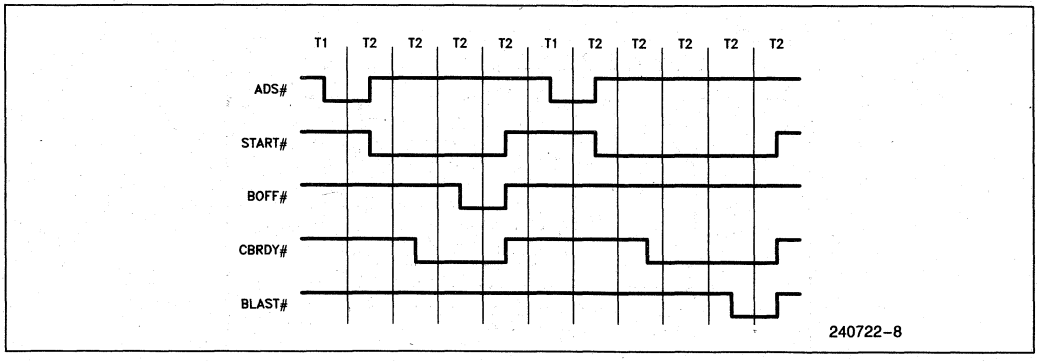


Figure 1.5. Aborted Line Fill

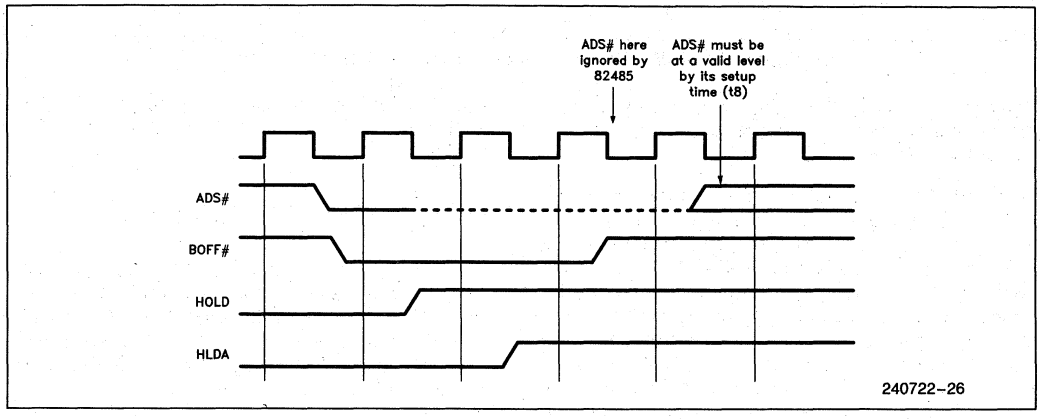


Figure 1.6. BOFF# Asserted in T1

There are several ways to avoid this system restriction:

1. Do not assert BOFF# in T1.
2. Use a "two clock" backoff: in the first clock AHOLD is asserted and in the second clock BOFF# is asserted. This guarantees that ADS# will not be floating low.
3. Do not assert HOLD when BOFF# is asserted.

1.4 Incompatibilities

Below are a list of some special design considerations that the 485Turbocache Module requires to be designed into an i486 CPU system. They have been summarized to point out any possible inconsistencies between the i486 CPU specification and the 485Turbocache Module specification:

1. Invalidation cycles may only be performed every two clocks. Unlike the i486 CPU, the 485Turbocache Module only allows EADS# assertion every other clock at most.
2. The minimum clock high voltage is slightly higher than the i486 CPU specification. It is still within TTL levels, however.
3. The i486 CPU will recognize HOLD during non-burst, non-cacheable, code prefetches. These prefetches are cacheable by the 485Turbocache Module. Since the module does not see the HLDA signal, another bus master could hold the CPU in mid-cycle, begin its own transfer, and coincidentally complete the cacheable transfer. This is only possible in systems that have another bus master that can drive the module's ADS pin. In these systems, the CPU's HLDA pin should be inverted and connected to the module's BOFF# input. This guarantees that cycles interrupted by HLDA will be aborted, and not cached, by the 485Turbocache Module.

2.0 SYSTEM INTERFACE

The following section describes the basic connection of the 485Turbocache Module in an i486 CPU

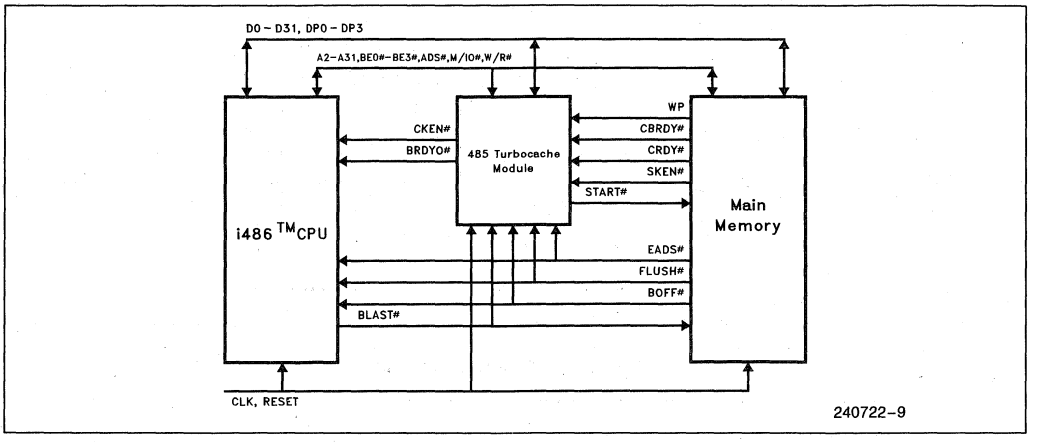


Figure 2.1. 485Turbocache Module Typical Configuration

system. The section highlights the CPU bus connections, memory bus connections, and gives specifics about their related signals.

A typical 485Turbocache Module connection to an i486 Microprocessor and memory subsystem is shown in Figure 2.1. All of the signals that the i486 CPU generate "feed-around" the 485Turbocache Module; That is, they go to both the 485Turbocache Module and the memory controller. In turn, most memory generated signals feed-around the 485Turbocache Module back to the CPU. This is what makes the 485Turbocache Module an optional cache. The following describes all the signals the 485Turbocache Module encounters.

2.1 i486™ Microprocessor Signals

The following 485Turbocache Module signals connect directly to the corresponding i486 CPU signals. These pins have the same name and functionality as the i486 Microprocessor pins.

2.1.1 ADDRESS LINES A2-A31

A2-A31 are the address lines generated by the i486 CPU and used by the cache as set and tag addresses. A 64k 485Turbocache Module cache will use A4-A14 as set address inputs and the remaining address bits as tag address. A 128k 485Turbocache Module uses A5-A15 as set address inputs, A4 as the line select bit for sectoring, and the remaining bits as tag address. Address lines A2 and A3 are used as burst address inputs.

The address lines are also used as invalidate inputs. At any time, if EADS# is asserted, the address that is present at the address inputs will be invalidated. The 485Turbocache Module will not invalidate unless CS# is sampled active. Note that the address is latched internally so that AHOLD assertion in T1 is permitted.

2.1.2 DATA LINES D0-D31 AND PARITY DP0–DP3

This is the processor data bus common to the i486 CPU, the 485Turbocache Module, and memory bus. The 485Turbocache Module transfers information to the CPU on read hits, and stores data from memory on read misses. The four parity bits, DP0-DP3 are treated just like extra data bits.

2.1.3 ADS#, W/R#, M/IO#

The processor control signals ADS#, W/R#, and M/IO# are used by the 485Turbocache Module to indicate the start of a new cycle, and identify the type of cycle. ADS# assertion indicates a T1 cycle and initiates the tag lookup process in the 485Turbocache Module. I/O cycles are ignored.

ADS# is the primary signal that activates the 485Turbocache Module. When ADS# goes low, the module begins the hit/miss tag lookup regardless of the state of Chip Select (CS#). For this reason, any bus master that controls the ADS# input to the module must meet the module address bus setup and hold times, regardless of the state of CS#. Chip Select, when inactive, disables the module outputs only. (Note that CS# must be asserted for invalidation cycles.)

2.1.4 BYTE ENABLES BE0#-BE3#

Byte enable inputs are used to complete partial byte or word writes to the 485Turbocache Module on cache write hit cycles. All other partial transfers are ignored by the 485Turbocache Module.

2.1.5 BLAST#

BLAST# is used by the 485Turbocache Module to indicate the end of a cycle. If BLAST# is asserted early during a cache line fill from a read miss, that transfer is left invalid by the 485Turbocache Module.

2.1.6 BOFF#

Once BOFF# is sampled by the 485Turbocache Module, it relinquishes control of the data bus in the next clock. If a read hit line transfer was in progress, that transfer will not continue once BOFF# is released. If a read miss transfer was interrupted by BOFF#, the 485Turbocache Module would mark the line as invalid even if the transfer continues once BOFF# has been released. The 485Turbocache Module will recognize invalidations during BOFF#, but will only do so if CS# is active.

2.1.7 FLUSH#

The 485Turbocache Module FLUSH# input behaves exactly like the i486 Microprocessor input. Once asserted, FLUSH# will invalidate the entire contents of its cache memory regardless of the state of CS#. While FLUSH# is asserted, the 485Turbocache Module continues to track CPU bus cycles and treats all accesses as cache misses, activating START# appropriately.

FLUSH# may be used asynchronously with both the i486 CPU and the 485Turbocache Module. If the proper pulsewidths are given, FLUSH# will be recognized, but, it is possible that the FLUSH# will be recognized on different clock edges for each device. This may happen if FLUSH# assertion or deassertion is near its setup and hold times when one device may recognize it and the other may not.

2.1.8 EADS#, AHOLD

EADS# assertion causes the 485Turbocache Module to invalidate the address present on the address bus if CS# is seen active. AHOLD need not be asserted, nor is it even used as an input to the 485Turbocache Module. EADS# may be asserted at most once every other clock as that is the fastest 485Turbocache Module invalidation rate. The section titled "invalidation cycles" describes where EADS# may be asserted for maximum performance.

EADS# may not be asserted on the clock edge of the last T2 of a cycle (the first T1 of the next cycle) if AHOLD is not asserted.

2.1.9 RESET

RESET is an asynchronous input that causes the 485Turbocache Module to reset its internal machines to a known state: its entire cache contents invalidated, and expecting the start of a new bus cycle. RESET must be high for at least 10 clocks for the 485Turbocache Module to reset properly from a warm boot. For a cold boot, RESET must remain active for 3000 ns (100 clocks at 33 MHz, 75 clocks at 25 MHz). There must be no bus activity for at least 4 clocks after the falling edge of RESET so the 485Turbocache Module can reset internally. The falling edge of RESET causes the 485Turbocache Module to sample its WPSTRP# strapping option.

2.2 CPU Bus Interface Signals

These are signals generated by the 485Turbocache Module, or decoded from the i486 CPU that correspond to the CPU bus.

2.2.1 CHIP SELECT CS#

Chip Select is used to select the proper 485Turbocache Module cache module if multiple modules are used, otherwise, with one 485Turbocache Module, CS# may be grounded. CS# is generated by decoding the lowest order tag addresses coming into the module. For example, two 128k cache modules would decode A16 for their chip selects. A16 high would select module 1, while A16 low would select module 2. The following table summarizes the addresses used for decoding:

Size	Modules	Address Bit(s) to Decode
64k	2	A15
64k	4	A15, A16
128k	2	A16
128k	4	A16, A17

For compatibility, A16 and A17 may be decoded for 64k modules. Performance may be increased because of increased granularity, however, if A15 and A16 are used.

With CS# inactive, invalidation cycles are ignored, START# is inactive, and CKEN# is inactive. CKEN# does, however, always activate in T1 as it is not possible for the 485Turbocache Module to recognize CS# before then.

If required, the LOCK# signal may be used as a term in the creation of CS#. If locked cycles do not generate CS#, START# must be generated externally so memory may handle the cycle.

2.2.2 CPU CACHE ENABLE CKEN#

CKEN# is generated by the 485Turbocache Module to indicate that its current transfer, during a read hit cycle, is cacheable. It is always driven (not an open-collector output) and must be used as one of the terms that generates KEN# to the i486 Microprocessor. CKEN# is always active in T1, but then goes inactive and remains inactive unless the cycle is a read hit cycle.

For read miss and write cycles, CKEN# goes inactive in T2 and remains inactive until the next T1. It is the responsibility of the system to generate the KEN# signal to the i486 CPU in these cases.

In a read hit cycle, CKEN# goes active again in the second T2 and remains active throughout the cycle. This forces external KEN# logic to activate KEN# and make the cycle cacheable to the i486 CPU. However, if the line being transferred is write-protected, AND the WPSTRP# pin is strapped low, CKEN# stays inactive in T2 and remains inactive throughout the cycle. This allows write protected lines in the 485Turbocache Module to be cacheable only to the 485Turbocache Module.

2.2.3 BURST READY OUT BRDYO#

The 485Turbocache Module generates BRDYO# when it is bursting data back to the i486 CPU during read hit cycles. BRDYO# is always driven (not an open collector output) and should be used by external logic to create the BRDY# input signal to the i486 CPU. Since the 485Turbocache Module is a zero waitstate, single clock burst cache, BRDYO# is activated in the first T2 until the fourth T2 unless the cycle is interrupted.

2.3 Memory Interface Signals

Memory Interface Signals are signals coming to or from the main memory subsystem. The only signal the 485Turbocache Module generates to the memory system is START#, which is the only signal that must be handled should the 485Turbocache Module be designed as an option.

2.3.1 PRSN#

This signal is tied low inside the 485Turbocache Module. If the system pulls this signal high with a 10K pullup resistor, cache presence will be indicated by that line being pulled low. PRSN# signal is used to indicate that external logic should only start memory cycles when START# goes active rather than from ADS# active.

2.3.2 START#

START# is a signal asserted by the 485Turbocache Module to indicate that the memory subsystem must process the current cycle. START# is always driven and valid and is asserted for all read miss cycles and memory write cycles. START# is not activated for I/O cycles, or if CS# is sampled inactive. START# is normally active in the first T2, but may be delayed if an invalidation cycle forced the previous cycle to be elongated (see 1.3.4 Invalidation cycles).

2.3.3 WRITE PROTECT WP

The Write Protect input is an active high input that indicates to the 485Turbocache Module that the current line transfer is write-protected. It is sampled on the clock edge of the third BRDY# of a line transfer of a read-miss cycle. The 485Turbocache Module saves this information as a single bit in each tag location. In 128k configurations where there is a single tag for 2 consecutive lines, the write protect bit is valid for both lines. If a location has been write-protected, and writes to that location will be ignored.

WP is a synchronous input and must meet the 485Turbocache Module setup and hold times regardless of whether it is being sampled or not.

2.3.4 WRITE PROTECT STRAPPING OPTION WPSTRP#

WPSTRP# is a strapping option that is sampled during RESET. It indicates whether write protected items in the 485Turbocache Module should be cacheable in the i486 CPU cache. If WPSTRP# is high , CKEN# will go active in T2 during all read hit cycles to indicate that they are cacheable. If WPSTRP# is low, CKEN# will be inactive in T2 for read hit cycles to locations that are write-protected. This allows write protected items to be cached by the 485Turbocache Module and not by the i486 CPU.

2.3.5 SYSTEM CACHE ENABLE SKEN#

The SKEN# input to the 485Turbocache Module is like the KEN# input to the i486 Microprocessor. It is sampled just like KEN#, the clock before the first and last transfers of a line fill, to indicate whether the line is cacheable. If the KEN# input to the i486 CPU is connected to the SKEN# input of the 485Turbocache Module, the i486 CPU internal cache and the 485Turbocache Module will cache the same items. It is possible to control KEN# and SKEN# separately so the 485Turbocache Module and i486 CPU cache different areas of memory.

SKEN# is a synchronous input and must meet the 485Turbocache Module setup and hold times regardless of whether it is being sampled or not.

2.3.6 CACHE READY AND BURST READY CRDY#, CBRDY#

CRDY# and CBRDY# are the ready and burst ready inputs to the 485Turbocache Module. They should behave exactly like the i486 CPU RDY# and BRDY# inputs. CBRDY# should be used in conjunction with BRDYO# to generate the i486 CPU BRDY# input. Likewise, CRDY# should be used to form the i486 CPU RDY# input.

The 485Turbocache Module does not sample the CBRDY# or CRDY# inputs during read hits, so it is not possible to artificially add waitstates to the 485Turbocache Module's burst transfer. The CBRDY# and CRDY# inputs must, follow 485Turbocache Module setup and hold times even outside the sampling window.

3.0 SYSTEM CONFIGURATIONS

Two of the most important features of the 485Turbocache Module are its cascadability and its optionality. Below, it is explained how to design a system with a single 485Turbocache Module, multiple 485Turbocache Modules and a socket for an optional 485Turbocache Module.

3.1 Single Cache

In a single cache configuration, the addition of a 485Turbocache Module requires no or little extra logic. Most of the signals are common to the i486 CPU, the memory bus controller, and the 485Turbocache Module. The others, such as KEN#, SKEN#, and START# will be discussed individually.

3.1.1 i486™ MICROPROCESSOR BUS INTERFACE

As seen in Figure 2.1, the i486 CPU-related signals are connected to both the 485Turbocache Module and the memory controller. These are the address bus, data and parity bus, ADS#, W/R#, M/IO#, BE0#-BE3#, BLAST#, RESET, and CLK.

Since a single 485Turbocache Module resides on the address bus, CS# may be tied low so the part is always chip selected.

3.1.2 MEMORY BUS INTERFACE

On the memory bus side, BOFF#, FLUSH#, and EADS# are connected to the i486 CPU and the 485Turbocache Module in parallel. The memory ready signals, CRDY# and CBRDY#, are connected directly to the 485Turbocache Module, but are combined with other system ready signals to form the i486 CPU RDY# and BRDY# inputs. One of the system ready signals is the 485Turbocache Module BRDYO# which must be ANDed with CBRDY# and other burst ready signals to form BRDY# into the CPU.

The memory system must also generate the WP input. If write-protection is not needed, WP may be tied to V_{SS}. If the system would like to prevent write-protected lines in the 485Turbocache Module from being cached by the i486, WPSTRP# should be tied to V_{SS}.

3.1.3 KEN# AND SKEN# GENERATION

The KEN# input to the i486 Microprocessor is a result of all the cache enable signals in the system. Since the 485Turbocache Module activates CKEN# only during a read hit cycle, the CKEN# output may be ANDed with the system cache enable signal to form KEN# to the i486 CPU.

If the 485Turbocache Module and i486 CPU internal cache will cache the same areas of memory, the KEN# input to the i486 CPU may be tied to the SKEN# input of the 485Turbocache Module. Otherwise, the memory system can generate 2 cache enable signals: One that is ANDed with CKEN# to produce KEN#, and another for the SKEN# input.

3.1.4 START# GENERATION

START# goes low to indicate that the memory system must complete the current cycle. This is true for all memory writes and read misses. It is the memory subsystem's responsibility to recognize I/O cycles and begin an I/O access without waiting for START#.

START# is asserted in T2, but may be delayed if there was an invalidation in the previous cycle (see 1.3.4 Invalidation cycles). Because the assertion of START# may be somewhat unpredictable, it is recommended that START# be used to either begin a DRAM RAS cycle, or enable DRAM output buffers.

Figure 3.1a shows that START# may be the indication to DRAM control to begin a cycle. Once START# is sampled active, a RAS and CAS cycle begin. This will incur an extra waitstate to cache read misses since the earliest a memory cycle will begin is the first T2.

Figure 3.1b shows that START# may enable DRAM data buffers. The actual DRAM cycle begins once ADS# and M/IO# are sampled low, but will not complete until the buffers have been gated allowing data to be written to the i486 CPU data bus. Should the cycle be a 485Turbocache Module read hit, the buffers are never enabled. Since the 485Turbocache Module takes 5 clock cycles to complete the burst transfer, RAS precharge time can easily be absorbed.

See 4.3.4 START# Predictability for detailed information how START# may be asserted in a predictable manner.

3.2 Multiple Cache

A multiple cache scheme is similar to the single cache scheme because all of the i486 Microprocessor bus interface signal connection remain the same. Like the single cache example, only KEN#, SKEN#, START#, and now CS#, need special handling. Figure 3.2 is an example of a 512k multiple cache configuration.

3.2.1 MEMORY BUS INTERFACE

Like the i486 Microprocessor bus interface signals, BOFF#, FLUSH#, and EADS# are connected to the CPU, memory system, and all caches in parallel. The ready and burst ready outputs from the memory system connect to the CRDY# and CBRDY# inputs to all 485Turbocache Module caches. The CBRDY# signal is then ANDed with the BRDYO# outputs from all 485Turbocache Modules to form BRDY# to the i486 CPU.

3.2.2 START#

START# is activated by a single 485Turbocache Module at a time because CS# is active for a single 485Turbocache Module at a time. START#, therefore, may be ANDed with all other START# signals to form a system start indication. See section 3.1 Single Cache for details how START# may be used.

3.2.3 KEN#

Like START#, CKEN# is only activated for chip selected modules. Therefore, all CKEN# outputs may be ANDed together to form the i486 CPU KEN# signal. A system cache enable signal must also be included in the AND terms since it is the system's responsibility to generate KEN# during read miss cycles.

3.2.4 SKEN#

Since SKEN# is used during read miss cycles and ignored otherwise, the system cache enable signal can be connected to all 485Turbocache Modules' SKEN# inputs. If multiple sources can create the

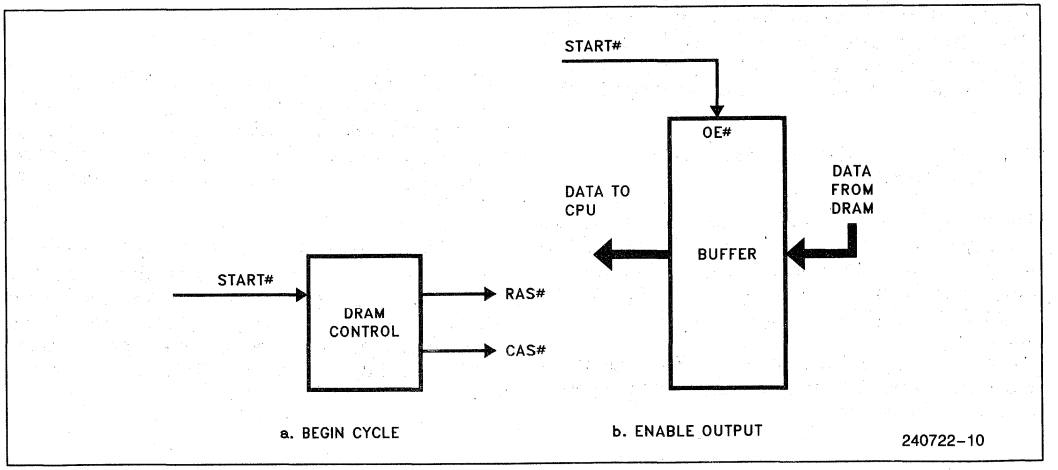


Figure 3.1. Using START# in DRAM Control

KEN# signal to the i486 CPU, KEN# may be fed into all 485Turbocache Modules. If the i486 CPU caches different memory locations than the second-level cache, SKEN# must be generated separately and then connected to all 485Turbocache Module inputs.

3.2.5 CS#

Chip select is used to identify which 485Turbocache Module is being addressed. It is the result of decoding the lowest order tag address bits. Figure 3.2 shows how a PLD chooses one of four 485Turbocache Modules. Anytime an address is present on the address bus, including invalidation cycles, one of the 485Turbocache Modules is selected.

3.3 Optional Cache

The 485Turbocache Module is an optional cache. However, its most powerful feature is allowing a system to reconfigure itself easily once a 485Turbocache Module has been installed. To accomplish this, the 485Turbocache Module is designed as a write-through cache with all signals feeding around to the memory subsystem whether the 485Turbocache Module is present or not. There are only a few considerations that need to be made to allow the 485Turbocache Module to be fully optional.

3.3.1 SIGNAL CONSIDERATIONS: START#, CKEN#, BRDYO#

If the 485Turbocache Module is not present in a system that expects it to be, the START# signal will never be asserted and memory will never begin a cycle. A solution to this problem is to connect the PRSN# presence pin into the memory controller that accepts START#. If PRSN# is high, the 485Turbocache Module is not present, and all memory cycles should begin with the assertion of ADS#. Note that START# should have a pullup resistor to ensure it is not left floating.

When the 485Turbocache Module is removed from a system the CKEN# and BRDYO# signals, which are combined with external logic to form KEN# and BRDY#, will be left floating. All CKEN#, BRDYO#, START#, and PRSN# pins should have pullup resistors tied to them. This assures an inactive state when no 485Turbocache Module is present.

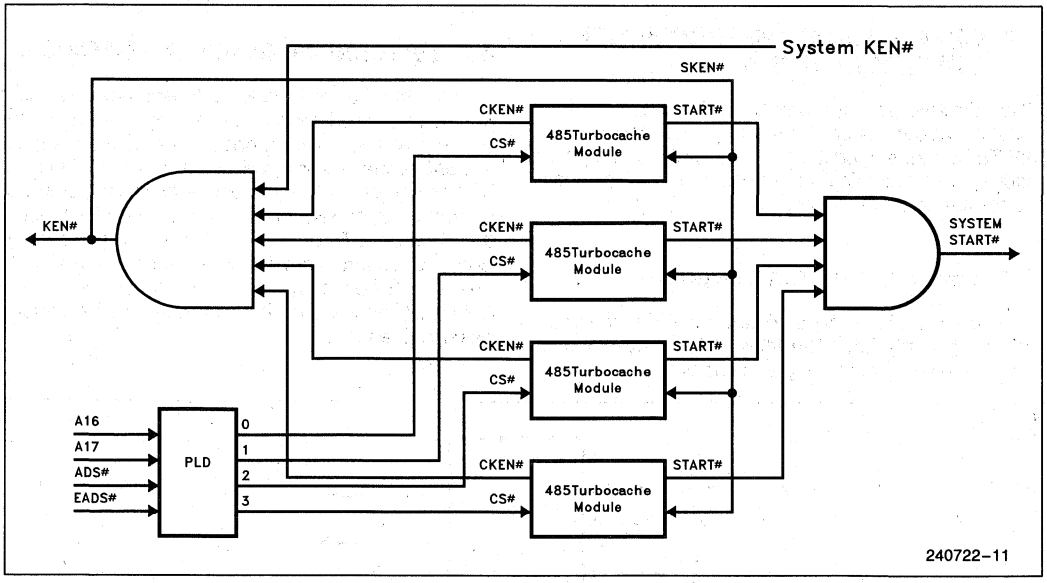


Figure 3.2 Multiple Cache Configuration

3.3.2 CONSIDERATIONS WITH MULTIPLE CACHES

As long as all the START#, CKEN#, BRDYO#, and PRSN# signals have pullup resistors tied to them, all empty cache sockets will respond like inactive caches. There is, however, a chip selecting problem since CS# decoding varies with the number of caches that are present.

Chip select decoding logic, like Figure 3.3 shows, should have all PRSN# pins as input. From this information, the correct chip select decoding can be generated. The logic in Figure 3.3 is able to keep CS1 asserted if one cache is detected, decode A16 if 2 caches are detected, or decode A16 and A17 if all 4 caches are present.

The most difficult problem to overcome when allowing an optional number of multiple caches is to account for capacitive load changes. Since each cache has a capacitive load on the data bus and clock lines, some amount of design effort must be spent resolving capacitive loading. When designing with 4 caches, each cache will probably have to receive a dedicated clock line. As well, the data bus will have to be buffered outside of the CPU and cache core.

4.0 OPERATIONAL/PERFORMANCE CONSIDERATIONS

The following sections provide more detailed information about operating and designing-in the 485Turbocache Module. This includes testing the cache, understanding sectoring, and making small performance adjustments.

4.1 Testing and Data Integrity

The 485Turbocache Module can monitor data integrity using parity bits. The i486 Microprocessor has the capability of outputting and checking data parity. The memory subsystem must also support parity to use the parity support on the 485Turbocache Module. This data parity information is stored with every byte inside the 485Turbocache Module, and is checked by the i486 CPU during data reads. To be able to identify data errors from memory or cache, the parity error check output (PCHK#) of the i486 CPU can be sampled.

Power up self test programs test main memory functionality on a cell by cell basis since parity logic is not capable of detecting all memory failures. It is also important to test cache memory. The following algorithm will test any number of 64k 485Turbocache Modules or 128k 485Turbocache Modules up to 512k of cache memory:

1. Flush or Reset the cache.
2. Write "1" to every bit of a 512k block of memory.
3. Read the 512k block twice; this fills the cache.
4. Disable CS# and write "0" to the 512k block; this fills memory.
5. Read the 512k block:
   - Repetitive assertions of START# indicate the cache boundary (size of cache)
   - Data $\neq$ 1 indicates bad tag or SRAM
6. Repeat with "0" in the cache and "1" in memory.

4.2 Sectored vs Non-Sectored Cache

The 64k 485Turbocache Module was designed as a 64k non-sectored cache; this means each tag of the cache points to 1 line of data in the cache memory. A 128k cache requires twice the number of tags to be non-sectored. This increases tag size, complexity, and reduces tag lookup speed. For this reason, the 128k 485Turbocache Module is a sectored cache. Each tag in the 128k 485Turbocache Module points to 2 consecutive lines in the cache. A Line Select bit, address bit A4, determines which line is being referenced.

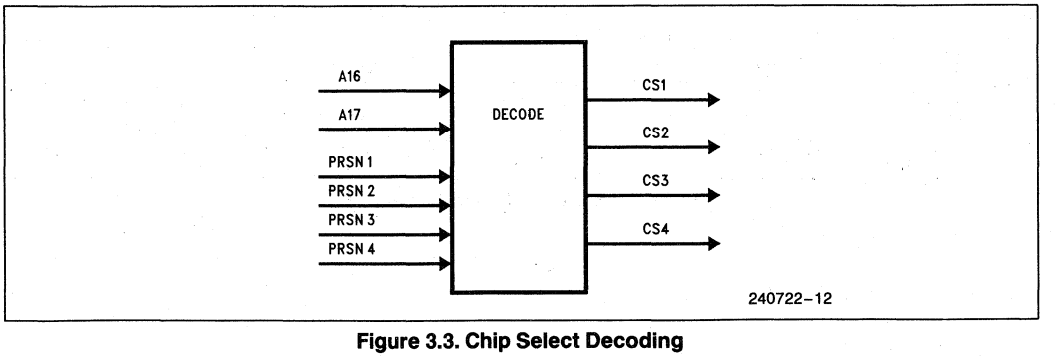


Figure 3.3. Chip Select Decoding

Figure 4.1 is an example of one tag in a sectored cache. If this tag points to address 2500h, then the adjacent line is reserved for address 2510h (A4 high). If, for example, address 2510 had been written first, the tag would still contain 25 and only address 2500 could be placed in the first line.

Since the Line Select Bit is used for a sectored architecture, all set and tag address bits are shifted higher in the address space. The 128k 485Turbocache Module internally compensates for this shift so pin-compatibility with the 64k 485Turbocache Module is maintained. This allows either cache configuration, 64k 485Turbocache Module or 128k 485Turbocache Module, to be hardware-transparent.

Because a sectored cache references 2 consecutive lines, the odds of filling both lines is reduced, and thus the hit rate of the cache. A sectored cache will have a slightly reduced hit rate compared to an equivalent non-sectored cache, but simulations have shown the performance penalties to be minimal (1 to 2 percent). Simulations have also shown that a two-way set associative sectored 128k cache offers significantly better performance than a direct mapped 128k non-sectored cache.

4.3 Performance Considerations

The following section offers a few special considerations that will increase cache performance or ease hardware design. These considerations are simply design notes and are not deviations from the i486 Microprocessor specification.

4.3.1 SKEN# ASSERTION

SKEN# is an input to the 485Turbocache Module to indicate the cacheability of a line during a read miss cycle. It is sampled exactly like KEN# in the i486 CPU, one clock before the first dword transfer of a line fill, and one clock before the last dword.

During a line fill, the 485Turbocache Module loads the dwords of the line directly into the appropriate spot in cache memory. This means that once SKEN# has been sampled active by the 485Turbocache Module, it must "commit" a line and invalidate a location to prepare for the incoming line. Once a line fill completes with a proper SKEN#, the line can be validated.

A potential performance loss exists if a system designer chooses, during non-cacheable cycles, to keep SKEN# active, but inactivate SKEN# the clock before the first transfer (see Figure 4.2). Once the 485Turbocache Module sees SKEN# low in the first T2, it commits a line in the cache by invalidating an entry despite the fact that SKEN# was later deasserted. The performance loss can be avoided if SKEN# was held inactive until cacheability could be determined.

4.3.2 INVALIDATION WINDOW

When an invalidation is requested with the assertion of EADS#, the 485Turbocache Module must immediately invalidate the address present on the address bus. If the tag portion of the 485Turbocache Module is in use, the invalidation takes priority and will suspend the other action. This may decrease performance. To avoid this, EADS# should not be issued in the second, third or fourth transfer of a cache read miss cycle. Section 1.3.4 Invalidation Cycles under Functional Description explains this in detail.

4.3.3 BOFF# ASSERTION

If BOFF# is asserted and the 485Turbocache Module is in the middle of a cacheable read miss cycle, the 485Turbocache Module treats the current line fill as non-cacheable. Once BOFF# is released and the cycle continues, the 485Turbocache Module will treat the rest of the cycle as a non-cacheable cycle.

In most systems BOFF# is a rare occurrence, thus the performance loss is negligible. If, however, BOFF# is regular and predictable, system performance can be increased by timing BOFF# so that the four dword transfers of a line fill are never interrupted. Section 1.3.5 BOFF# Cycles under Functional Description explains aborted cycles in more detail.

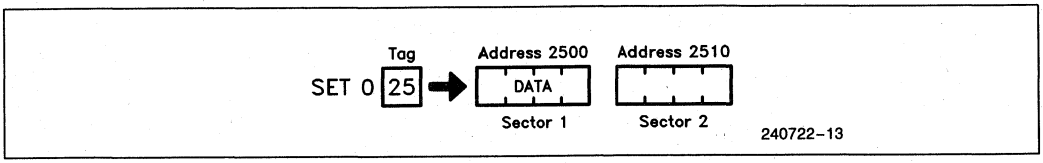


Figure 4.1 Sectored Example

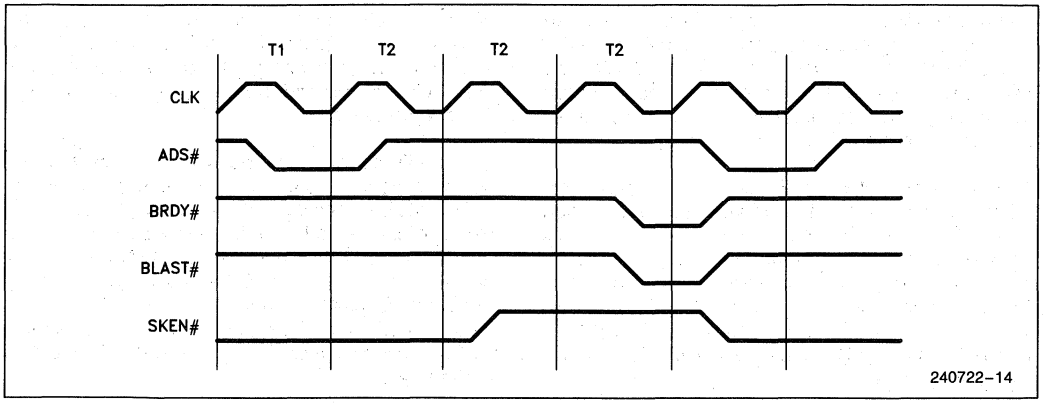


Figure 4.2 Method of SKEN# Generation Not Recommended

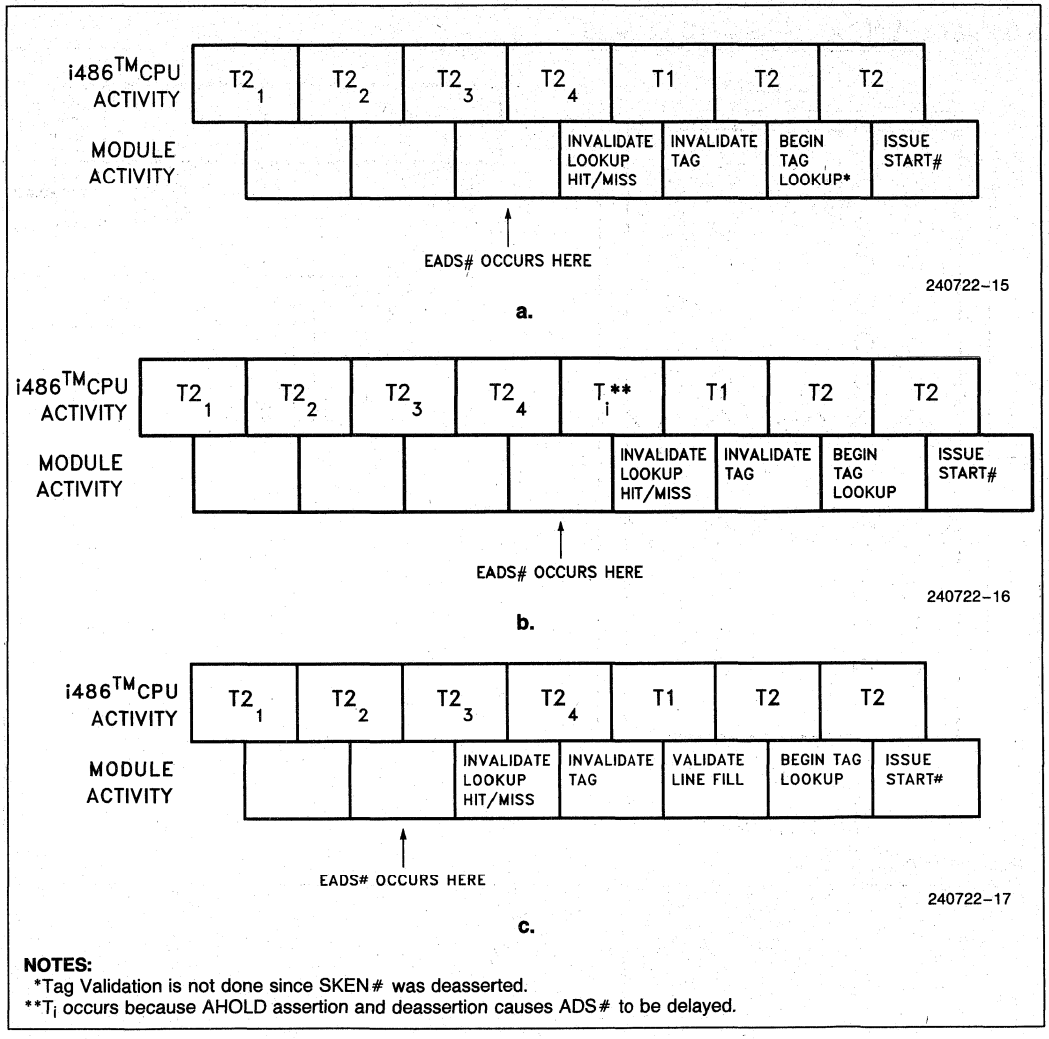
4.3.4 START# PREDICTABILITY

START# is asserted in the first T2 of a read miss cycle unless an invalidation occurred in the previous cycle. The section titled "Invalidation Cycles" explains why START# may be delayed. If START# must be a predictable signal to the system, and invalidation cycles cannot be timed to occur before the second transfer of a read miss cycle, there is a way to ensure the predictability of START#.

When EADS# is asserted towards the end of a read miss cycle, there are 3 tag accesses that need to be made before T1 of the next cycle: invalidate lookup, the actual invalidation (if a hit), and validation of the current line fill (if cacheable). Since there is no way to predict the hit/miss possibility of an invalidation request, it is assumed that 2 tag accesses will be required to service it. One tag access can be saved, then, by making the current line fill non-cacheable.

To do this, SKEN# to the 485Turbocache Module may be deasserted if AHOLD is detected. If SKEN# is deasserted the clock before the last CBRDY#, the line is non-cacheable. Figure 4.3a shows how assertion of EADS# during the third transfer of a burst cycle incurs a 1 clock delay in START#. Figure 4.3b shows EADS# assertion in the fourth clock, but since AHOLD will cause the CPU to delay ADS# at least one extra clock, START# is delayed only 1 clock as well. Assertion of EADS# in the second transfer of a burst causes a 1 clock delay in START# without deasserting SKEN# (see Figure 4.3c), so there is no advantage in dropping SKEN# for EADS# assertion then.

In summary, if SKEN# is deasserted in response to AHOLD during the third of fourth transfer of a line fill, START# will be delayed at most 1 clock. This makes START# predictable: It will always be valid in the second T2 of a read miss cycle. Note that if START# was not delayed, its value is retained in the second T2.

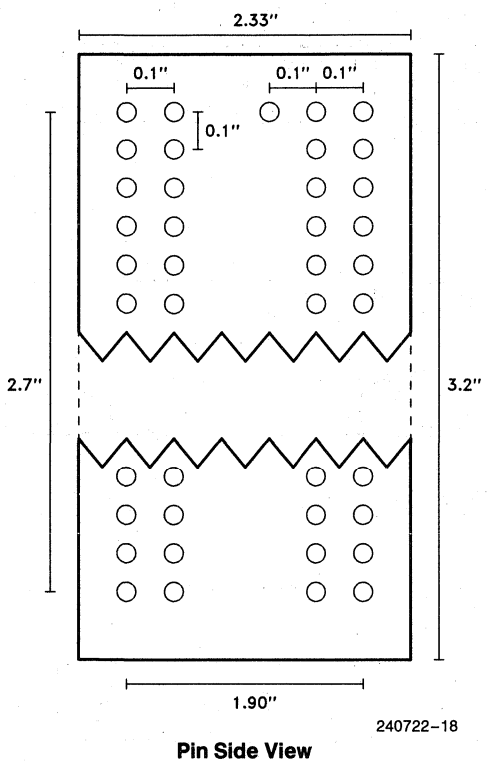


NOTES:
*Tag Validation is not done since SKEN# was deasserted.
**T_i occurs because AHOLD assertion and deassertion causes ADS# to be delayed.

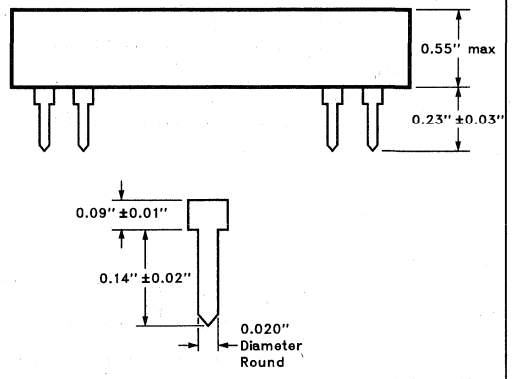
Figure 4.3 Predictable START# Delay

5

5.0 MECHANICAL SPECIFICATIONS



Pin Side View



Figures Not Drawn To Scale

6.0 ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature under Bias ....0°C to +70°C
Storage Temperature .....−55°C to +150°C
Voltage on Any Pin
with Respect to Ground ...−0.5V to V_{CC} + 0.5V
Power Dissipation:
64k 485Turbocache Module 4W
128k 485Turbocache Module 6W

NOTICE: This data sheet contains preliminary information on new products in production. The specifications are subject to change without notice.

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

7.0 D.C. CHARACTERISTICS (V_{CC} = 5V ±5%)

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	2.2	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		0.4	V	1
V_{OH}	Output High Voltage	2.4		V	2
V_{CIL}	Clock Input Low Voltage	−0.3	+0.8	V	
V_{CIH}	Clock Input High Voltage	2.2	V_{CC} + 0.3	V	
I_{CC}	Supply Current Supply Current		800 1200	mA mA	82485MA 82485MB
I_{LI}	Input Leakage Current: D0–D31 and Parity D0–D31 and Parity CLK CLK TAI15, TAI16, WPSTRR# All Other Inputs		 ±20 ±40 ±65 ±75 −400 ±15	 µA µA µA µA µA µA	 82485MA 82485MB 82485MA 82485MB
I_{LO}	Output Leakage Current: D0–D31 and Parity D0–D31 and Parity		 ±20 ±40	 µA µA	 82485MA 82485MB
C_{IN}	Input Capacitance: —A2 Through A31 —BE0 Through BE3 —ADS#, M/IO#, W/R# —CRDY#, CBRDY# —D0 Through D31 —D0 Through D31		 20 20 20 20 25 45	 pF pF pF pF pF pF	 82485MA 82485MB
C_{CLK}	Clock Input Capacitance Clock Input Capacitance		45 75	pF pF	82485MA 82485MB
C_O	I/O Capacitance		25	pF	

ADVANCE INFORMATION
SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:
1. Measured at 4.5 mA.
2. Measured at 1.0 mA.

5

8.0 A.C. CHARACTERISTICS (V_{CC} = 5V ±5%)

All A.C timings are tested with a capacitive load of 50 pF unless otherwise specified.

Symbol	Parameter	25 MHz		33 MHz		Fig.	Notes
		Min(ns)	Max(ns)	Min(ns)	Max(ns)		
t_1	CLK Period	40	42	30	42	9.1	
t_2	CLK High Time	14		11		9.1	1
t_3	CLK Low Time	14		11		9.1	2
t_4	CLK Fall Time		4		3	9.1	
t_5	CLK Rise Time		4		3	9.1	
t_6	A2–A31, BE0#–BE3# Setup Non-Snoop	17		13 *		9.3	
t_{6a}	A2–A31, BE0#–BE3# Hold Non-Snoop	3		3		9.3	
t_7	ADS#, M/IO#, W/R# Setup	17		13		9.2	
t_{7a}	ADS#, M/IO#, W/R# Hold	3		3		9.2	
t_8	BLAST# Setup	10		9		9.2	
t_{8a}	BLAST# Hold	3		3		9.2	
t_9	CRDY#, CBRDY# Setup	8		5		9.2	
t_{10}	CRDY#, CBRDY# Hold	3		3		9.2	
t_{11}	SKEN# Setup	8		5		9.2	
t_{12}	SKEN# Hold	3		3		9.2	
t_{13}	D0–D31, DP0–DP3 Setup	5		5		9.2	
t_{14}	D0–D31, DP0–DP3 Hold	3		3		9.2	
t_{15}	WP Setup	8		8		9.2	3
t_{16}	WP Hold *	3		3		9.2	
t_{17}	BOFF# Setup	10		8			
t_{18}	BOFF# Hold	3		3			
t_{19}	EADS# Setup	8		5		9.3	4
t_{20}	EADS# Hold	3		3		9.3	
t_{21}	A4–A31 Setup (Snoop)	5		5		9.3	
t_{22}	A4–A31 Hold (Snoop)	3		3		9.3	
t_{23}	RESET, FLUSH# Setup	10		5		9.4	
t_{24}	RESET, FLUSH# Hold	3		3		9.4	
t_{25}	RESET, FLUSH# Pulse Width (Asynchronous Use)	60		45			

ADVANCE INFORMATION
SEE INTEL FOR DESIGN-IN INFORMATION

8.0 A.C. CHARACTERISTICS (V_{CC} = 5V ±5%) (Continued)

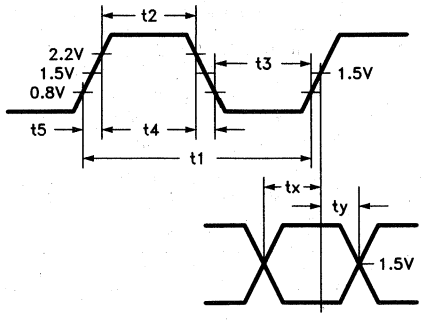
All A.C timings are tested with a capacitive load of 50 pF unless otherwise specified.

Symbol	Parameter	25 MHz		33 MHz		Fig.	Notes
		Min(ns)	Max(ns)	Min(ns)	Max(ns)		
t_{26}	BRDYO# Valid		22		16	9.3	
t_{27a}	CKEN# Valid	3	18	3	15	9.3	
t_{27b}	CKEN# Hold		12		12	9.3	5
t_{28}	START# Valid	5	22	5	16	9.2	
t_{29}	D0–D31 Valid (Read Hit)		30		24	9.3	6
t_{30}	CS# Setup	6		6		9.2	7
t_{31}	CS# Hold	3		3		9.2	

ADVANCE INFORMATION FOR DESIGN-IN SEE INTEL INFORMATION

NOTES:
1. At 2.2V.
2. At 0.8V.
3. Setup to CLK edge of third BRDY# in line fill.
4. Setup to CLK edge where EADS# is valid.
5. Hold time from CLK edge in which CKEN# will be sampled.
6. Valid up to C_L = 100 pF.
7. At the clock edge in which ADS# or EADS# is sampled.

9.0 WAVEFORMS



tx = input setup times
ty = input hold times, output float, valid and hold times

Figure 9.1. CLK Waveforms

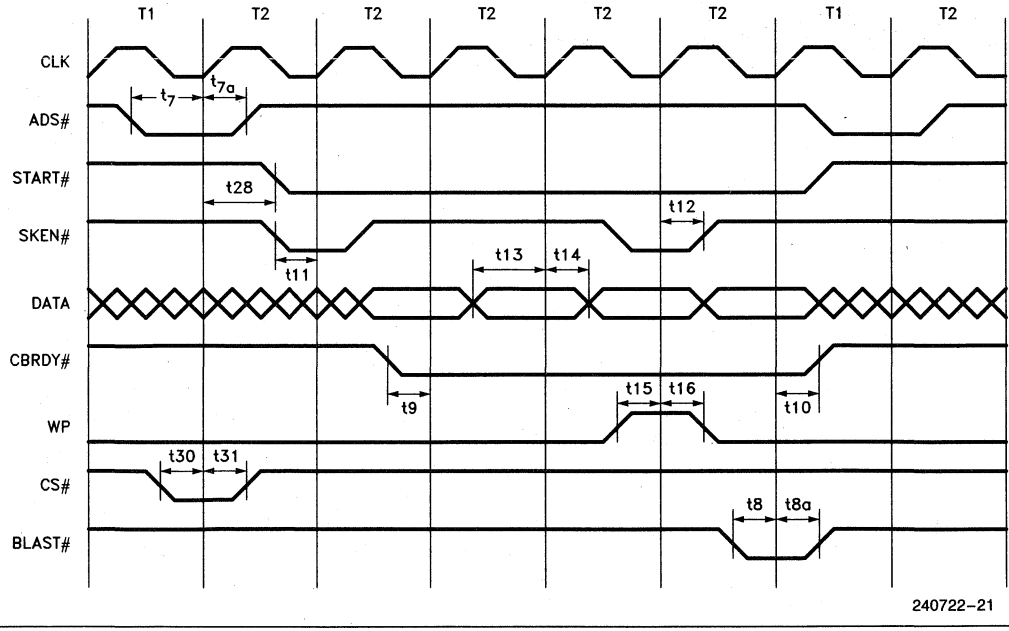
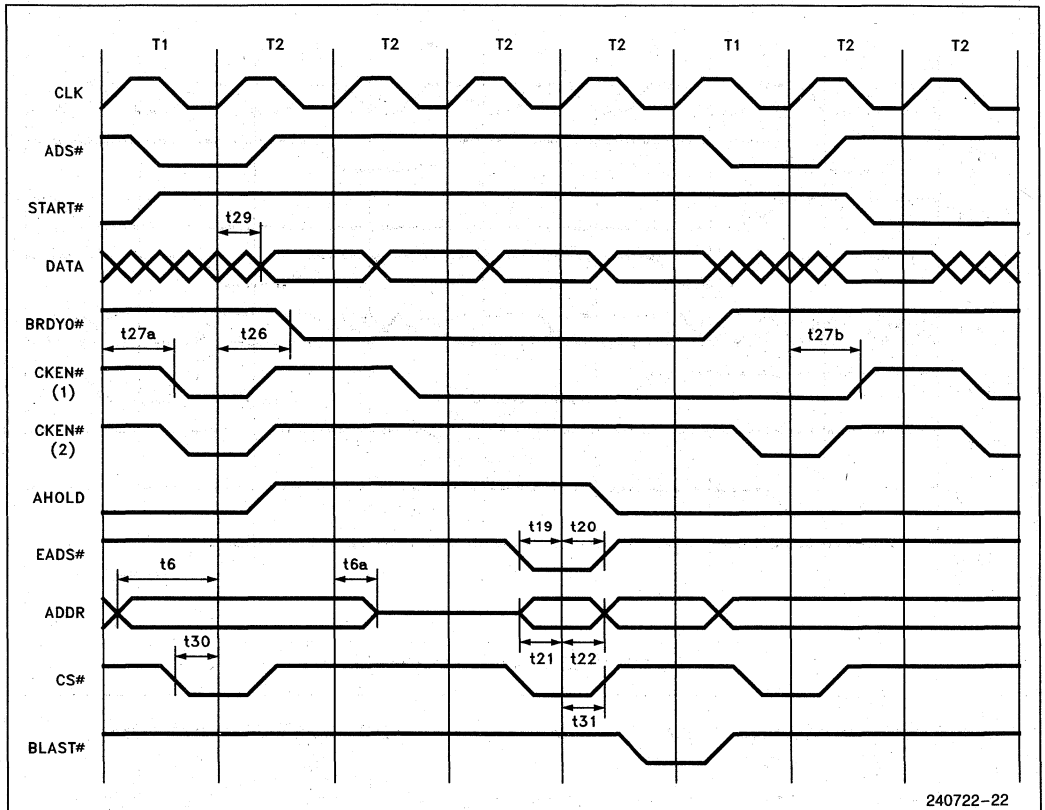


Figure 9.2. Write Protected Read Miss



1. Normal CKEN# behavior.
2. CKEN# behavior if line is Write Protected and WPSTRP# is low.

Figure 9.3. Read Hit Cycle and Write Cycle with Invalidation

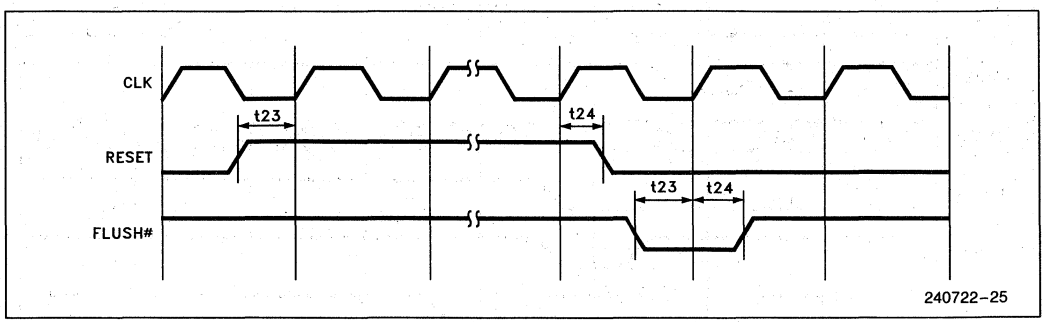


Figure 9.4. RESET and FLUSH #

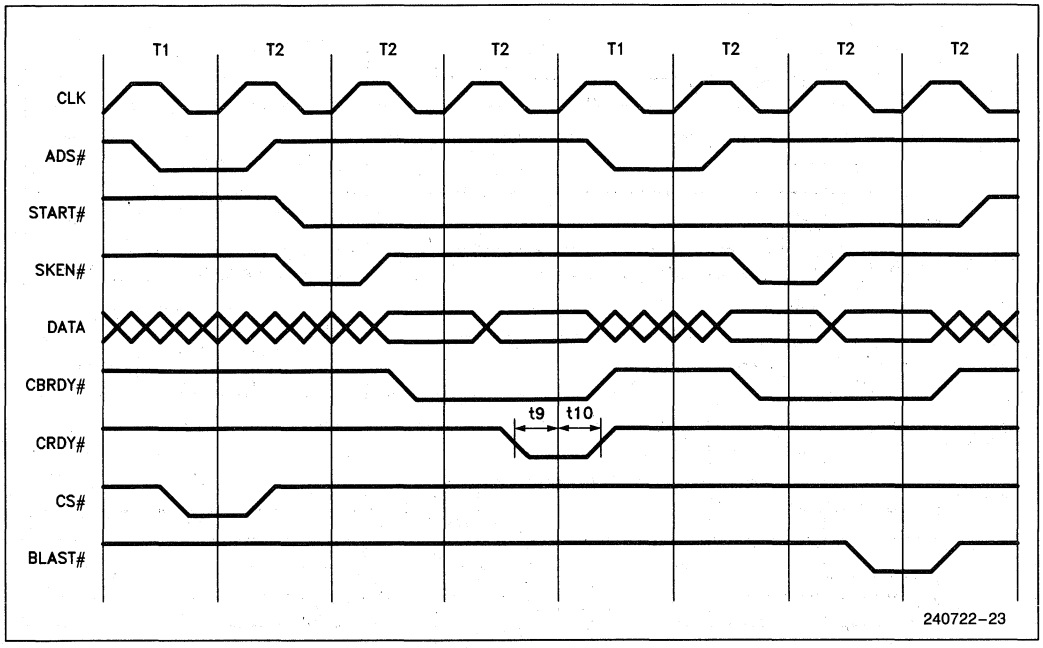


Figure 9.5. Multiple Cycle Line Fill

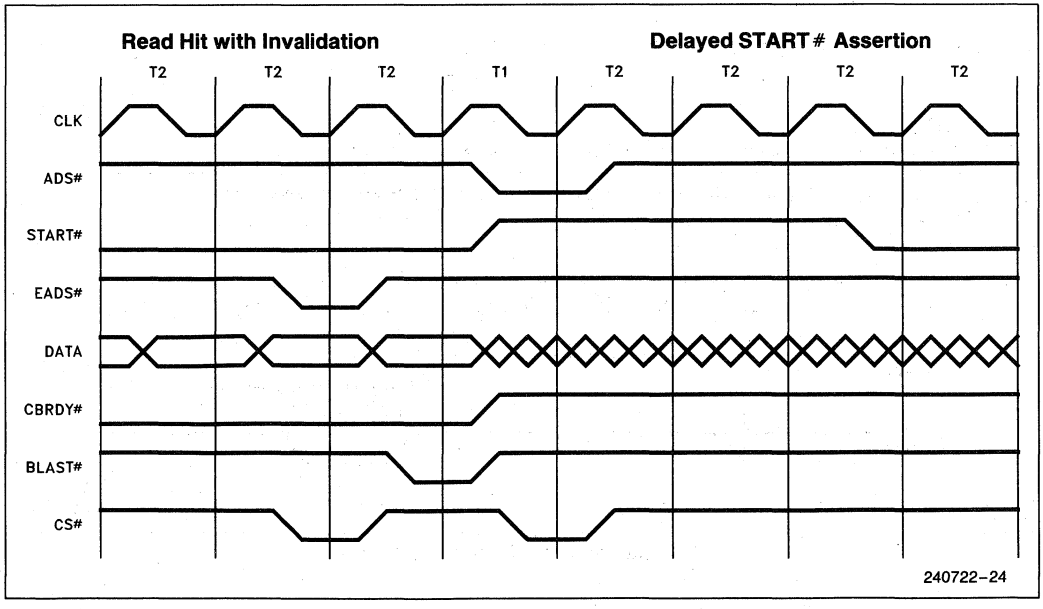


Figure 9.6. Invalidation Causing Delayed START#

10.0 REVISION HISTORY

Revision -002 of the 485Turbocache Module Data Sheet contains several updates and corrections to the original version. A revision summary of major changes is listed below:

Throughout Document	The name of the cache module has been changed from Turbocache 486 Module to 485Turbocache Module.
Section 1.3.1	Clarified that all transfers seen by the 485Turbocache Module are assumed to be 32-bit transfers.
Section 1.4	Removed one incompatibility between the 485Turbocache and the i486 CPU.
Section 2.1.9	Corrected RESET specifications.
Section 5.0	Made mechanical specifications more precise.
Section 6.0	Modified absolute maximum ratings.
Section 7.0	Modified V_{IH} and V_{OL} specifications. Added input and output leakage current specifications.
Section 8.0	Corrected AC specifications t_7 and t_{25}. Added AC specifications t_{6a} and t_{7a}.

5

82485
SECOND LEVEL CACHE CONTROLLER FOR THE i486™ MICROPROCESSOR

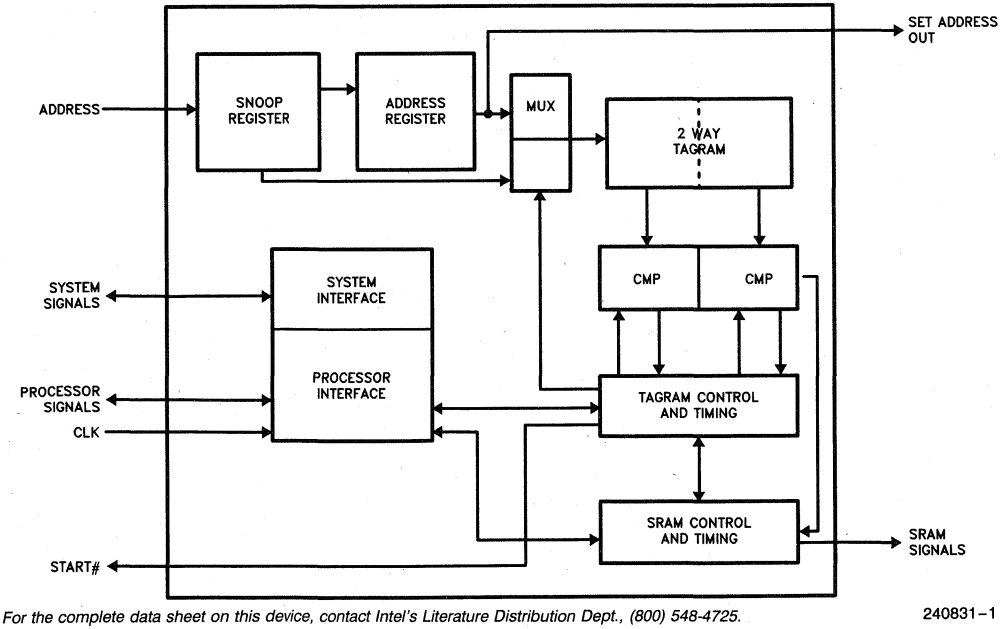
- **High Performance**
  - **— Zero Wait State Access on Cache Hit**
  - **— One Clock Bursting**
  - **— Two-Way Set Associative**
  - **— Write Protect Attribute Per Tag**
  - **— Start Memory Cycles in Parallel**
- **Easy to Use**
  - **— Matches i486™ Microprocessor Bus Timing**
  - **— Supports Invalidation Cycles**
  - **— Maintains Memory on Writes**
- **High Integration**
  - **— Single Chip Tag RAM and Controller**
  - **— No Logic Needed for CPU and Cache Connection**
  - **— Maps Full 4 Gigabyte Address Space**
- **Flexible System Configurations**
  - **— Supports 64K or 128K Cache Memory Per Controller**
  - **— Allows Multiple Controllers for Larger Cache Size**
  - **— Supports Non-Cacheable Memory Areas**

The 82485 is a second-level cache controller designed to improve the performance of i486™ Microprocessor systems. One 82485 cache controller supports 64K or 128K bytes of second level cache memory that maps to the entire 4 Gigabytes of the i486 microprocessor address space. The controller is completely software transparent. Several controllers may be cascaded to provide larger cache sizes. One controller plus SRAMs provides a 64K or a 128K cache. External EPROM can be cached yet remain write protected. The 82485 is fully compatible with the i486 microprocessor. All i486 CPU bus cycles and timings are supported.

A complete, optional second level cache controller using the 82485 is available as the 485Turbocache Module from Intel (data sheet order number 240722).

i486 is a trademark of Intel Corporation.

82485 Internal Block Diagram



For the complete data sheet on this device, contact Intel's Literature Distribution Dept., (800) 548-4725.

240831-1

November 1990
Order Number: 240831-001

APPLICATION NOTE

AP-447

November 1990

A Memory Subsystem for the i486™ CPU including Second Level Cache

5

GREGORY A. ROBERTSON
SENIOR APPLICATION ENGINEER

Order Number: 240799-001

A MEMORY SUBSYSTEM FOR THE i486™ CPU INCLUDING SECOND LEVEL CACHE

CONTENTS

PAGE

1.0 INTRODUCTION 5-209

2.0 THE 485TURBOCACHE SECOND LEVEL CACHE MODEL 5-209

3.0 PROCESSOR FEATURE REVIEW 5-215

3.1 The Burst Cycle 5-215

3.2 The KEN# Input 5-216

3.3 Bus Characteristics 5-217

4.0 DRAM INTERFACE OVERVIEW ... 5-220

4.1 Functional Blocks 5-220

4.2 Address Path Logic 5-221

4.3 Data Path 5-223

4.4 Second Level Cache Support 5-224

4.5 Control Logic 5-225

5.0 MEMORY SUBSYSTEM FUNCTION 5-227

5.1 CPU Interface Function 5-227

5.2 Data Path Control 5-228

CONTENTS

PAGE

5.3 Address Path Control 5-228

5.4 DRAM Interface 5-228

5.5 Controller Signals 5-229

5.6 Read Cycles 5-229

5.7 Write Cycles 5-231

5.8 Consecutive Bus Cycles 5-233

5.9 Page Miss Cycles 5-234

5.10 Refresh Cycles 5-236

6.0 CONTROLLER IMPLEMENTATION 5-237

6.1 Cycle Tracking Logic 5-238

6.2 RAS# Logic 5-241

6.3 CAS# Logic 5-242

6.4 Write Control Logic 5-244

6.5 Burst Address Logic 5-244

7.0 SUMMARY 5-246

7.1 Timing Restrictions 5-247

APPENDIX A 5-248

1.0 INTRODUCTION

The i486™ CPU contains several improvements over its predecessor, the highly successful 386™ CPU. One of the most important of these is the processor's data access rate. The i486 CPU can access instructions and data from its on-chip cache in the same clock cycle. To support the processor's redesigned internal data path, the external bus has also been optimized and can access external memory at twice the rate of the 386 CPU. The internal cache requires rapid access to entire cache lines. Invalidation cycles must be supported to maintain consistency with external memory. All of these functions must be supported by the external memory system. Without them, the full performance potential of the CPU cannot be attained.

The requirements of todays multitasking and multiprocessor operating systems also put increased demand on the external memory system. OS support functions such as paging and context switching can degrade reference locality. Without efficient access to external memory, the performance of these functions is reduced.

Second level caching is a technique used to improve the memory interface. Some applications, such as multiuser office computers, require this feature to meet performance goals. Single-user systems, on the other hand, may not warrant the extra cost. Given the variety of applications incorporating the i486 CPU, memory system architecture will be very diverse.

In this application note, we will work with an example to discuss the details of memory system design. In the example, we have supported as many functions of the CPU as possible. An optional second-level cache is included. A write buffer is also implemented to reduce write latency. The cache supports zero wait state read cycles. The DRAM controller supports the following devices with the wait states shown in Table 2. The DRAM speed given in Table 1 is the RAS access time (tRAC). Table 2 summarizes the bus clocks required for each function.

Table 1

CPU Clock Freq.	DRAM Speed
25 MHz	100 ns
33 MHz	70 ns

Many of the functions and optimizations included here will not be required in every application. The example provides guidelines for the hardware designer but will not necessarily provide the optimal cost/performance solution for many applications. For example, 11 PLDs are required to implement the memory control logic partially due to the implementation of a back-off capability. An address register must also be used to implement this function. If this function is not used, the control logic can be substantially reduced. These and other optimizations will be discussed in the summary.

Table 2

DRAM Function	First Access Burst	Subsequent Burst Accesses	Write Cycles
Page Hit	3	1	2
Page Miss	7	1	5*

NOTE:
*Write miss latencies occur only during cycles subsequent to a write miss cycle.

The discussion assumes a working knowledge of computer system design. Items discussed but not explained include DRAM operation, PLD programming and operation, worst-case timing analysis and i486 CPU bus operation. The complete schematics and PLD equations are in Appendix A.

2.0 THE 485TURBOCACHE SECOND LEVEL CACHE MODULE

Several different types of second level cache architectures are possible candidates for use with the 486 CPU. For single cpu systems the different architectures offer similar performance benefits in most cases. The reason they are so similar is the mechanism which improves performance. The primary benefit of the second level cache is bus cycle latency reduction.

In most systems which incorporate a single i486 CPU, bus traffic from other bus masters is minimal. With any reasonable memory system the CPU uses at most 50% to 70% of the bus. Therefore reduction of bus cycle latency is the only performance benefit external logic can offer.

The second level cache used in this example is an econonmical method of reducing read cycle latency. The 485Turbocache module contains the control circuits, data and tag ram required to implement a 128k byte cache. It is organized as a two way set associative cache. Modules can be cascaded to provide up to 512K bytes of cache memory.

One of the most interesting aspects of this device is it can be a system option. To provide this capability the device is configured as a look-aside cache. It monitors the CPU address and control signals. When a cycle occurs in which the cache can supply data, it intervenes. The cache module then supplies an entire 16-byte line with no wait states.

The performance improvement offered by this cache is substantial in some environments. This performance improvement is particularly obvious when executing multitasking, multiuser operating systems such as

UNIX and OS/2. Some users, however, may not require the performance improvement offered by the cache. In these cases the cache as an option is attractive.

By designing the cache subsystem as an option both user's requirements can be met. A single system design can be manufactured for both customers. The UNIX or OS/2 user can add the cache module. Other users may or may not require the module. They can choose the system configuration which meets their price-performance needs.

When a single or multiple 485Turbocache Module devices are connected to an i486 processor system, the processor's internal cache should map the entire address space including that of the 485Turbocache Module devices to provide the highest performance. This is the most efficient configuration. The i486 CPU can access a line from its internal cache in one clock and the 485Turbocache Module provides the next fastest access in two clocks for the first doubleword and the remaining three doublewords in three clocks.

No matter how many 128-kbyte modules are cascaded, the set and tag addresses are connected to the same pins on the 485Turbocache Module. The processor's address bits A2–A31 are connected to A2- -A31 on the 485Turbocache Module. Internally, address bits A4–A15 are sent to both sets, to select one of 4,096 locations. Because the cache is two-way set associative, each address points to information stored in two banks. On each read or write cycle, the value of A16–A31 is compared to the tags stored at the location addressed by A4–A15. If they are equal, and if the valid bit is set, then a hit occurs. If a read cycle is in progress, then the 485Turbocache Module returns data to the i486 CPU. If the hit cycle is a write cycle, then the new data is updated in the 485Turbocache Module.

When multiple 485Turbocache Modules are used, the chip select starts by decoding A16 onwards. For example, with a 256-kbyte cache A16 and A17 are decoded for generating the CS#. The set and tag addresses of a system with four 485Turbocache Modules is shown in Figure 1.

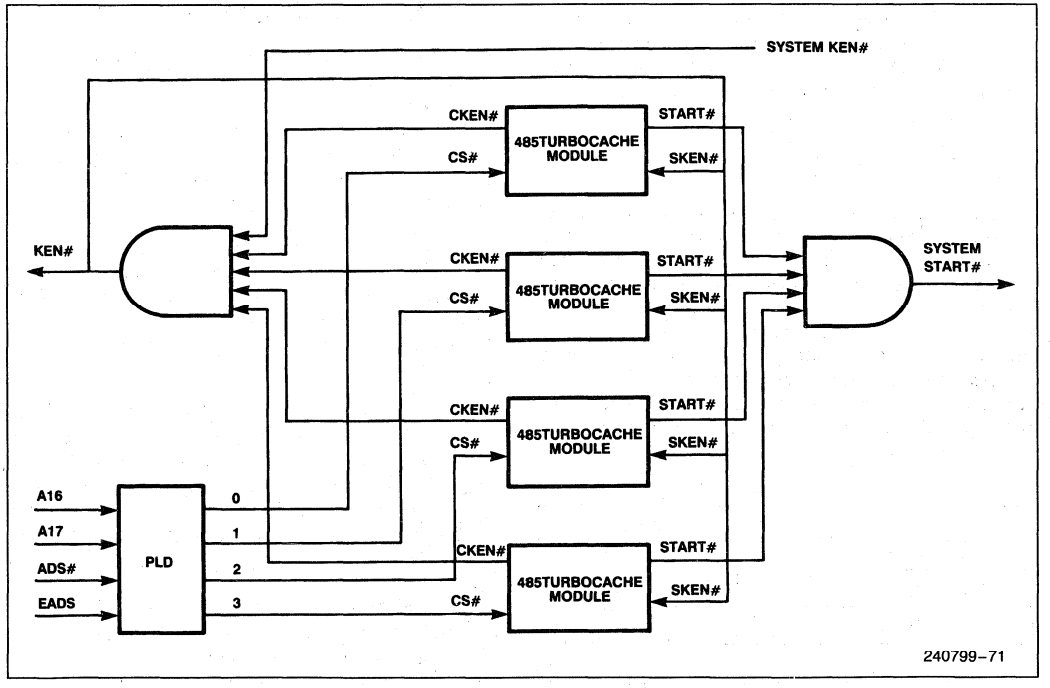


Figure 1. Multiple 485Turbocache Module Configuration

The BRDY0# output and the CBRDY# input must be used in forming of the i486 CPU's BRDY# input. Similarly, the CRDY# input must be used in forming of the i486 CPU's RDY# input. Signals that are common to the i486 CPU and the 485Turbocache Module include BOFF#, BLAST#, EADS#, BE0#–BE3#, and DP0–DP3.

The memory system generates KEN# to the i486 CPU when read data needs to be cached. The 485Turbocache Module receives this signal as the SKEN# input and produces CKEN# when appropriate. The 485Turbocache Module's CKEN# output can be used in the formation of the KEN# input to the i486 CPU. CKEN# can be used in conjunction with other logic that can deassert KEN# to the CPU when the system wants the current line fill to be cached by the 485Turbocache Module and not cached in the i486 CPU. The CKEN# signal is always asserted in T1, but is then deasserted if CS# is inactive.

The 485Turbocache Module connects directly to the i486 CPU's address lines A2–A31. The designer may have to add external buffers to the address outputs, depending upon the loading. Other signals connected to the i486 CPU include the burst control signals, the bus cycle definition signals, the byte enables, the ADS# signal, and the data and parity signals. The 485Turbocache Module and CPU connections are shown in Figure 2. The 485Turbocache Module main memory controller and bus controller interface are shown in Figure 3.

Read Hit Cycles

A read hit cycle occurs when requested data is present in the 485Turbocache Module. The i486 CPU attempts to retrieve the entire line from the 485Turbocache Module without incurring wait states. This may be accomplished by activating the KEN# input at the end of T1 (the clock in which ADS# becomes active). There is very little time to decode the address, generate the KEN# signal to the i486 CPU, and complete a zero wait state read operation. Because KEN# is sampled twice, it is possible to always assert KEN# in T1 and to wait until the end of a line fill to decide whether the data is cacheable. (See Section 3.2.)

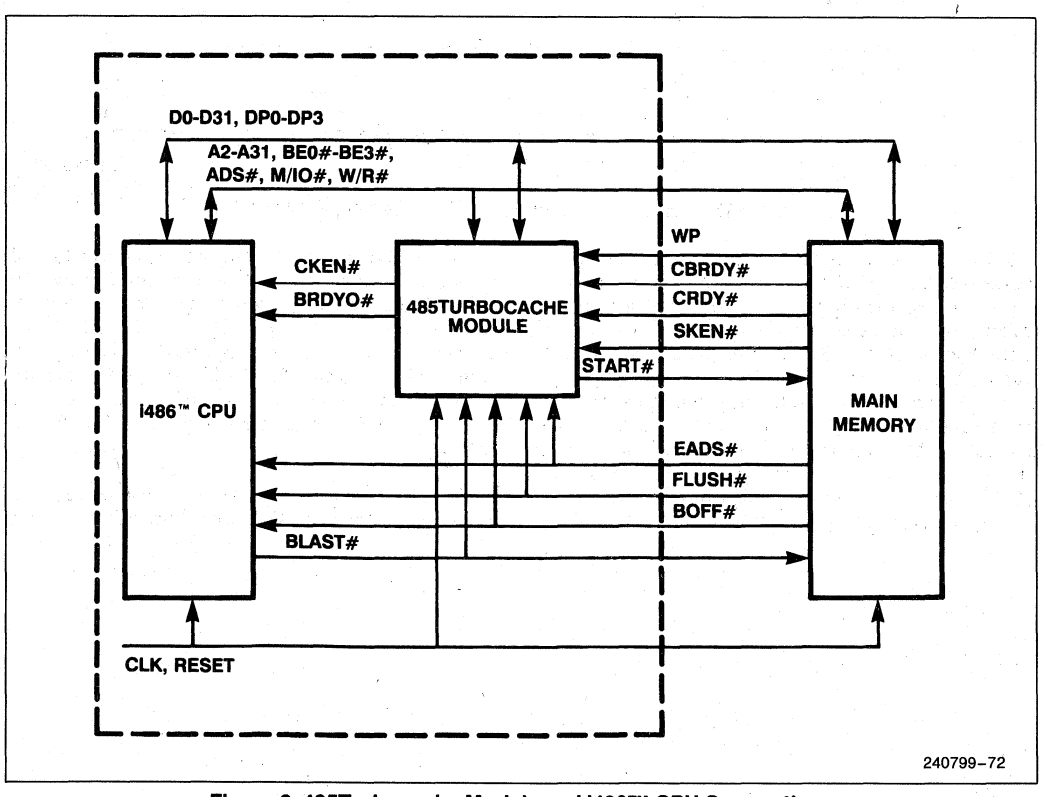


Figure 2. 485Turbocache Module and i486™ CPU Connections

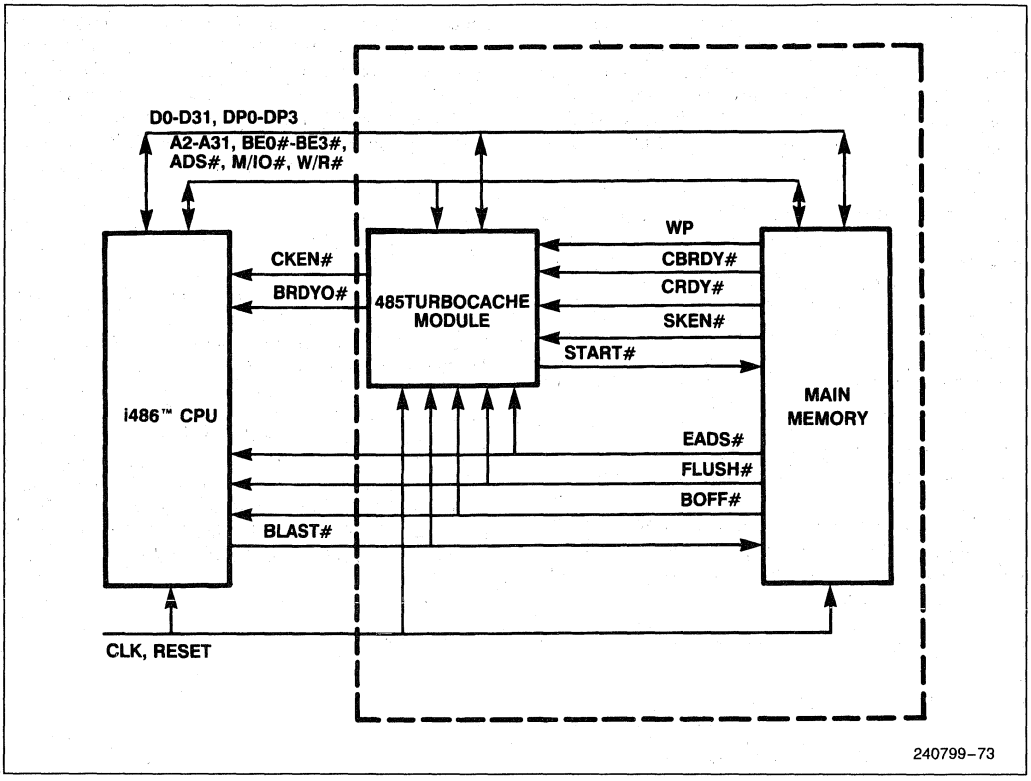


Figure 3. 485Turbocache Module and Main Memory Connections

CKEN# is used in the formation of the KEN# signal to the i486 CPU. Therefore, CKEN# is always activated in T1 (see Figure 4 and Figure 5). If a read hit occurs, data can be sent to the i486 CPU in zero wait states and can still be cached in the processor's on-chip cache. The 485Turbocache Module asserts CKEN# which remains asserted for the duration of the read hit cycle (unless WPSTRP# is low and the line is write protected). This means that the i486 CPU will cache the entire line unless external logic is added to cause the KEN# signal to be sampled high in the clock before the last BRDY0# from the 485Turbocache Module.

If the CKEN# input from the 485Turbocache Module is connected directly to the KEN# input of the i486 CPU, then the CPU will always sample KEN# active at the end of T1. To deassert KEN# to the processor, the system must create another signal that is used in the formation of the i486 CPU's KEN#, and the 485Turbocache Module's SKEN#. Using this technique a non-cacheable, non-burst cycle can be performed.

The BRDY# signal to the i486 CPU can be generated from many sources. Therefore, the various signals

should be logically "ORed" to generate the actual i486 BRDY# input.

On a cache read hit, the 485Turbocache Module generates a BRDY0# signal for each of the doublewords it transfers. The 485Turbocache Module asserts BRDY0# in the first T2 cycle, and BRDY0# remains asserted for the duration of the burst. If the i486 CPU either terminates a burst early or fails to generate a burst cycle as defined by BLAST#, the 485Turbocache Module will deassert BRDY0# after the i486 CPU has sampled the required data.

Write Cycles and I/O Cycles

The 485Turbocache Module is a write-through cache, so main memory is updated with every write hit or miss. The 485Turbocache Module is not required to generate a ready signal to the i486 CPU for write cycles. However, it does perform a comparison and updates the cache memory when a write hit occurs (provided the location isn't write protected). The 485Turbocache Module is not updated on write misses. The timings for write operations are shown in Figure 4 and Figure 5.

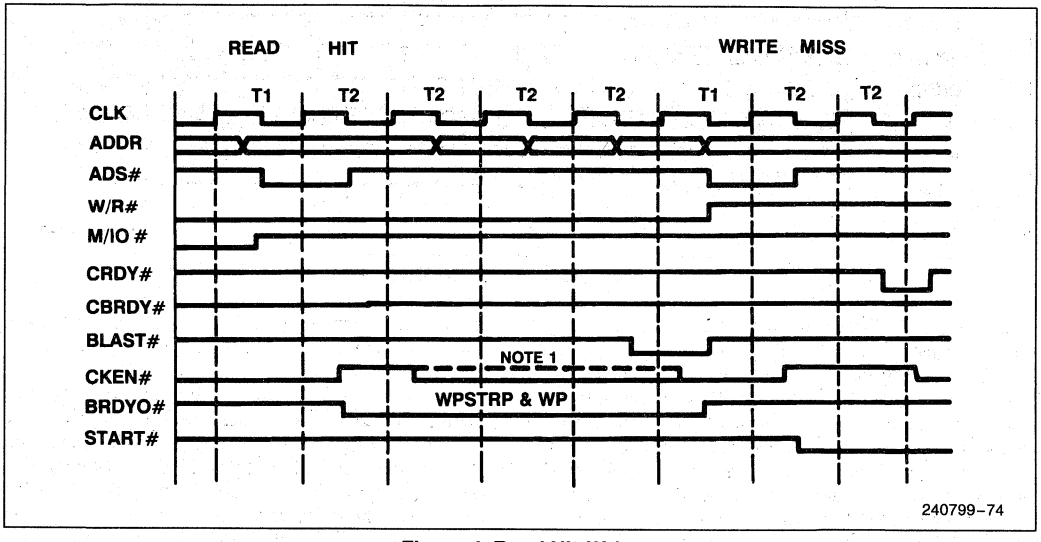


Figure 4. Read Hit-Write

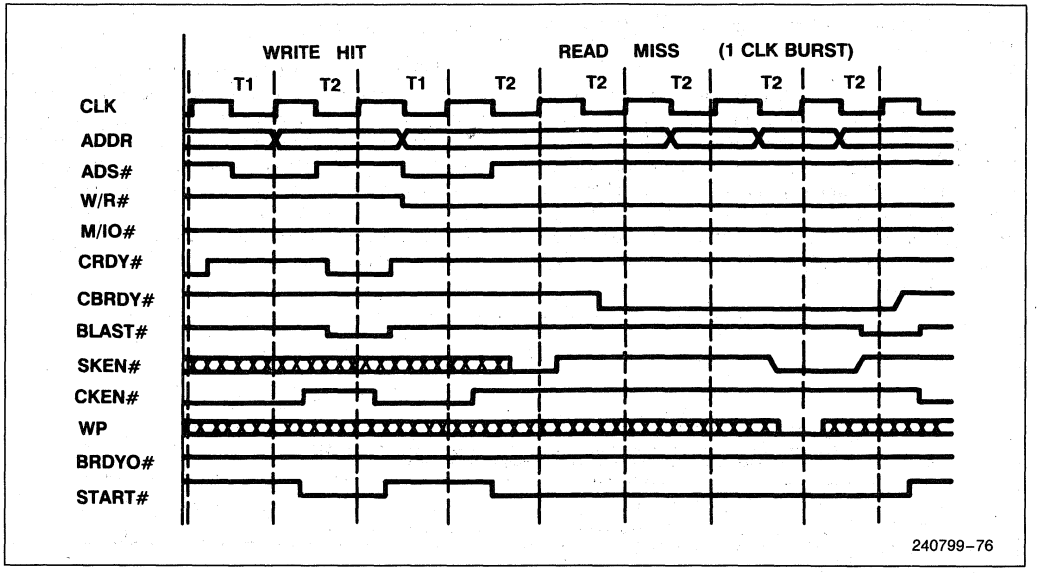


Figure 5. Write-Read Miss

Because the 485Turbocache Module is a write-through cache, writes are immediately forwarded to the system. If a processor write occurs on a valid entry that is not write protected, the new data will be stored into the memory in zero wait states. The 485Turbocache Module will not generate a ready signal. It is the systems's responsibility to update the system memory on all writes and to terminate all cycles with a ready signal. Even after the 485Turbocache Module has completed its internal write update, it remains idle until the system returns a ready to the processor.

A cache location can be write protected by asserting the WP input to the 485Turbocache Module. The WP signal must be valid during the third BRDY0# or RDY# of a cache line fill cycle. It sets a state bit within a particular cache location and remains in effect until the bit is invalidated. Tieing WPSTRP# low will not allow the write protected entry to be cached by the i486 CPU in subsequent accesses. The entry can be invalidated by any of the following: a flush operation, a reset operation, an invalidation cycle, or an LRU replacement.

When an i486 CPU cycle produces a write hit to a write-protected 485Turbocache Module location, data in the cache is not modified. The 485Turbocache Module responds in the same way whether or not a write hit location is write protected by asserting the START# signal. It is the designer's responsibility to prevent inconsistencies between the 485Turbocache Module and main memory when using the WP signal.

The 485Turbocache Module ignores all I/O cycles. When an I/O cycle is executed by the i486 processor, the system responds and terminates the cycle. The 485Turbocache Module does not assert the START# signal for I/O accesses, and the system should monitor the M/IO# signal rather than wait for the assertion of the START# signal.

System Cacheability Indication

The 485Turbocache Module uses the cache enable scheme of the i486 CPU. A cache update to the 485Turbocache Module requires activating the SKEN# signal. The signal is sampled twice, first on the rising clock edge before the first ready signal from BRDY# or RDY#, and again on the rising clock edge before the last ready. If SKEN# was deasserted at either of the specified sample times, then the access is considered non-cacheable. SKEN# is ignored during write cycles.

Typically, the system will use the same logic to generate the i486 CPU's KEN# signal and the 485Turbocache Module requires activating the SKEN# signal. However, it is not necessary for both to be asserted during an access. It is possible to use different cacheing maps for the CPU cache and the 485Turbocache Module cache because the i486 CPU and the 485Turbocache Module maintains their own cache contents via snooping.

Cascadable Cache

The 485Turbocache Module can be cascaded to configure a deeper cache memory for the processor. Up to four can be used to provide as much as 512 kbyte of cache.

System Control Signals and Cascadable Caches

The START# signal used by memory is the logical OR for each individual 485Turbocache Module START# output. If any cache has information that is needed by the processor, then its START# signal is at a high level, and it inhibits the main memory START# signal (as there is no need to access the main memory). If needed data is not present in any of the 485Turbocache Modules, then the START# signals are low, and main memory data is accessed.

The KEN# input to the i486 processor should be a logical OR for each of the 485Turbocache Modules and for a memory controller output. The memory controller output can be asserted high to indicate that the information to the i486 CPU is non-cacheable.

The SKEN# signal is the cache input to the 485Turbocache Module. The memory controllers must assert SKEN# when a transfer to the 485Turbocache Module is cacheable. The SKEN# inputs for all of the 485Turbocache Modules must be tied together. The controller that has its CS# asserted determines which cache will receive the information.

The EADS# signal from the memory controller must be connected to the i486 CPU and to all of the 485Turbocache Modules. In this way, invalidation cycles are executed in all the 485Turbocache Module devices simultaneously.

The entire memory space is covered in a single cache or a cascaded cache configuration. When multiple 485Turbocache Modules are used, only one 485Turbocache Module is selected by asserting the CS# pin.

For example, TA0 through TA15 are always connected to A16 to A31. In the configuration with one 485Turbocache Module, the chip select is grounded. In the two 485Turbocache Module configurations, A16 is used to decode between the two caches. In the four 485Turbocache Module configurations, A16 and A17 are used to generate the CS# signals.

3.0 PROCESSOR FEATURE REVIEW

The improvements made to the CPU bus interface obviously impact the memory subsystem design. It is important to understand the impact of these features before attempting to define the system. This section is a review of the bus features which affect the memory interface. The features and their impact on memory system design is discussed.

5

3.1 The Burst Cycle

The i486 CPU's burst bus cycle feature has more impact on the memory logic than any other feature. It is the most significant departure from previous bus architectures. A large portion of the control logic is dedicated to supporting this feature. The second level cache is also primarily dedicated to supporting burst cycles.

To understand why the logic is designed this way, we must first understand the function of the burst cycle. Burst cycles are generated by the CPU if, and only if, two events occur. First, the CPU must request a cycle which is longer in bytes than the data bus can accomodate. Second, the BRDY# signal must be activated to terminate the cycle. When these two events occur a burst cycle will take place. Note that this cycle will occur regardless of the state of the KEN# input. The KEN# input's function is discussed in the next section.

With this definition we see that several cases are included as "burstable". Some examples of burstable cycles are listed in Table 3. These cycle's length is shown in bytes to clarify the case listed.

Table 3

Burst Bus Cycle	Size (bytes)
All Code Fetches	16
Descriptor Loads	8
Cacheable Reads	16
Floating Point Operand Loads	8
Bus Size 8(16) Writes	4 (max)

The last case shows that write cycles are burstable. In this case a write cycle is transfered on an 8 or 16 bit bus. If BRDY# is returned to terminate this cycle the CPU will generate another without activating ADS#.

Using the burst write feature has debatable performance benefit. Some systems may implement special functions which benefit from the use of burst writes. However, the 486 CPU does not write cache lines. Therefore, all write cycles are 4 bytes long. Also, most of the devices which use dynamic bus sizing are read only. This fact further reduces the utility of burst writes.

Due to these facts, the design example used here does not implement burst write cycles. In fact, the BRDY# input is only asserted during main memory read cycles and cache hit cycles. RDY# is used to terminate all memory write cycles. RDY# is also used for all cycles which are not in the memory subsystem or are not capable of supporting burst cycles. The RDY# input is used, for example, to terminate an EPROM or I/O cycle.

3.2 The KEN# input

The primary purpose of the KEN# input is to determine whether a cycle is to be cached. Only read data and code cycles can be cached. Therefore, these cycles are the only cycles affected by the KEN# input.

Figure 6 shows a typical burst cycle. In this sequence the value of KEN# is important in two different places. First, to begin a cacheable cycle KEN# must be active the clock before BRDY# is returned. Second, KEN# is sampled the clock before BLAST# is active. At this time the CPU determines whether this line will be written to the cache.

The state of KEN# also determines when read cycles can be bursted. Most read cycles are initiated as 4 byte long from the CPU's cache unit. When KEN# is sampled active the clock before BRDY# or RDY# is returned, the cycle is converted to a 16 byte cache line fill by the bus unit. This way, a cycle which would not have been bursted can now be bursted by activating BRDY#.

Some read cycles can be bursted without activating KEN#. The most prevalent example of this type of read cycle is code fetches. All code fetches are generated as 16-byte cycles from the CPU's cache unit. So, regardless of the state of KEN#, code fetches are always burstable. In addition, several types of data read cycles are generated as 8-byte cycles. These cycles, mentioned previously, are descriptor loads and floating point operand loads. These cycles can also be bursted at any time.

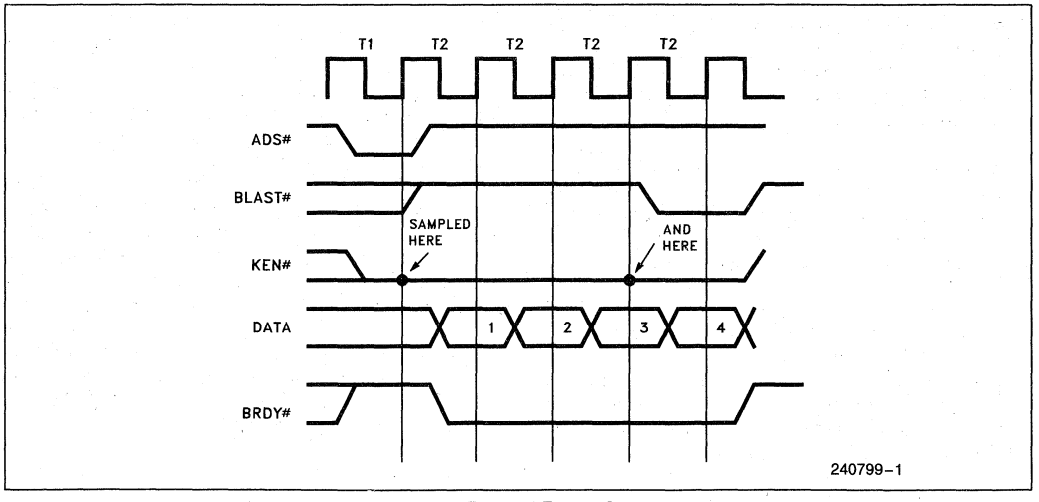


Figure 6. Typical Burst Cycle

It's obvious that the use of the KEN# input affects performance. The design example used here illustrates one way to use this signal effectively.

The primary concern when using KEN# is generating it in time for zero wait state read cycles. Most main memory cycles will be zero wait state if a second level cache is implemented. In this example, the main memory is one wait state during most read cycles. Any Cache access will take place with zero wait states. KEN# must, therefore, be valid during the first T2 of any read cycle.

Once this requirement is established, a problem arises. Decode functions are inherently asynchronous. Therefore, the decoded output which generates KEN# must be synchronized. If not, the setup and hold times of the CPU will be violated and internal metastability will result. With synchronization, the delay required to generate KEN# will be at least three clocks. In this example 4 clocks are required. In either case the KEN# signal will not be valid before BRDY# is returned for zero or one wait state cycles.

This problem is resolved if KEN# is made normally active. Figure 7 illustrates this function. In this diagram KEN# is active during the first two clocks of the burst cycle. If this is a data read cycle, KEN# being active at this time causes it to be converted to a 16 byte length. The decode and synchronization of KEN# takes place during the first two T2 states of the cycle. If the cycle turns out to be non-cacheable, KEN# will be deacitvated in the third T2. Otherwise KEN# will be left active and the data retrieved will be written to the cache.

Some memory devices may be slow enough that 16-byte cycles are undesireable. In this case more than three wait states will exist. The KEN# signal can be deactivated prior to returning RDY# or BRDY# if three or more wait states are present. As a result these slow cycles will not be converted to 16-byte cache line fills.

3.3 Bus Characteristics

The internal cache causes other effects which impact the memory subsystem design. Perhaps the most obvious of these is the effect on bus traffic. The fact that the internal cache uses the write-through policy dramatically increases the number of write bus cycles. Fig. 8 illustrates this effect. The top chart shows the bus cycle mix for an application executed with the 386DX CPU. The bottom chart shows the same application executed with the i486 CPU. The percentage of write bus cycles jumps to 70% from 30% when this application is executed with the i486 CPU.

It seems intuitively obvious that many of these write cycles would be consecutive. In fact, 70% of all write cycles are consecutive. Furthermore, 50% of all write cycles occur three in a row. It is obvious from these statistics that optimizing the memory subsystem for write cycles can improve performance. But it is important to optimize the memory system for consecutive write cycles. Improving individual write cycle latency will not buy much performance if subsequent write cycles suffer.

A technique called write posting proves ideal for this purpose. This technique allows consecutive write cycles to be overlapped. It also allows write cycles to be overlapped with second level cache cycles and reduces overall write miss latency.

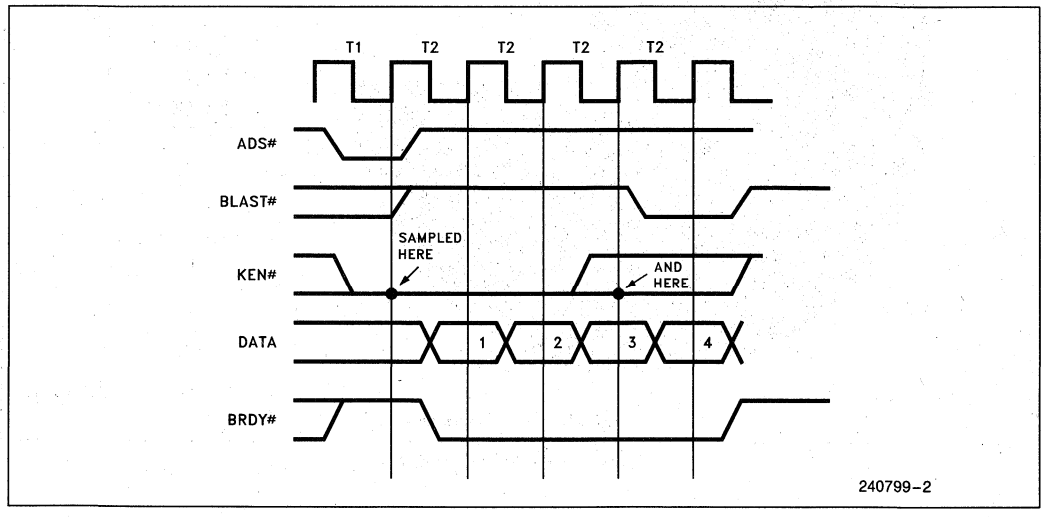
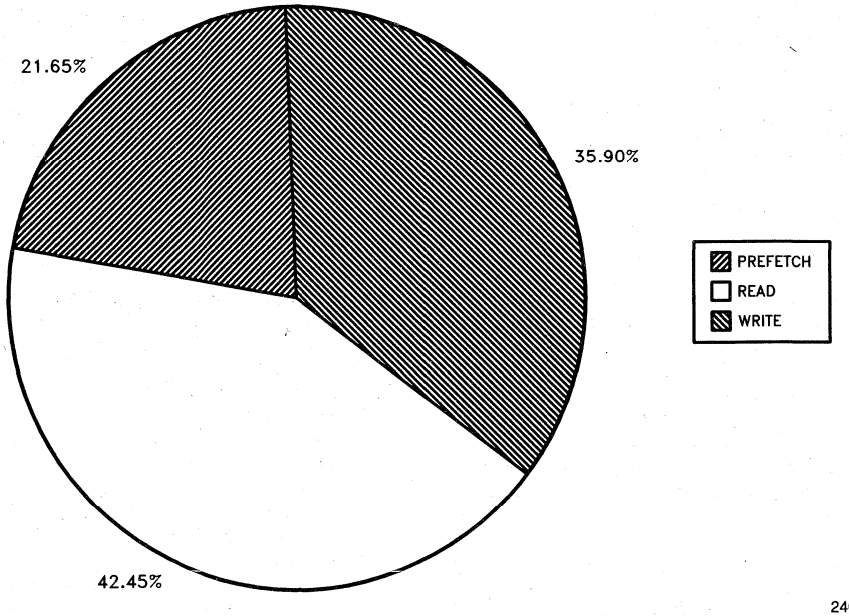
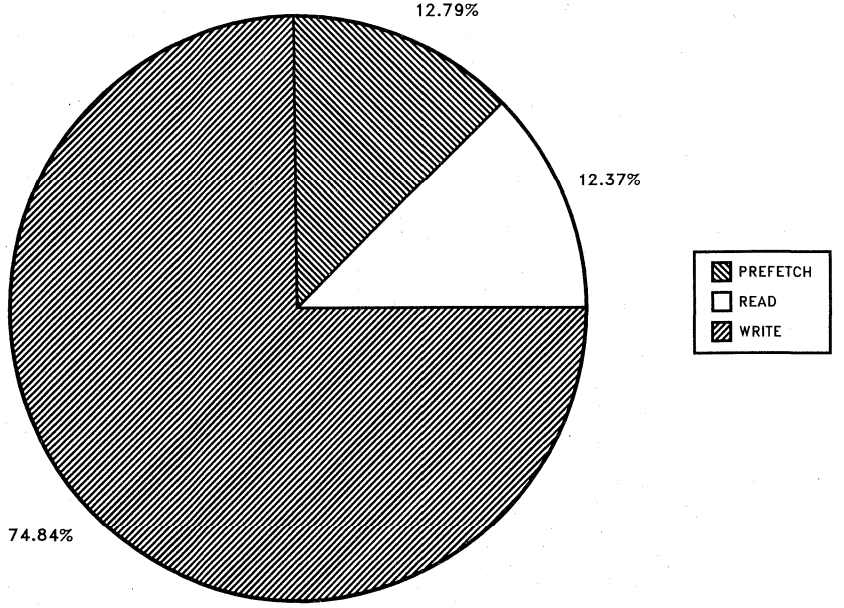


Figure 7. Burst Cycle KEN Normally Active



386DX CPU Bus Cycle Mix



i486 CPU Bus Cycle Mix

Figure 8. CPU Bus Cycle Mix

Using the write posting technique adds complexity to the system logic. It is therefore valid to ask what performance improvement is gained by using this technique. This question is especially pertinent when we consider the logic already implemented in the i486 CPU to improve write performance. The internal i486 write buffers decouple the processor execution unit from the external bus.

Analysis has shown that, in general, 6% degradation in performance can be expected for every additional wait state added to write cycles. This analysis was performed by measuring the CPU clocks required to execute several applications.

The same analysis has shown that write posting reduces average write latency to 2.5 clocks. Without write posting average write latency is 4 clocks. From this data we can conclude that approximately 9% performance improvement can be obtained by using write posting. This improvement may increase due to other affects. These affects, such as overlapping write cycles with cache reads, are discussed in subsequent sections.

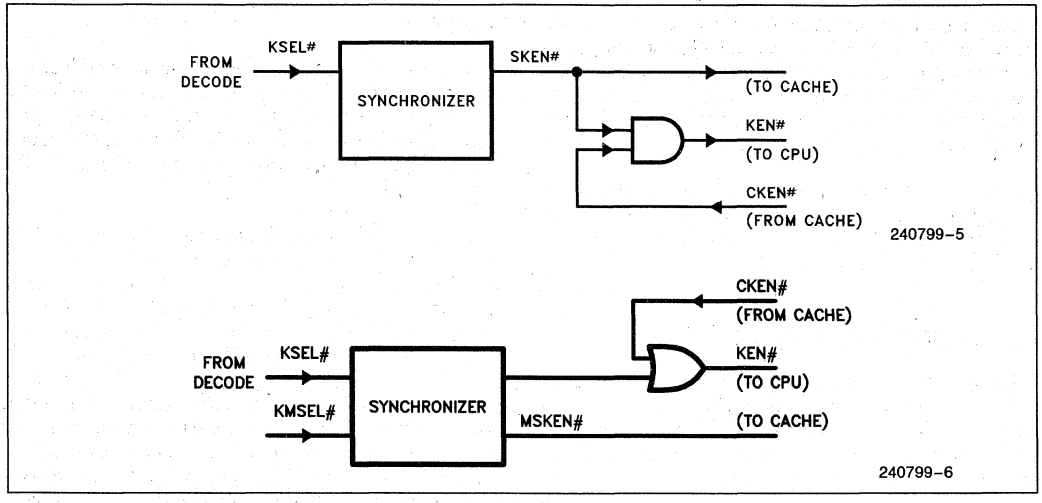


Figure 9. KEN# Logic for Second-Level Cache

5

4.0 DRAM INTERFACE OVERVIEW

The i486 CPU bus interface unit integrates several functions which improve the memory access rate. These features must be supported by the memory subsystem to provide the intended performance benefit. They are supported by the memory subsystem example. The example also includes logic support for a second-level cache. An overview of the subsystem is presented in this section. Details of the function and logic design of this subsystem are presented in later sections.

This subsystem follows a modular design. Only minor changes to particular logic sections are needed to implement variations. For instance, the PLD which generates the CAS# signal needs only minor changes to support Static Column mode DRAMs. It is also simple to implement a non-interleaved DRAM controller based on this design.

Other possible optimizations will be pointed out throughout the discussion. This first section summarizes the features and functions present in the design example presented in this section.

4.1 Functional Blocks

Two common design techniques are employed in interfacing the i486 CPU to DRAMs. The first, interleaving, is used to support the burst bus feature. The second, write posting, is used to reduce write cycle latency. Both techniques improve performance, and without them, performance is degraded by the access requirements of currently available DRAMs.

Interleaving can be implemented in several ways. Here, alternate 32-bit DRAM banks are accessed.The bank accessed is determined by the value of A2. In this way,even DWORDs (A2 = 0) are stored in one bank while odd DWORDs (A2 = 1) are stored in the other. When data is retrieved from memory during a cache line fill, cycles are overlapped to allow single clock DWORD accesses. Timing of this operation is detailed in the next section.

A multiplexor alternates data flow between the DRAM banks and the appropriate data path is selected according to the value of A2. The multiplexor prevents bus contention.

With write posting, bus cycles are again overlapped to reduce latency. Figure 10 illustrates how this technique is applied within the write cycle. The RDY# signal terminates the cycle in the clock after ADS# becomes active. This creates a zero- waitstate write cycle, the fastest possible.

When the cycle terminates, however, data must still be written to memory. The delay allows additional DRAM access time. Figure 10 shows that data is actually written to memory two clocks after RDY# is returned to the CPU. The CAS# signal completes the write cycle four clocks after it is started by the CPU.

Write data and address registers support the posted write function by holding write data and address after RDY# is returned to the CPU. These registers are required to allow the CPU to start another cycle immediately following the first (see Figure 10). ADS# is activated in the clock after RDY# is returned to the CPU. This cycle starts before the first is complete, and the cycles overlap by two clocks.

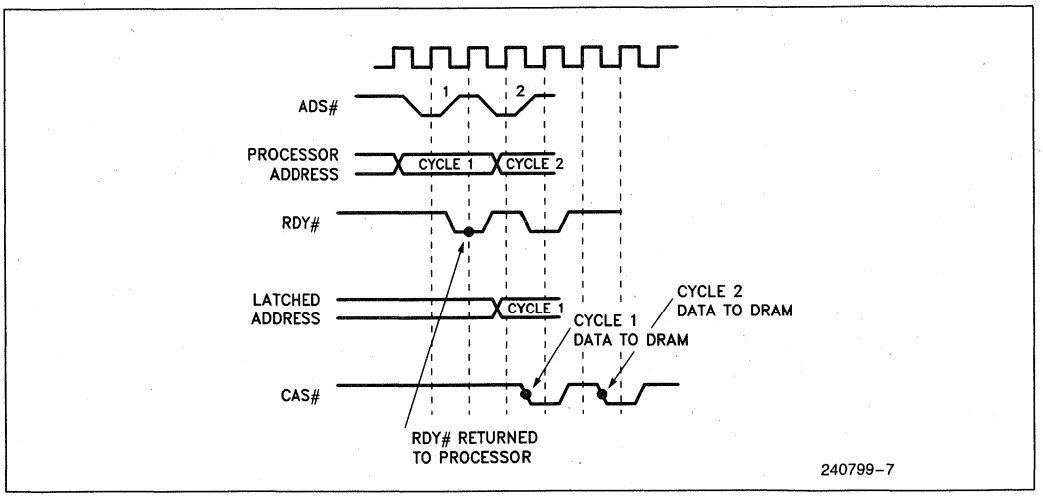


Figure 10. Write Posting

In effect the write cycle completes in two clocks. Write cycles can be overlapped in this manner indefinitely. The timing and logic required to support this function is described in Section 5.3.

Address registers also support invalidation with the AHOLD signal. They are required if AHOLD is activated when bus are cycles in progress to hold the current address while the bus cycle completes.

The efficient CPU interface and invalidation support make this DRAM subsystem well-suited for use with an optional cache. The memory system includes specific functions designed to support the optional 486 Turbocache module. The subsystem supports 256K × 4 and 1Mbyte × 1 DRAM configurations.The minimum memory configuration is 2 Mbytes with 256K × 4 devices; the maximum is 16 Mbytes with 1Mbyte × 1 devices. Additional banks can be added to increase the memory capacity.

The control logic for this example is implemented with EPLDs.The modular approach allows quick modification so that the example can be tailored for specific implementation requirements.

The control state machine is distributed among the various EPLDs, and each functional block receives control input from other blocks. In addition most of the functional blocks are implemented as state machines.

Figure 11a is a top level block diagram of the memory system. This diagram depicts the sections of logic that will be described subsequently. We will first discuss the address path logic.

4.2 Address Path Logic

Unlike processors without on-chip caches, the address bus of the i486 processor is bidirectional. The address pins serve as inputs whenever external memory is changed by DMA or another CPU. The address is driven into the CPU to invalidate the corresponding cache entry if present.

Invalidation of the 486 CPU's internal cache can be performed in several different ways. This example supports invalidation cycles during a memory access.

As described in the previous section, AHOLD is used to perform the invalidation function. AHOLD tristates the 486 address bus. Address registers must be used to hold the address to allow the current bus cycle to be completed. These registers hold the current address when AHOLD is activated.

The registers shown in Figure 11b hold the entire row and column address, as well as the current byte enables and control definition. These signals are latched at the rising clock edge of the first T2 of a bus cycle. They must be held from this edge to allow zero wait state write cycles.

5

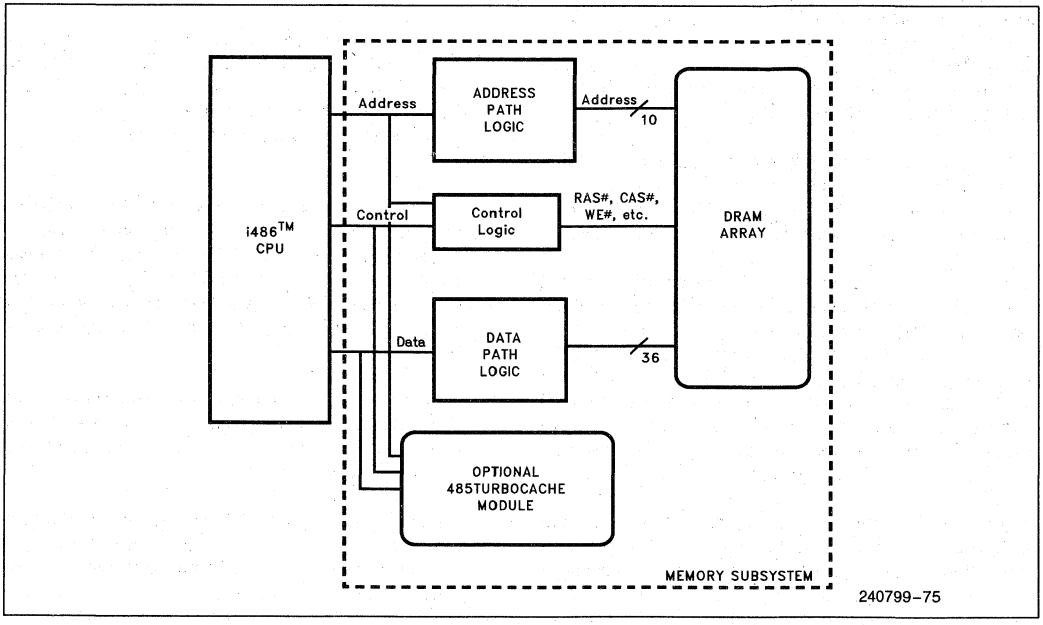


Figure 11a. Memory Subsystem Block Diagram

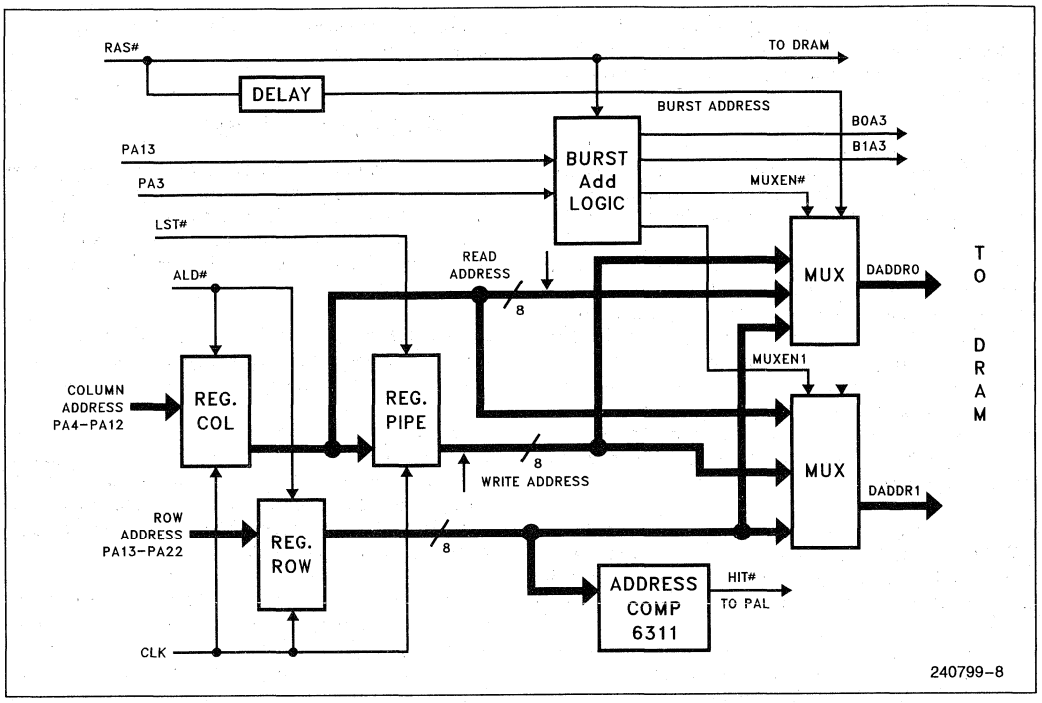


Figure 11b. Address Path Logic

Registers with enable inputs are needed. The enable input can select the CLK edge appropriate for latching the address and control state. The control logic generates the enable signal ALD which disables the CLK input of the registers during a bus cycle. When ALD is active (High) the current row and column addresses are held in the registers. 74AS823 registers have enable inputs and are used in this example.

An additional address register is required for posted write cycles. This register holds the write column address. The address is latched only on write cycles and is held until the write cycle completes at the DRAM.

Separate write and read address paths are implemented with a 3 to 1 address multiplexor. The read address path is required to meet the timing of a three CLK read cycle. In this case the read address must propagate through the address mux one CLK sooner than the write address. If the initial read access is 4 CLKs long the read and write address paths can be combined. See section 5.1 for a complete description of read cycle timing. The third address path is for the row address.

A delay line is used to meet the row address DRAM hold time requirement(tRAH). The RAS# signal is delayed 20ns to create the DRAS# signal. This signal is used as the multiplexor path select input. When DRAS# is inactive (high) the multiplexor always selects the row address path. When DRAS# is active (low) the mux enable signal (MEN0# or MEN1#) controls whether the read path or the write path is selected.

The comparator and register combination is connected to the row address path to generate the HIT# signal. This signal indicates that the current cycles address is in the same DRAM row as that of the previous cycle and also determines whether RAS# will be deactivated.

In this example a standard component designed specifically for this purpose is used. This component contains a register and a comparator. The register in this component holds the previous row address. When a bus cycle occurs to a new DRAM row, the new row address is latched. The RALE signal enables the row address latch.

The timing of this component meets the requirements of a 33 MHz CPU clock. Discrete registers and comparators can be used to improve the timing of the HIT# signal, if desired.

The last important address logic component is the burst address generator. This state machine generate A3 and A2 during burst accesses and is needed to achieve zero wait state performance during burst cycles. It predicts the value of A2 and A3. Section 5.6 contains a complete description of the burst cycle timing.

Note that because interleaving is used, A3 is the lowest order DRAM address. Two A3 equivalent signals are generated. One for Bank 0 (B0A0) and one for Bank B1A0. These signals are connected directly to the DRAM devices to meet critical timing requirements. The signals must also reflect the lowest order row address during miss cycles. As a result A13 is, therefore, an input to this logic. It is the lowest order row address when 1MBx1 DRAMs are used.

4.3 Data Path

A2 must also be predicted during burst read accesses. For this purpose, the burst address logic creates the DATASEL signal. DATASEL reflects the value of A2 for each access of a burst cycle and is used to control the data multiplexor as shown in Figure 12.

During burst cycles, the data multiplexor alternates between the bank 0 and bank 1 data paths. A2 must alternate states each clock for interleaving to function properly. The i486 CPU's burst address sequence is defined such that A2 changes state on every access.

A2 also selects the bank to which data is written. Data path logic is not involved in steering data during writes. Figure 12 shows separate data registers for each bank. Separate registers are only required to divide the data paths. These registers hold the same write data on every write cycle. The CAS# and WE# (write enable) signals control doubleword and byte steering.

Because of write data timing, the data registers must have the enable function. This function, can be used to select the clock upon which data is latched. The processor clock can be used as the register clock input to guarantee proper data setup and hold times.

As Figure 12 indicates, the MRDY# signal enables the write data registers and terminates memory write cycles. Data is therefore latched during the last clock of any write cycle.

MRDY# is restricted to write cycles while the MBRDY# signal is used for read cycles. The need for these signals illustrates the convenience of the CPU's dual-ready inputs. The MBRDY# signal enables the output of the data path multiplexor to prevent bus contention.

These ready signals are combined with similar system logic signals to form the processor RDY# and BRDY# inputs. I/O, peripheral and other non-burst devices can use the RDY# input. Burst devices, such as a second level cache controller must also use the BRDY# input. The MBRDY# and MRDY# signals are, therefore, used only with the DRAM control logic. They are isolated from the rest of the system by combinatorial logic.

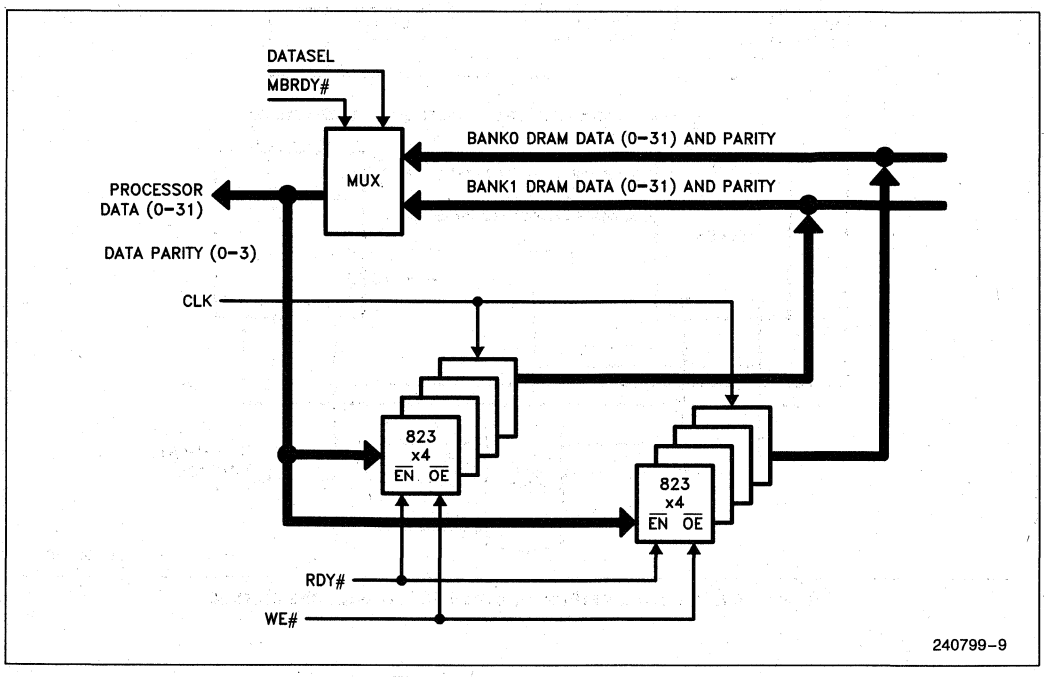


Figure 12. Data Path Logic

4.4 Second Level Cache Support

Second level cache strategies for the i486 CPU are diverse and application dependent. The example described illustrates a second level cache strategy that is ideal for single CPU systems.

The 485Turbocache second level cache used in this example is optional and is used to complement the i486 internal cache to improve the performance when running complex applications and operating systems. Some users will not require the extra performance. Since the cache is optional, O.E.M.'s or end-users can decide whether it should be included. System board design and manufacturing costs are thus eased since one system board supports multiple performance requirements.

The 485Turbocache is a completely self contained cache module. Optionality is accomplished by including control logic, tag ram and data ram in one package. A socket is added to the system board in much the same manner as a math coprocessor socket. In systems which, for example, run UNIX, the cache module is simply plugged in.

This option must, of course, be supported by the system logic. Specifically, the memory control logic is directly interfaced to the cache module. The DRAM controller example described here is particularly well- suited for this cache configurations.

The support included in the 485Turbocache module's memory control logic for the 485Turbocache module is illustrated in Figure 13. Since the 485Turbocache is a write-through cache, provision must be made for read cycles. When read data is found in the second level cache, the cycle is called a cache hit. At the time this cycle is determined to be a cache hit, it has already been started in the DRAM controller. This cycle must be aborted by the DRAM controller.

The BRDYO# signal from the 485Turbocache module provides a convenient cache hit indication. This signal is included in the decoder function. When a cache hit occurs, the DRAM controller aborts the cycle. The memory chip select signal is not activated and the first level control logic is reset aborting the cycle. The control logic then waits for another cycle to start. This function is very similar to the back-off function.

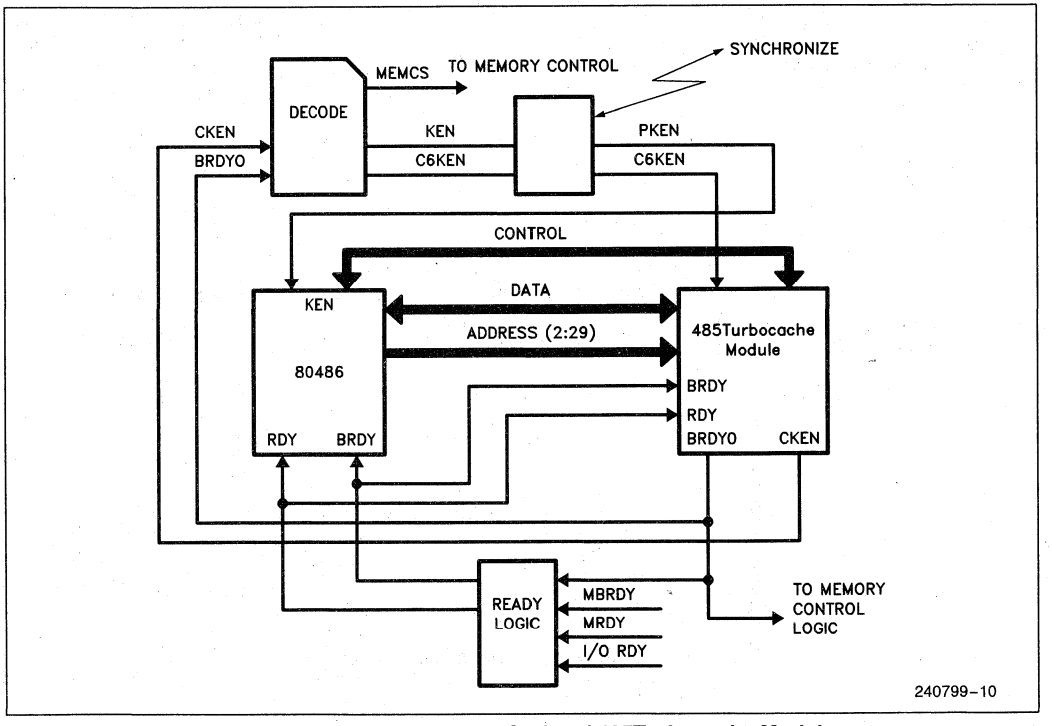


Figure 13. Logic Required for Optional 485Turbocache Module

Like the i486 internal cache, the 485Turbocache module supports non-cacheable memory by decoding. The SKEN# input is analogous to the i486 CPU's KEN# input. This function is also supported by the decode logic. Note that, as with the KEN# signal, SKEN# must be synchronized to the CPU clock.

Separate cache enable inputs also allow areas of memory to be noncacheable in the i486 CPU internal cache yet cachable in the second level cache. This feature is convienient for BIOS.

4.5 Control Logic

Memory control logic generates the signals that control the memory devices, multiplexors, and registers described earlier. These control signals can be generated in a variety of ways. This example employs a distributed state machine.

Since this example is a prototype, PLDs were the logical choice for the controller implementation. Because the number of terms in a PLD is limited, the state machine implementation must be distributed. Function distribution was determined based on this constraint. Figure 14 shows a block diagram of the controller, with each block made up of one or two PLDs.

There are two levels of logic in the controller shown in Figure 14. The first is made up of two PLDs, one which tracks bus cycles and another which generates the MRDY# signal. The first level signals to PLDs in the second level that a cycle has started. The second level is made up of several PLDs which generate the actual control signals such as RAS# and CAS#.

Implementing the controller in this manner has two important advantages. First, more decode time is allowed. The cycle start signal, CIP#, is used by the second level logic to sample the decode output. CIP# is valid in the first T2 of any bus cycle. As a result, decode does not need to be valid until the end of this T2 bus state. Without this function, the decode output must be valid at the end of every T1 bus state. In this case, the time allowed for decode at 33 MHz is very short. With 7-ns PLDs, the time allowed for decode would be 7ns. With 5-ns PLDs, this time is still only 9ns. The advantage of the extra clock period is clear.

The second advantage of the two level approach is similarly clear. The AQ0 signal indicates the start of a bus cycle to all second-level PLDs. Without this signal ADS# would have to be connected to these devices, and the resulting load on ADS# would be prohibitive.

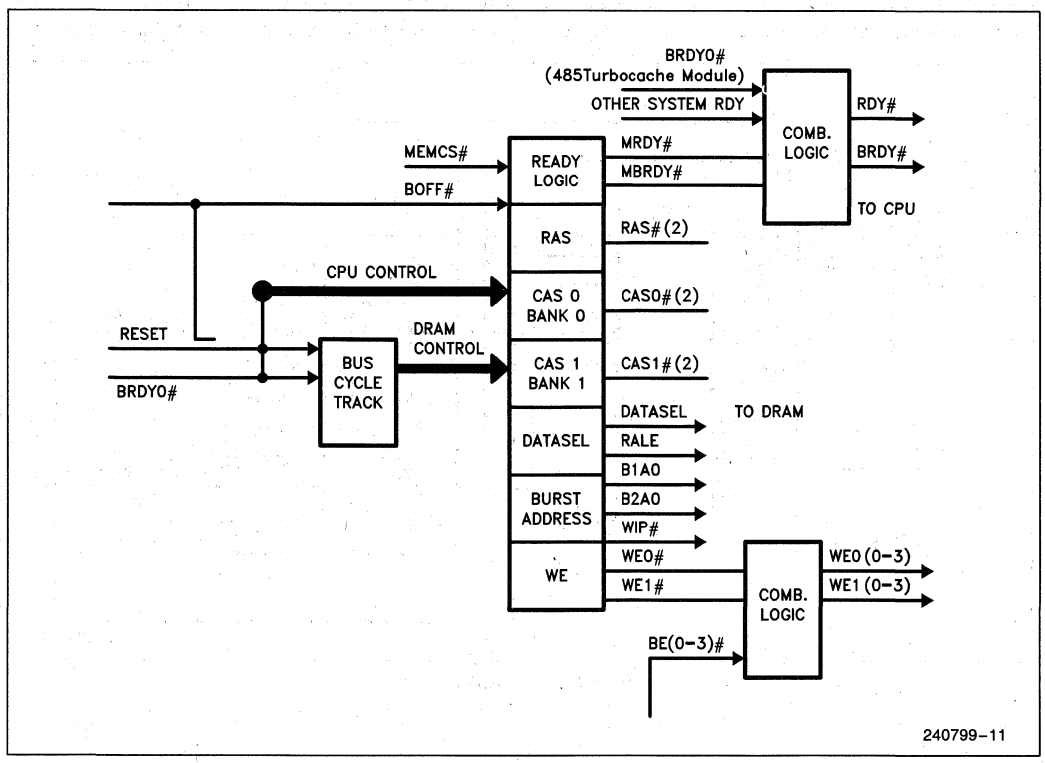


Figure 14. Control Logic Overview

Invalidation within bus cycles is another case that makes decode design difficult. The AHOLD signal must be used to implement this function. As its name implies, AHOLD can be active in any clock. If AHOLD is active in the first clock (T1) of a bus cycle, the CPU address lines are tristated in T2. Unless decode is latched at the begining of T2, it will not be valid for the DRAM cycle.

The two-level approach allows decode to be a transparent function. The decode circuit is shown in Figure 15. The 85C508 shown here includes a flow-through latch function. Using this function, the decode outputs can be latched. The DALE signal is generated at the beggining of the first T2 of any bus cycle. This signal activates the latch input of the 85C508. In this manner, decode is held during T2. If AHOLD is active in T1, the decode outputs may not be valid in T2. In this case, the cycle must not be started until the CPU address is redriven. Cycle-tracking PLD handles this function. By delaying the cycle start signal, the DRAM cycle is delayed. When AHOLD is deasserted, the CPU redrives the address again. At that time, CIP# is activated and the cycle begins. If AHOLD is active in any other clock, the bus cycle can continue normally.

The first level of interface with the memory subsystem, the cycle tracking PLD handles many other functions, most of which relate to synchronization. Refresh synchronization is one example, as is determining the RAS# precharge duration. AQ0# is not the only signal which supports the AHOLD function. Address registers, controlled by the PLD, generate the ALD signal to disable the registers during bus cycles. These and other functions of the control logic are described completely in Section 5.11.

The PLDs in the next level of logic perform more specific functions. RAS# and CAS# are generated at this level, and the PLDs that generate these signals are devoted solely to this function. The RAS# PLD generates four RAS# signals, RAS0#–RAS3#. These signals are identical but drive different DRAM modules to reduce the load on the RAS# signal.

The RAS# function is designed to support page or static column mode memory devices. To support these devices, RAS# must be left active between accesses to the same row. The RAS# state machine is designed so that RAS is deactivated only for a refresh or page miss cycle. This module generates RAS# for both DRAM banks.

For the CAS# function, the PLD's are responsible for implementing burst accesses. During write cycles, the CAS# signals determine which DRAM bank is written to. All even doublewords (A2 = 0) are stored in bank 0 while odd doublewords (A2 = 1) are stored in bank 1. When data is retrieved from memory, cycles can be overlapped. to allows zero wait state burst accesses.

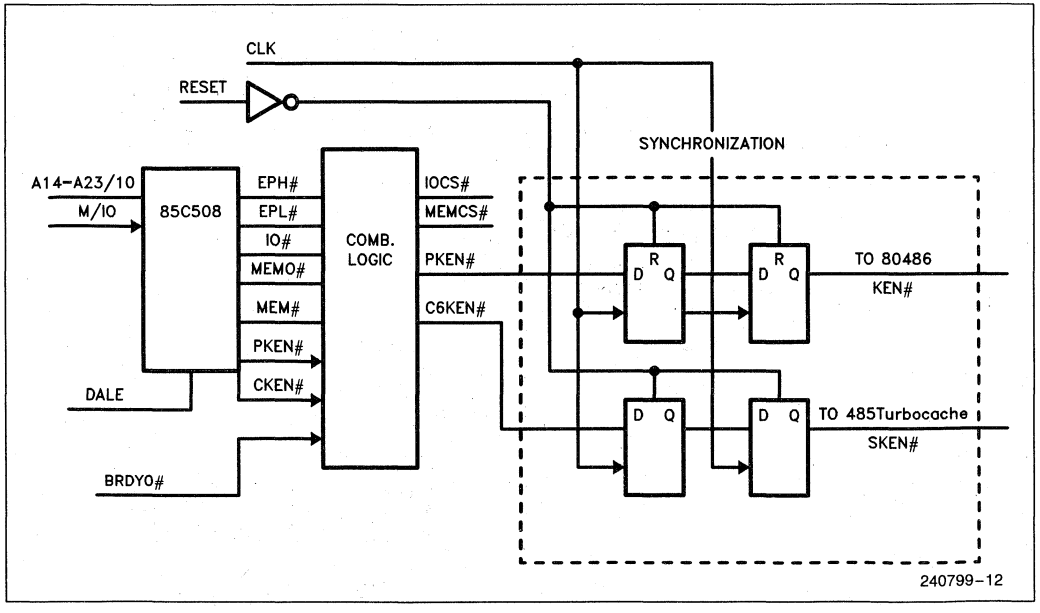


Figure 15. Decode Logic

Address generation is another important consideration in burst accesses. The address for the last three access of a burst must be generated by logic because the CPU cannot generate these addresses in time to allow zero-wait state accesses. The burst address logic shown in Figure 14 is actually two PLDs which generate the burst address for bank 0 and bank 1, respectively. The burst address consists of two signals- -the lowest order DRAM addresses from each PLD.

Because of timing constraints, these signals are connected directly to the DRAM devices. The burst address PLD must generate the burst address, provide the multiplexer function for row and column addresses and generate the write address. The burst address signals must, therefore, reflect the value of A13 during miss cycles. These reflect during burst read and write cycles. These signals reflect A3.

B00MA0 and B01MA0 are the burst address signals for bank 0. Two identical signals are used to divide loading. B10MA0 and B11MA0 are the burst address signals for bank 1. A detailed description of the burst address function is given in Sections 5.6 and 5.16.

The DSEL PLD main function is to generate the data select signal. As described above, this signal is used during a burst to switch the data path multiplexer. It reflects the value of A2 during burst read cycles only and is one component of the burst address. The DSEL PLD also generates the RALE signal to control the row address register described above.

BRDY# terminates all read cycles. MBRDY# is generated by the MRDY PLD and is separated from the RDY# signal to facilitate posted writes by preventing data bus contention. When a write cycle is immediately followed by a read, the read cycle must be delayed. This delay is implemented by delaying MBRDY# until the previous write cycle is complete. MBRDY# is combined with other burst ready inputs using combinatorial logic.

WIP# (write in progress) indicates to the MRDY PLD that a write is taking place, and MBRDY# is not generated unless this signal is inactive. WIP# tracks the state of the CAS# state machines.

The WE PLD generates WIP# and other signals associated with the write function. The MUXEN# signals control the address multiplexors and activate the write address path during write cycles. The WE# signals are used to create the DRAM W inputs and to implement byte steering. They are combined with latched CPU byte enables using combinatorial logic. In this way, DRAM W inputs are not active for unselected bytes. Data bus contention on unselected bytes is prevented by controling the write data register output enables.

By implementing byte steering in this way the CAS# logic is simplified. The CAS# timing path is critical during burst read cycles, and by placing the byte steering logic in the write enable path, CAS# timing restrictions are eased.

The MRDY# signal terminates all write cycles. The logic used to generate this signal is unusual because it uses the ADS# input and is therefore at the first level. This configuration is needed to implement zero wait state write cycles.

MRDY# must be active by the end of the first T2 to terminate a write cycle and maintain zero wait-state performance. To meet this restriction, it must be active during any write cycle, or before decode is available because the CPU RDY# signal must not be activated during non-memory write cycles, MRDY# is inhibited by the decode output, MEMCS#, in combinatorial logic.

5.0 MEMORY SUBSYSTEM FUNCTION

In this section we will explore the function of the memory subsystem in detail. Each of the signals will be described, and bus cycles will be illustrated to show the memory logic function.

The bus cycle description in this section is specific to this example. Signals such as KEN# and RDY#, for example, are shown as they are driven by this particular control logic. The signals are not restricted to the timing shown here.

A list of the memory control signals follows.

Memory Interface Signals

5.1 CPU Interface Signals

KEN#	KEN# is an input to the processor, indicating whether the next bus cycle is cacheable or not. This signal is a logical AND of SKEN# and CKEN# signals.
PBRDY#	PBRDY# is the burst ready input to the processor. This is a logical AND of the BRDY# signal from the system and the BRDYO# from the second level cache.

5.2 Data Path Control

DATASEL	DATASEL reflects the value of A2 during burst accesses. It is used to control the data multiplexor for bank 0 and bank 1 data paths.
MRDY#	MRDY# enables the write data registers that are used to support write posting and terminates memory write cycles.
MBRDY#	MBRDY# is used for read cycles and enables the output of the data path multiplexor.
WE0#/WE1#	WE0# and WE1# signals enable the outputs of data write registers used for write posting. Both the signals are active during a write and CAS# determines the correct bank to which the data is written.
WBE00#-WBE03#	WBE00#–WBE03# are a combination of write enable and byte enable signals. They control which byte is written into bank 0 during a write cycle.
WBE10#-WBE13#	WBE10#–WBE13# control which byte is written into bank 1 during write cycles.

5.3 Address Path Control

ALD	ALD disables the clock input to the registers that hold the row and column addresses corresponding to the current bus cycle.
MUXEN0#,1#	MUXEN0#, MUXEN1# control signals are inputs to the address multiplexors and are used in selecting the read or write paths to the respective banks.
RALE#	RALE# enables the row address latch, allowing a new row address to be latched for successive bus cycles.
DALE#	DALE# activates the latch inputs of the decode logic in the first T2 of a bus cycle and holds the decode during the bus cycle.
B00MA0/B01MA0	B00MA0 and B01MA0 are the burst address signals for bank 0. They correspond to the value of A3 during burst read cycles.
B10MA0/B11MA0	B10MA0 and B11MA0 are the burst address signals for bank 1. They correspond to the value of A3 during burst read cycles.

5.4 DRAM Interface

HIT#	HIT# is active if the row address for the current memory cycle is the same as the previous memory cycle.
WIP#	WIP# indicates that a write cycle is in progress and a read to the DRAM needs to be delayed till WIP# becomes inactive.
CIP#	CIP# indicates a memory cycle is in progress. If the current cycle is not to DRAM, CIP# is deactivated else it remains active till the end of the bus cycle.
RAS0–3#	RAS0–3# go active for a valid row address. It remains active between accesses to the same row and is de-activated only for page miss and refresh cycles.
DRAS#	DRAS# is the delayed RAS# signal to accomodate the RAS# hold time requirements.
RFRQ	RFRQ indicates that a refresh of the DRAM is required. This signal is activated every 15.6 us.
RFACK	RFACK is asserted as a response to RFRQ and indicates that the DRAM controller is ready to perform the refresh cycle. It is active during idle cycles or after the current cycle is complete.
PCHG	PCHG determines the timing of refresh cycles and RAS# precharge count.
CAS0#/CAS1#	CAS0# and CAS1# signals are active when a valid column address is present on the bus and control the bank to which the data is written into.
MEMCS#	MEMCS# is active when a read or a write is performed to the DRAM. It is the synchronized output of the address decoder.

5.5 Controller Signals

CT — CT indicates that a new cycle had started while a cycle was in progress or the refresh cycle was taking place. It is de-activated when the pending cycle is recognized.

SKEN# — SKEN# indicates if any of the caches is enabled. It is an input to the second level cache and is similar to the KEN# signal input to the processor.

CKEN# — CKEN# is the output of the second level cache. It is activated twice for a valid line fill - first to enable a 485Turbocache cache line fill and the second time to validate it.

LA2, LA313 — LA2 and LA313 are latched versions of address lines A2 and A13. LA313 is the lowest order DRAM address line. The multiplexor output reflects A3 when RAS# is loand A13 when RAS# is high.

M# — M# indicates the occurrance of a write miss.

BRDYO# — BRDYO# is a burst ready signal driven by the second level cache. It is activated when a read hit occurs in this cache.

5.6 Read Cycles

Timing Diagram 16 shows a burst read cycle. At the start of the bus cycle, RAS# is inactive. This case is a rare occurence because RAS# is normally active. Unless a cycle is the first bus cycle after a reset or refresh cycle, RAS# will be active in T1.

It is useful to examine this case because it demonstrates a complete DRAM cycle. The basic function of most of the control logic is illustrated.

The cycle begins with the activation of ADS#. The controller samples this signal and activates both ALD and CIP#. The CPU address registers are disabled by ALD. Therefore, the previously latched address is held throughout the bus cycle. The latched address is valid in the first T2 of the bus cycle.

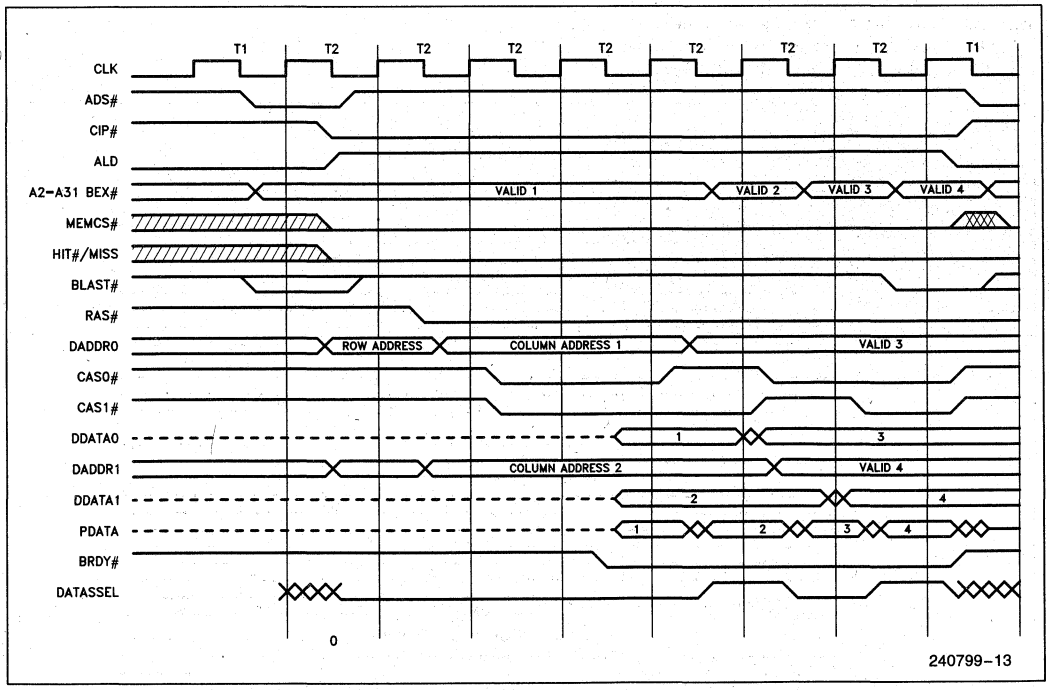


Figure 16. Burst Read Cycle

The row address comparison is made with this address. As a result, the HIT# signal is not valid until the rising edge of the second T2. At this rising clock edge, the CIP#, MEMCS# and HIT# signals are sampled. If MEMCS# is sampled active, the RAS# signal is activated.

The delay line holds the DRAS# signal high for 20 ns after RAS# is activated. In this way the row address is maintained to meet tRAH, the row address hold time. When DRAS# is activated, the address multiplexers switch to the column address path. The MUXEN# signals are not active, and the read path is selected.

In the third T2 of the bus cycle CAS# is asserted. This cycle begins with A2 low and the first access is to bank 0. Due to the access time of the DRAM two clocks are required to retrieve data from memory. MBRDY# is asserted in the fourth T2 of the bus cycle, and this action completes the first access of the burst read. The access is completed in five clocks. The minimum time for this access is two clocks indicating that three wait-states were added to the first cycle.

The timing diagram reveals two important points about burst cycle implementation. First DRAM access requires two clocks. Second, the burst address from the CPU is not available until the clock after MBRDY# is sampled active. These circumstances make implementing zero-wait-state burst cycles difficult. The DRAM bank interleaving alleviates this difficulty.

The first advantage of interleaving is revealed in the second and third T2 states. Access to both the first and second memory doublewords can be made simultaneously. This function requires that the burst address be predicted. As mentioned above, the burst address from the CPU is not available until several clocks later. The burst address for both the first and second accesses is generated in the second T2. Therefore, CAS# for both banks can be asserted in the next T2 state.

The second advantage of interleaving is seen in fifth T2 of the burst cycles in which DATASEL switches the data multiplexer. The second doubleword is driven on the CPU data bus. In this CLK, the burst address for the third access of the cycle is generated. CAS00# and CAS01# are also deasserted to begin the third access. Note that this access is started before the second access is completed. The cycle overlap shown allows new data to be driven on the CPU data bus every clock. This way zero-wait-state access is achieved.

Timing is even more critical during page hit cycles. Fig. 17 shows the timing of this cycle. Because of the function of RAS#, this cycle is more common than the cycle discussed above. The row address is the same as in the previous cycle. Therefore, the RAS# signal is left active.

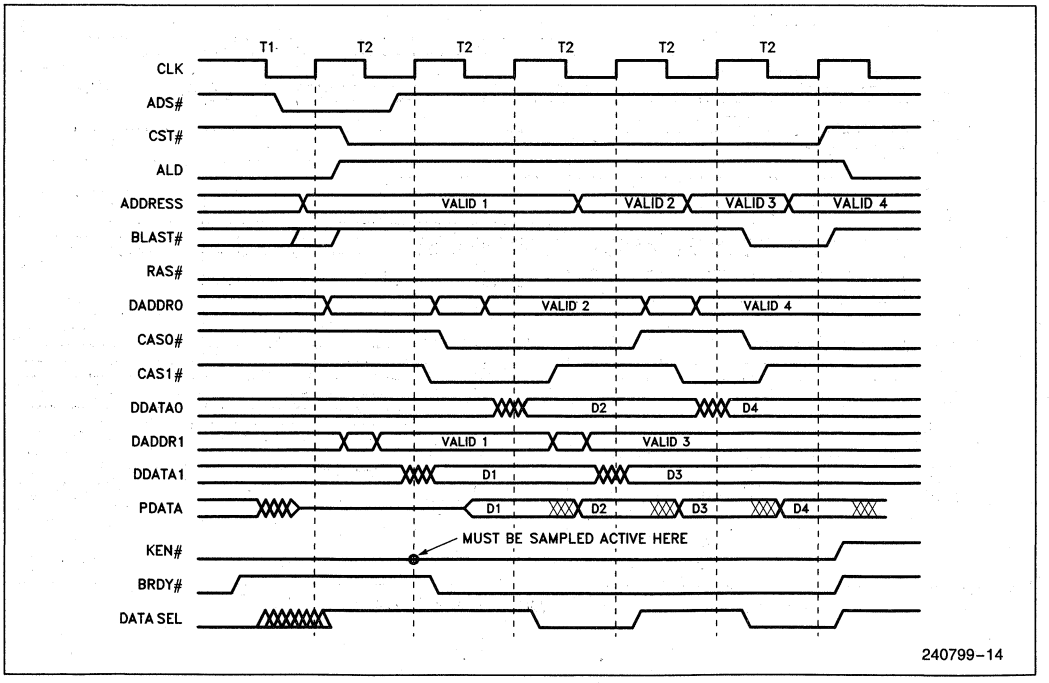


Figure 17. Burst Read DRAM Page Hit

When a burst read starts with RAS# active, fewer clock, are required to complete the first access. This reduction improves performance. As a result, however, some timings become more critical. One of these is the time allowed to generate the burst address.

The CAS# signals are asserted in the second T2 of the bus cycle. MBRDY# is also asserted at this time. To meet the address access time of the DRAMS, the burst address must be generated in the second T2. The rest of the read column address must also be available at this time. Two logic functions are needed to meet this timing requirement. First, read and write address paths must be separate to allow the read address to be available in the first T2. Second, the burst address path logic must latch the CPU A3 signal directly. In this way, the logic can generate the necessary address in time. The burst address state machine must track the state of A3 at the begining of every cycle. The state machine function is described in Section 5.11.

The timing of KEN# must also be considered in this example. KEN# must be valid at the begining of the second T2 of the cycle. If it is not, the cycle will not be cached, and a 16-byte access can not be generated. If KEN# is active, a 16-byte burst access will be generated, and the cycle will be cached as long as KEN# is active in the second to last T2.

At first glance this timing may not appear critical. KEN# is a decode function, and decode is valid at the clock edge called for. The KEN# input to the CPU must be synchronized to clock, however. Since decode is not synchronous, a two-clock synchronizer delay is required, and this delay is the reason that KEN# is normally active in this example.

From the time CAS# is activated, this cycle is exactly the same as in the previously described burst cycle. It is terminated when BLAST# is asserted, and MBRDY# is deasserted when BLAST# is sampled active.

5.7 Write Cycles

As described in Section 4.1, a posted or delayed write function is employed in this example to reduce write cycle latency. Latency is reduced since write cyles are overlapped with other cycles including other Write cycles or reads from the second level cache. Write cycles normally make up 70 percent of all cycles, and overlapping can increase performance accordingly.

Figure 18 illustrates the posted write implementation. In this example cycles begin when RAS# is inactive. As with read cycles, this case is rare in practice.

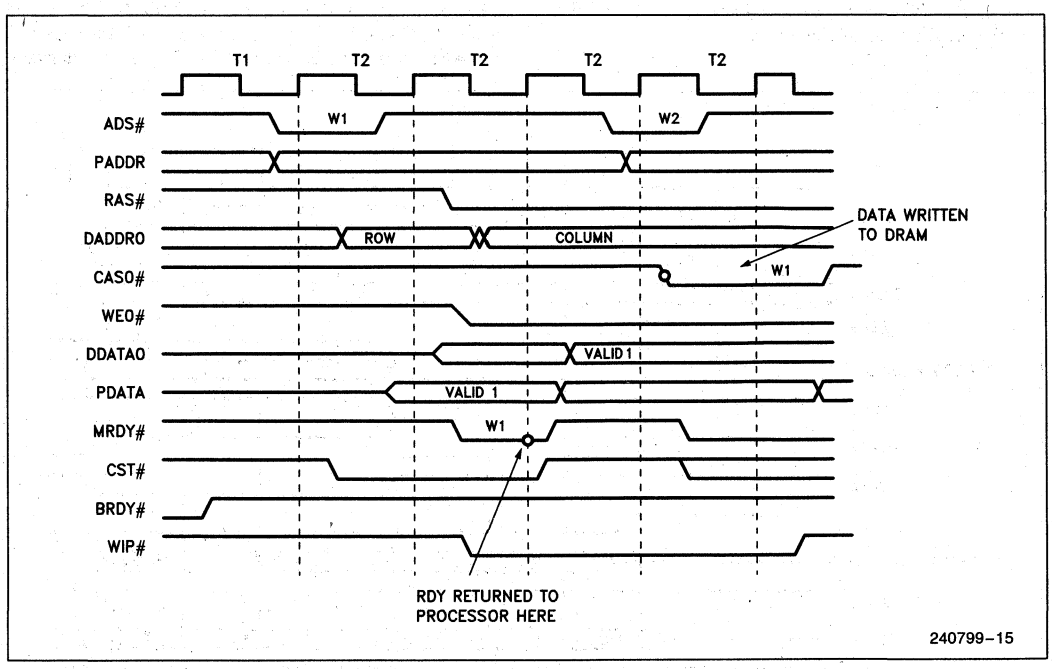


Figure 18. Basic Write Cycle

The cycle begins like a read. The CPU drives ADS# active, and the decode is sampled. RAS# is activated if the cycle is in DRAM space. In the second T2 of the cycle, however, the latched version of W/R# (LW/R#) is sampled active at the rising edge of the second T2. In response, the control logic begins several write cycle functions at this clock edge.

The CAS# state machine for the appropriate bank enters the write sequence. The MUXEN# and WE# signals are asserted. MRDY# is also asserted, terminating the cycle at the CPU. The MUXEN# signals activate the write address path. This address is not present at the multiplexor outputs, however, until the next clock at which the write pipeline register latches the write address.

The write data is latched at the same clock edge. The write data registers are enabled by MRDY# which simultaneously terminates the CPU cycle. Note that data is latched in both the bank 0 and bank 1 registers.

The WE0# and WE1# signals are also both active. The CAS# signals determine which bank is written to. These signals are asserted within two clocks after MRDY#. This action completes the write cycle. Note that, while five clocks are required clocks are required to complete the cycle, the CPU cycle is terminated in three CLKs. The wait state is only required if RAS# is inactive at the start of the cycle.

In Figure 18 the next bus cycle starts immediately after RDY# is sampled. In this case, CAS# is activated during the second clock of the next bus cycle. This overlap of cycles is similar to the pipelining feature used by many processors except that the i486 processor bus is not involved in the posting function. All logic for this function is implemented in the memory controller.

Figure 19 is a more typical i486 processor bus sequence which clearly illustrates the advantages of the posting technique. Four write cycles have occurred together without idle bus clocks occurring between cycles. Since all writes access the same DRAM row, RAS# is active throughout the sequence.

Without the extra clock to activate RAS#, MRDY# can be asserted in the clock after ADS# is asserted. These cycles, therefore, have no wait-states. As before, the write cycle is not complete when MRDY# is asserted but instead when CAS# is asserted two clocks after MRDY# to terminate the CPU bus cycle.

At zero wait-states, each write cycle still requires four clock cycles. The last two clocks of each write cycle overlap with the next cycle. The net effect on the CPU bus is the same as a string of two-clock write cycles, as illustrated in Figure 19.

The first write in this figure is to bank 0. The falling edge of CAS0# clocks the data into the bank 0 DRAM. This edge is denoted by W1 in the diagram.

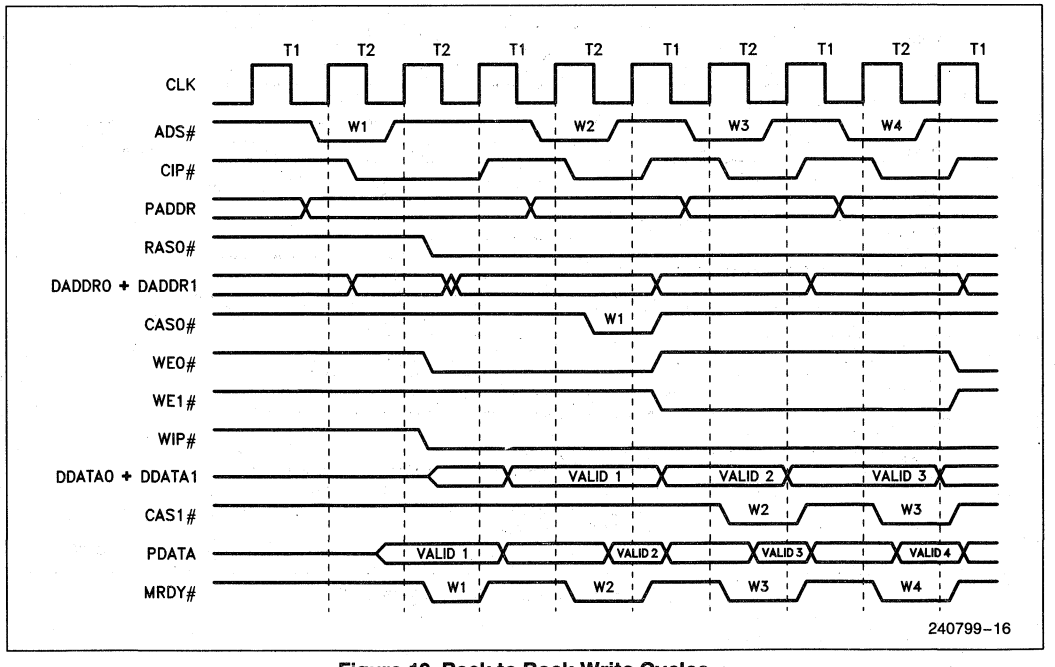


Figure 19. Back to Back Write Cycles

CAS0# is asserted in the same clock that MRDY# terminates the second write (W2), which accesses bank 1. CAS1# is activated in the same clock as MRDY# for the third write (W3).

The second and third writes happen to be to the same DRAM bank. As we see, no timing modification is required in this case. Write cycles can be completed with zero wait states in either case. This is important since writes often occur in sequence on the i486 bus, but not necessarily to sequential addresses. Write posting supports zero wait-state write cycles to sequential and non-sequential addresses.

This fact is also important if the design is to be modified. For example while, interleaved DRAMs may not be required in systems with a permanent second level-cache, the write posting technique may still be used in the system. The benefits of this technique still apply since write cycles may still be overlapped as described.

5.8 Consecutive Bus Cycles

The DRAM control logic is optimized for write cycles, as warranted by the i486 processor's bus characteristics. Over 70 percent of all cycles are writes. By employing the posted write technique, system performance is increased.

The posted write technique poses some special problems, however. Page miss, refresh and consecutive write-read cycles require special consideration. We will begin by discussing the consecutive write-read case. Page miss and refresh cycles will be discussed in sections 5.9 and 5.10.

When a read cycle immediately follows a write, the read cycle must be delayed as illustrated in Figure 20. The read cycle is delayed to allow the write to complete. Only read cycles to DRAM, i.e. (cache misses) need be delayed. Cache hits and write cycles overlap easily because the cache is on the CPU side of the DRAM controller.

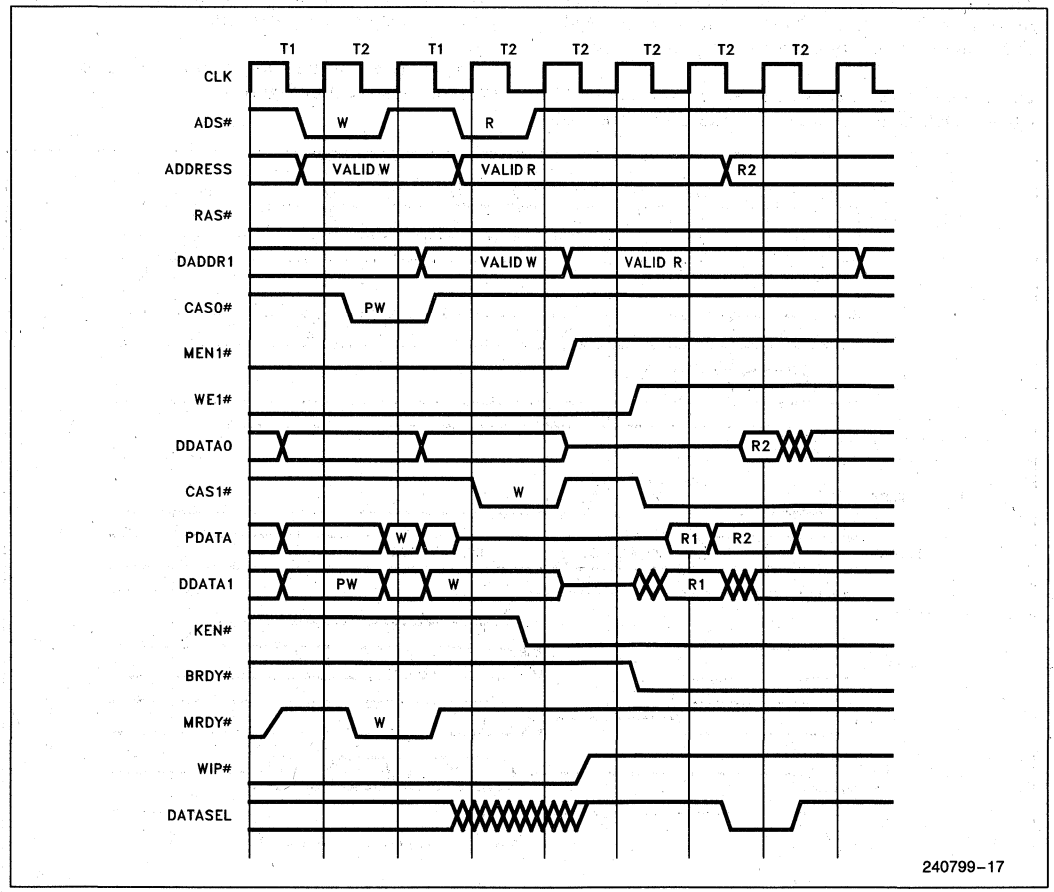


Figure 20. Consecutive Write-Read Cycle

Write cycles cannot overlap DRAM read cycles, however, primarily because of data bus contention. The DRAMs used here have common data I/O pins. In this case read and write data paths cannot be active at the same time.

To prevent data bus contention, the first data access of the read is delayed. In Figure 20 the first read access is to the same bank as the write. In addition, the read cycle accesses the same DRAM row. Two functions are required to ensure that the write is completed. First, the write address must be held until CAS# is asserted. Second, the data mux outputs must not be enabled until the CPU tristates the bus.

The first function is accomplished by the MUXEN# signals. The MUXEN# state machine tracks the CAS# function for the appropriate bank. When the write for that bank is complete, MUXEN# is deactivated. In this way, the read address path is not enabled until the CLK after CAS# becomes active. Normally, the read address would be valid in the first T2 of the read cycle; however it must be delayed one clock to allow the write complete. Note that if one or more idle CLKs intervenes between these cycles, no delay occurs.

The second function is accomplished with the WIP# signal which is active until all write cycles are complete. A read cycle to either bank will be delayed if it immediately follows a write. The first access of the read is delayed by MBRDY#, which is not asserted until the WIP# signal is deasserted.

WIP# is deasserted once all pending writes are complete. In Figure 20 the read cycle is delayed 3 CLKs by this signal; in other words, three additional wait-states are added. If a read does occur immediately after a write, the number of wait-states added will decrease by the number of idle CLKs between cycles. For example, if ADS# for the read is asserted three clocks after MRDY# for the write, MBRDY# will not be delayed.

5.9 Page Miss Cycles

As described previously, page miss cycles occur when the CPU generates a cycle which changes the DRAM row address. The RAS# signal must be deasserted to change the ROW address in the DRAMS. Any time RAS# is deasserted, it must remain high for the precharge time (tRP). A delay is added to every page miss cycle to satisfy this requirement.

For read cycles this function simply requires extra wait states as illustrated in Figure 21.

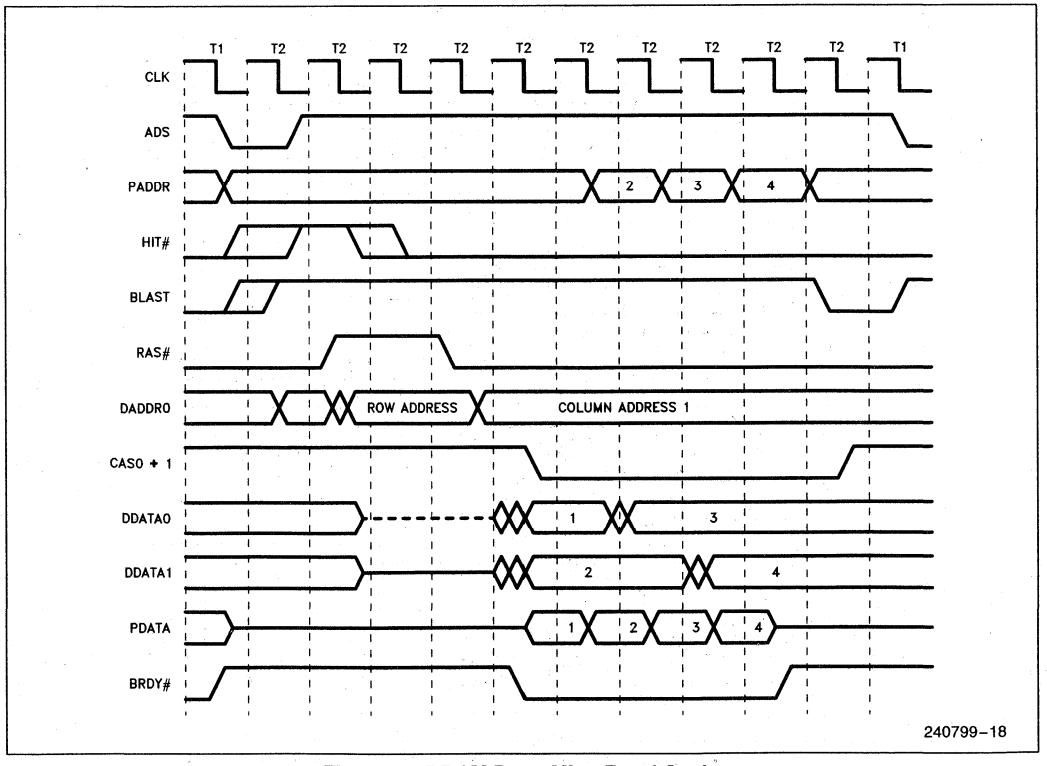


Figure 21. DRAM Page Miss-Read Cycle

The bus cycle starts with RAS# low or active. The row address generated by the CPU is different than in the previous cycle, and the row address comparator deasserts HIT#. This signal is valid in the first T2. HIT# is sampled at the RAS# PLD at the rising edge of the second T2. In response, RAS# is immediately deasserted and held inactive for two clocks. This time satisfies the RAS# precharge requirement.

Four wait states are added to process the miss cycle. These clocks are added to every read cycle which accesses a new DRAM row. The delay is accomplished, again, with the MBRDY# signal. MBRDY# will not be asserted when RAS# is inactive. Once RAS# is sampled active, MBRDY# is asserted. From here, the cycle proceeds as described in section 5.7.

Write miss cycles are more complex than read miss cycles, due mainly to the write posting technique. The added complexity results in lower latency than in a non-posted memory system, however. Figure 22 illustrates how this improvement is achieved.

The write cycle in Figure 22 also begins with RAS# active. The HIT# signal is deasserted in the first T2 at the same time that MRDY# is asserted. MRDY# could be inhibited at this point to prevent write cycle termination. The wait-states added to meet RAS# precharge time would then be added to this cycle. Five wait states are required to meet the precharge time.

The average number of write cycle clocks can be reduced, however, if another method is used. MRDY# can be allowed to terminate the cycle. In this case, any necesary wait-states will be added to the next cycle.

This method improves the average in two ways. First, some write miss cycles will not require wait- states. This is the case when the next cycle occurs four or more clocks after a write miss. In addition, wait states will be reduced when the next cycle occurs in two or three clocks. Second, three wait-states are required to complete the next cycle when it follows immediately as illustrated in Figure 22.

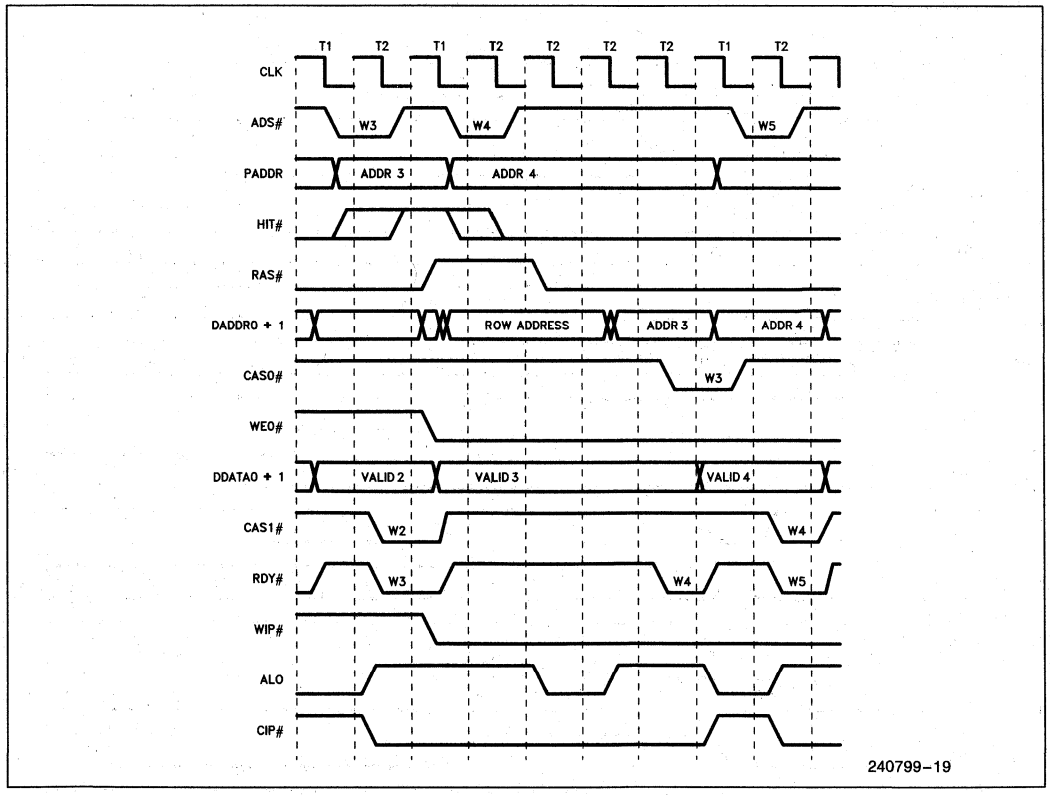


Figure 22. DRAM Page Miss-Write Cycle

The first cycle in this figure is a page miss. It is terminated at the CPU without wait-states. Because HIT# is not active in the first T2, RAS# is deasserted. At this point, additional clocks are added to perform the miss function. Part of the time required for RAS# precharge is overlapped with the next cycle. The two clock overlap reduces the number of wait-states required in the next cycle. Therefore, the average write cycle latency is reduced.

5.10 Refresh Cycles

The CAS# before RAS# refresh function is used in this example. This function uses internal counters in the DRAM devices to generate the refresh address. When the CAS# input is activated prior to RAS#, the internal counter is incremented. The output of the counter is then used as the address of the row to be refreshed.

Each refresh cycle refreshes one row of the DRAM array. The refresh cycles are distributed such that one occurs every 15.6 μs, with every row being refreshed in 8 ms. Refresh cycles are initiated by the RFRQ signal. This signal is activated every 15.6 μs by a counter.

RFACK is asserted in response to RFRQ. This signal indicates that the DRAM controller is ready to perform the refresh cycle. It also signals the counter circuit that RFRQ can be deasserted.

The function of RFRQ and RFACK is very similar to that of the CPU's HOLD and HLDA signals. RFRQ is sampled at the end of each cycle and during idle cycles. RFACK is activated in the clock after RFRQ is sampled, except immediately after write cycles.

Again, the posted write function must complete before the refresh cycle begins. If WIP# is active when RFRQ is sampled, RFACK will not be immediately asserted. RFACK will be asserted after WIP# is deactivated as illustrated in Figure 23.

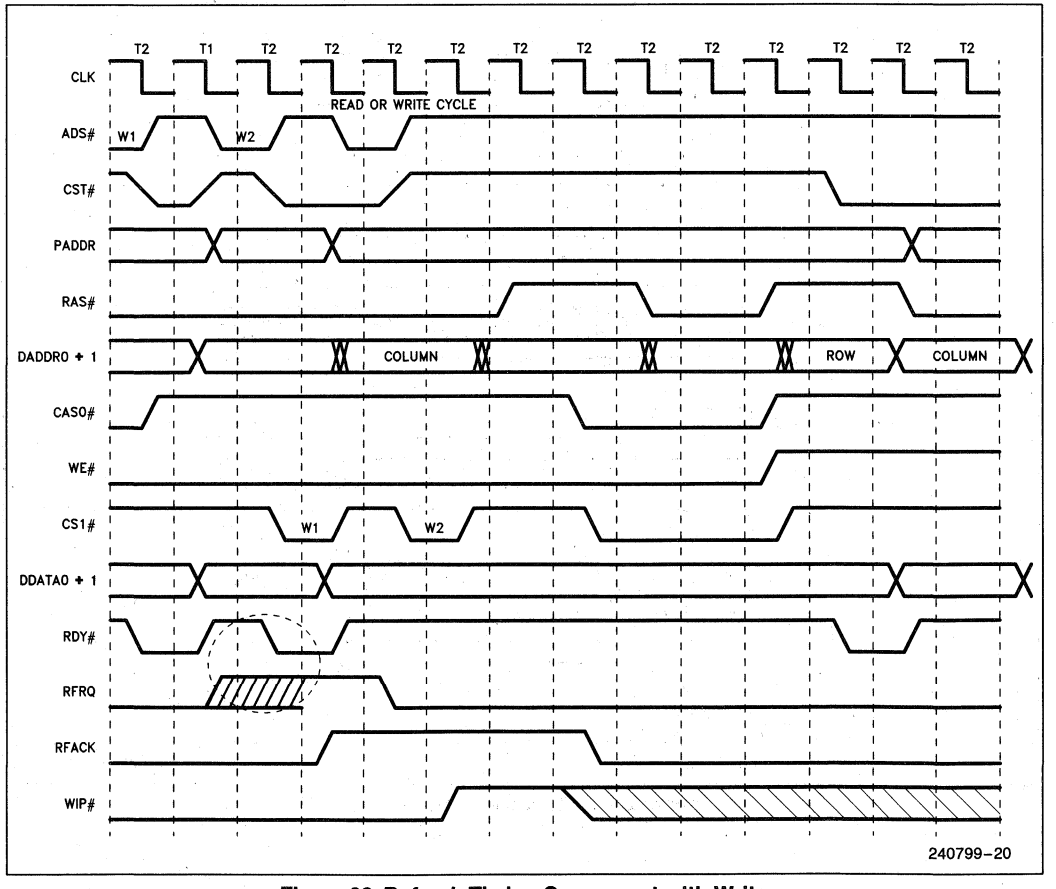


Figure 23. Refresh Timing Concurrent with Write

Another cycle can start between RFRQ and RFACK. The cycle start PLD tracks this case. GP# ~ will not be asserted for any cycle that starts during this interval. Once the refresh cycle is complete, this cycle can be started.

6.0 CONTROLLER IMPLEMENTATION

The functions described in the previous section are generated by the control logic. The controller, as outlined in Section 4.0, is made up of several PLDs. These devices enerate the control signals described in Section 5.0. The function of the logic is determined by the state machine definition. These state machines are distributed in the different PLDs of the controller.

In this section, we will explore the implementation of the control logic. The discussion will focus on the state machine definition. Certain conventions are followed throughout the discussion. These conventions are based on the state machine compiler used to generate the PLD equations. This compiler uses the exclamation point (!) to indicate the low or "0" condition of a signal. It uses the number symbols (#) to indicate that the signal is active low. For example, !ADS# indicates that the ADS signal is both low and active. The # symbol indicates that a signal is active when low. So symbol !ALD means that the ALD signal is not active. These symbols are used to indicate state transitions as shown in Figure 24. The state transition in Figure 24 depends on three signals: ADS#, ALD, and RAS#. The equation indicates that if both ADS# and ALD are active or if RAS# is not ctive at the next clock edge. the transition from S0 to S1 takes place. In the transition between S0 and S1, the Y# signal is activated. The definition of states indicates which outputs are changed in the transition. These conventions are used to describe the control state machines in the next section.

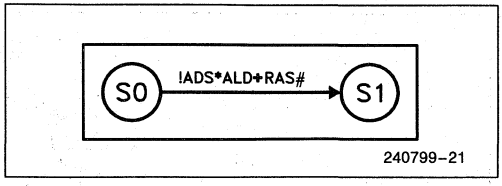


Figure 24. State Transition Example

6.1 Cycle Tracking Logic

The cycle tracking logic is contained in one PLD. The five state machines implemented in this PLD start and end DRAM cycles, control refresh timing and control the address registers. These state machines, along with the MRDY# state machine comprise the first level of control logic. All other control state machines depend on this first level to generate signals at the proper time.

The signals generated by this PLD are the following:

CIP# - Cycle in Progress
ALD - Address Latch Disable
CT - Cycle Track
RFACK - Refresh Acknowledge
PCHG- RAS Precharge Count

The primary cycle tracking state machine is shown in Figure 25. This state machine generates the CIP# and M# signals. CIP# indicates that the CPU has started a cycle. When it is active, the rest of the logic samples the CPU control and MEMCS# signals. If the current cycle is not to DRAM, it will be ignored and CIP# will be deactivated.

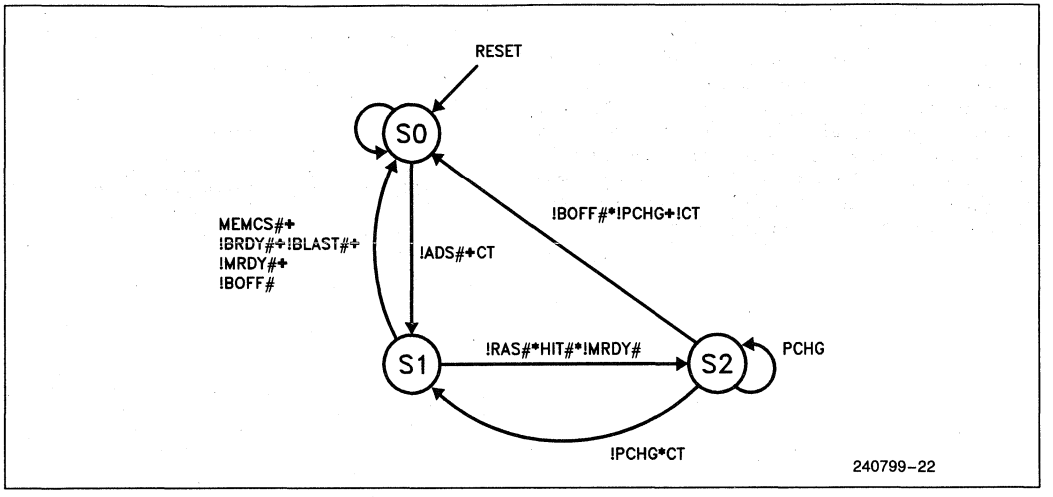


Figure 25. Cycle in Progress State Diagram

This function is defined by the S0 and S1 states in Figure 25. As shown, CIP# is activated when either ADS# or CT are sampled active. If the cycle is not to a DRAM address, the MEMCS# signal will not be active in the next clock. In this case, CIP# is deactivated to wait for the next ADS#. If the cycle is to DRAM, CIP# stays active until the end of the bus cycle. The bus cycle is terminated by one of three circumstances. All write cycles are terminated with the MRDY# signal. Read cycles are terminated by BRDY# and by BLAST#. The cycle can be aborted by BOFF#. Any of these three events causes CIP# to be deactivated (S1 to S0).

Two special cases are also handled by this state machine. When AHOLD is active in the same clock as ADS#, MEMCS# is not valid. In this case, the CIP# signal is not activated until AHOLD is deasserted. The state machine remains in S0 when AHOLD is active.

The second case is a write miss cycle. During a write miss, CIP# must be active for the cycle to complete. CIP# is active in this case after MRDY# is returned to the CPU. Cycles that start during the time CIP# is active must be tracked by the CT state machine. The M# signal indicates to the CT state machine that the cycles must be tracked.

The state in which M# and CIP# are both active is S2. This state is entered when MRDY# and RAS# are active and HIT# is inactive. By using MRDY# to qualify this transition, S2 is entered only during write cycles. Therefore, M# is only activated during write miss cycles. Note that any cycle will be recognized by the CT state machine when M# is active.

The CT state machine is shown in Figure 26. This state machine tracks cycles that start while the CIP# state machine is busy. It tracks CPU cycles that start during refresh cycles as well as to the two cases mentioned above.

This state machine tracks one cycle. Any cycle that starts while CIP# is busy is not terminated immediately. The MRDY# and MBRDY# signals are delayed until the previous cycle is finished. Therefore, anytime CT is active, there is only one cycle pending.

CT is deactivated when the pending cycle is recognized by the CIP# state machine. This event is indicated by CIP# active and M# inactive. When this event occurs, the CT state machine transitions to S0 deactivating CT.

The ALD signal is also active only during DRAM cycles. Therefore, its state machine is very similar to that of CIP#. As with CIP#, ALD is asserted when ADS# is sampled active. If the cycle is not to a DRAM address, ALD is deasserted. When a DRAM cycle is terminated, ALD is also deasserted. The S0- to-S1 transition is quite similar to that of CIP#.

The difference between the two state machines is revealed during write miss cycles. The S1-to-S2 transition is made if a write miss occurs. ALD must be held active during a write miss until RAS# is active. In this way the row address is held even if another cycle occurs. The combination of CIP# being active while PCHG is inactive indicates that RAS# will be active in this clock. ALD must be deactivated in this clock to allow the next address to be latched. ALD is re-activated if

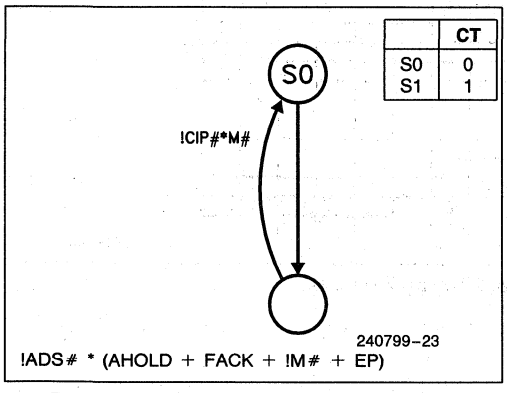


Figure 26. Cycle Tracking State Machine

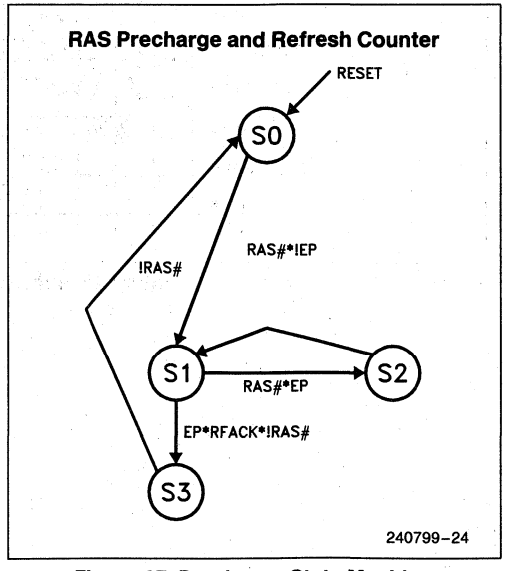


Figure 27. Precharge State Machine

another cycle has started during the write miss process. CIP# and MEMCS# are sampled during S0 for this purpose.

The PCHG state machine provides two functions. It determines the time RAS# is inactive during a miss or refresh cycle, and it determines the timing of refresh cycles. Figure 27 shows the state transitions of the PCHG state machine. Because the timing of this signal is not obvious, Figure 28 has been included. It shows a refresh cycle which occurs following a write cycle.

After RAS# is active the PCHG signal is activated. State S1 is maintained then until RAS# is deactivated. RAS# is only deactivated during a miss or refresh cycle or, of course, if RESET is asserted. During a miss cycle the transition to S0 is made deactivating PCHG. RAS# is then activated, resulting in two CPU clocks of RAS# precharge time.

States S2 and S3 define the timing of refresh cycles. The transition to this sequence is made when RAS# is sampled inactive while EP active. EP indicates that the RAS# state machine has entered the refresh sequence.

RFACK# initiates the refresh sequence. It indicates that the control logic is ready to accept a refresh request. The RFRQ signal is sampled at the end of a DRAM cycle or during idle clocks. Note that RFRQ cannot be recognized during a write miss.

RFACK# is deactivated after RAS# is deactivated at the beginning of the refresh sequence (See Figure 27 and Figure 28).

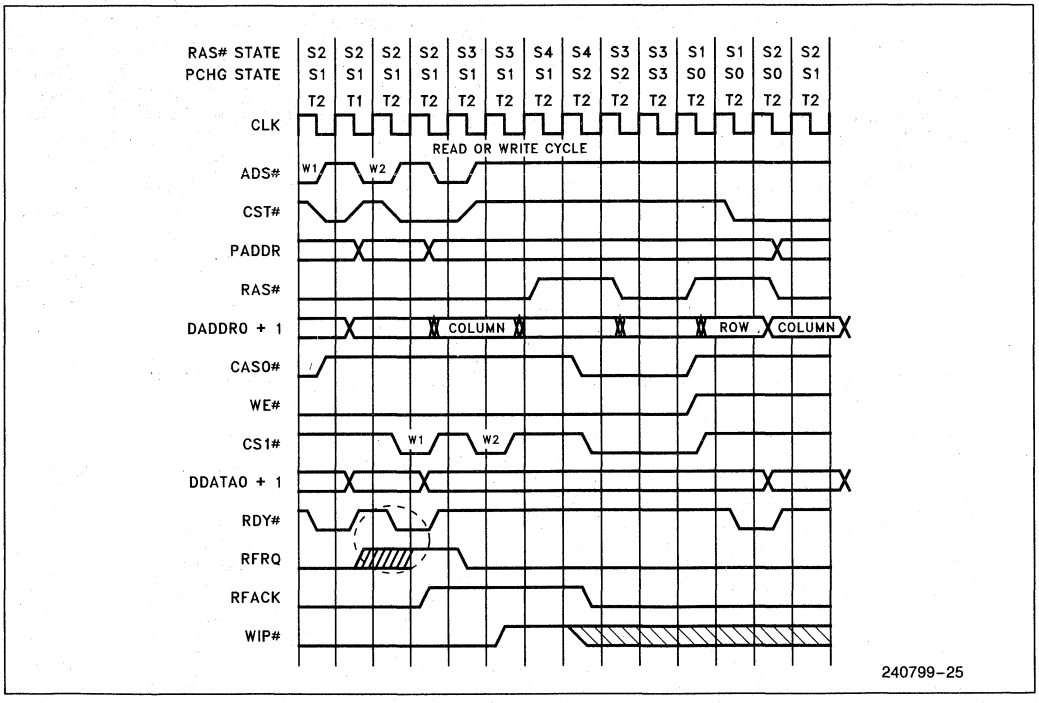


Figure 28. Refresh State-Timing Example

6.2 RAS# Logic

The RAS# logic for both memory banks occupies one PLD. Four RAS# signals are generated: RAS0# – RAS3#. These signals are generated to divide loading. Their timing is identical. The state machine for RAS is relatively simple and is shown in Figure 29.

States S0 and S1 are used to implement RAS# function for normal cycles. After RESET, the state machine waits for the first bus cycle. The first bus cycle is signaled by the CIP# signal. When CIP#, MEMCS# and PCHG are sampled active, RAS# is asserted. RAS# stays active until a miss or refresh cycle occurs.

A miss cycle is indicated when the HIT# signal is driven inactive. It is qualified by CIP# and MEMCS# being active. In this way, RAS# is only deactivated during DRAM cycles.

Once RAS# is deasserted during a miss cycle, it stays high until PCHG is sampled active. This function implements the RAS# precharge time. CIP# and MEMCS# will still be active during read miss cycles. Therefore, RAS# will be asserted in the next clock. For write miss cycles the WIP# signal must be used to restart RAS#. With a write miss, a non-DRAM cycle can occur before RAS# is asserted. WIP# is the only valid indication that a DRAM cycle has occurred in this case. WIP# is combined with MEMCS# to create the CSWIP# term which indicates a valid RAS# cycle.

When a refresh cycle occurs, the RAS# state machine transitions to S2. S2 and S3 are devoted to the refresh function. When RFACK is sampled active, the transition occurs. The refresh sequence shown in Figure 28 illustrates the function of these two states. Note that after a refresh cycle, RAS# is left inactive. The transition from S0 to S4 allows for refresh cycles that start when RAS# is inactive.

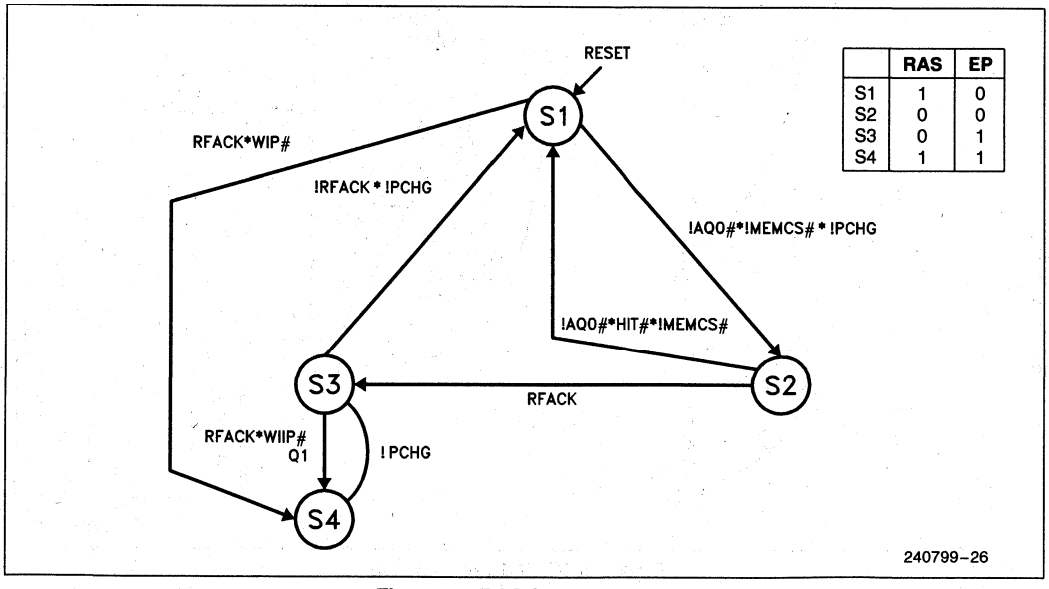
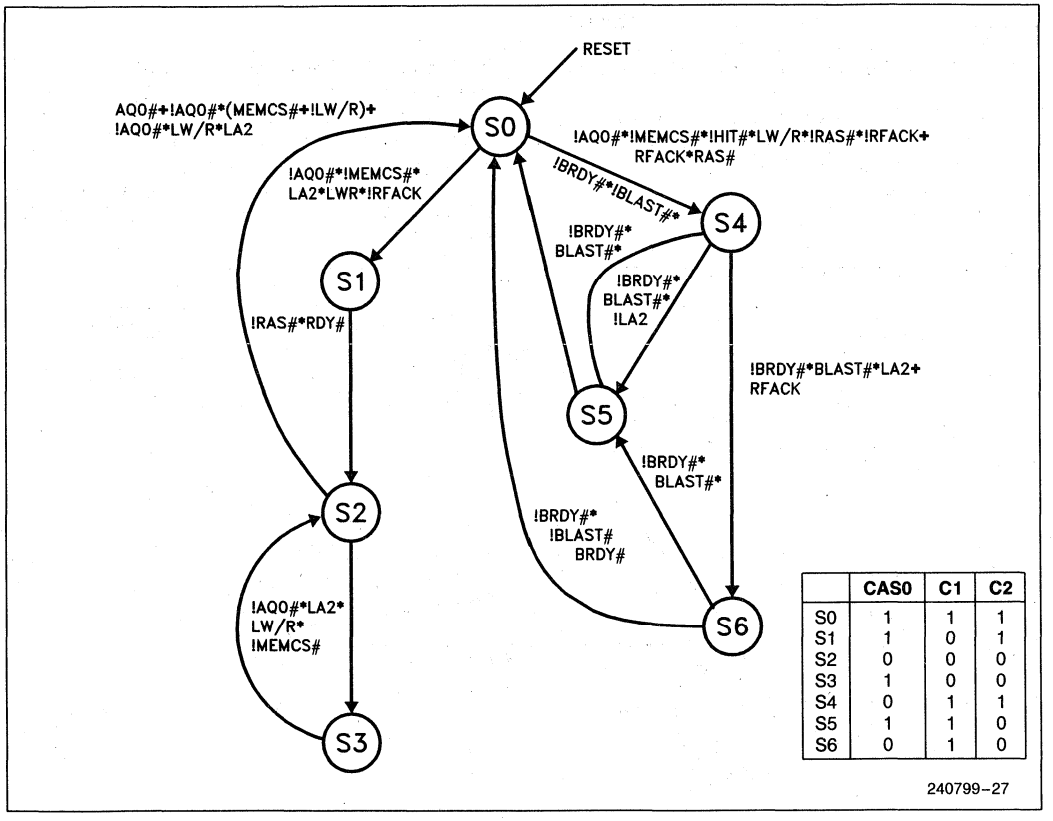


Figure 29. RAS State Machine

6.3 CAS# Logic

Two separate PLDs implement the CAS# function. These PLDs generate the CAS# signals for bank 0 and bank 1, respectively. The state machines which generate these signals are separate and independent. Each generates two CAS# signals. CAS00# and CAS01# for bank 0, and CAS10# and CAS11# for bank1. These signals drive separate DRAM modules due to drive requirements.

Figure 30 shows the state diagram for the bank 0 CAS# function. The states on the left side of the diagram implement the write function. The states on the right implement the read function. As with RAS#, the state machine waits until CIP# indicates that a cycle has started. When CIP# is active, the state of the latched version of W/R# determines which sequence is started.



	CAS0	C1	C2
S0	1	1	1
S1	1	0	1
S2	0	0	0
S3	1	0	0
S4	0	1	1
S5	1	1	0
S6	0	1	0

Figure 30. CAS State Machine

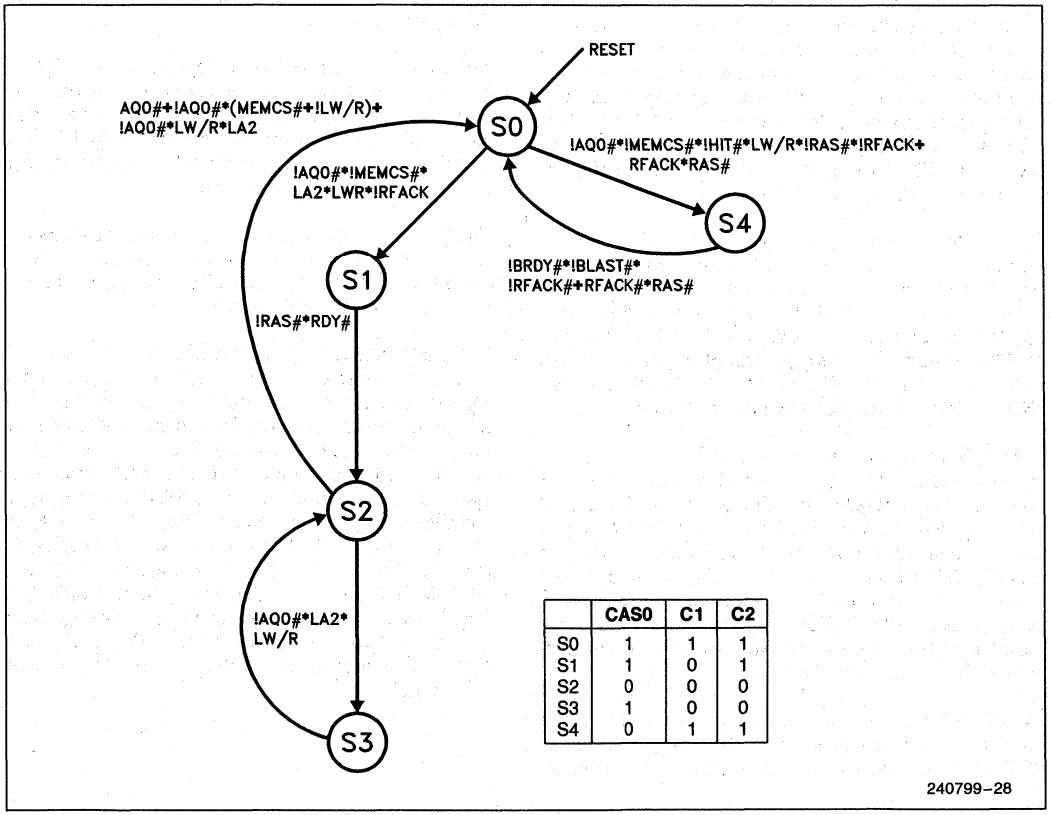
If the cycle is a read, S4 is entered. If the cycle is a write, LA2 is sampled to determine if the cycle is to bank 0. If LA2 is low, S1 is entered. Note that this function is the same for the bank 1 state machine. The only difference is the state of LA2, which starts the write sequence.

During a write cycle, CAS# is held inactive until the clock after RDY# is asserted. The state machine also waits in S1 during a write miss cycle. CAS# is asserted during S2. In this state, several events can occur. First, the CPU may not start another bus cycle. Second, it may start a bus cycle other than a DRAM cycle. Third, it may initiate a read cycle, and fourth, it may begin a write cycle to bank 1. If any of these events occur, S1 is entered. If another write cycle starts to the same bank, however, S3 is entered.

The case of sequential writes to the same bank involves S2 and S3 only. An unlimited number of write cycles can occur in the same bank. If the DRAM row is same, they will occur without wait-states. If a write miss occurs, RAS# will be deasserted, and the transition from S3 to S1 takes place.

During read cycles, the CAS# signals for bank 0 and bank 1 are activated at the same time. Therefore, the state machines enter S4 at the same clock. At this point, however, the state of LA2 determines which state machine enters S5. In S5, CAS# is deasserted to prepare that bank for the next access. If S6 is entered, the data from that bank has not yet been accessed. CAS# must be held active, in this case, until the data is sampled by the CPU. From S6, the next transition will be to S5 to continue the cycle, or S0 to terminate the cycle. If this bank was accessed first, the cycle will terminate from this state.

The read sequence is much simpler if static column mode DRAMs are used. The state sequence for static column mode is shown in Figure 31. The write sequence in this diagram is exactly the same as for the page mode CAS# control logic. The read function, however, requires only two states. From S0, the transition is made to S5 any time that a DRAM read cycle starts. Note that LA2 is not used to qualify this transition. Therefore, the CAS# signals for bank 0 and bank 1 are active at the same time.



	CAS0	C1	C2
S0	1	1	1
S1	1	0	1
S2	0	0	0
S3	1	0	0
S4	0	1	1

Figure 31. Static Column CAS State Machine

6.4 Write Control Logic

The posted write implementation requires logic support for a few key functions. These functions are required mainly to support posting with interleaved memory. Three types of signals are generated to implement these functions:

Multiplexer Select - These signals control the address multiplexers when RAS# is active. During write cycles, they must be active to select the write address path. These signals stay active during read cycles which are immediately preceded by a write. They are deactivated, when the write cycle is complete. Once they are deactivated the read cycle may proceed as the read path is selected.

Write Enable - These signals are combined with the byte enable CPU outputs (BE0# – BE3#) to create the WBE# signals. The WBE00# – WBE03# signals control which byte is written in bank 0 during a write cycle. The WBE10# – WBE13# signals perform the same function for bank 1.

Write In Progress - This signal is active when a write cycle has been started by either DRAM bank. It is active when either C01# or C11# is active. C01# and C11# are state outputs from the CAS# state machine which indicates that a write cycle is being performed. C01# is generated for bank 0 and C11# for bank 1. WIP# is only required for interleaved memory systems. The C01# (or C11#) output would be sufficient for a non-interleaved (single bank) system.

The state machines which generate these signals are shown in figure 32. The state diagram for the MEN0# signal is shown. This signal enables the address multiplexer for bank 0. MEN0# is activated whenever a write cycle occurs to an address with A2 low (0). The MEN1# function is the same except that it is activated when A2 is high (1). The AQ0#, MEMCS# and LW/R# signals are used to indicate a valid write cycle.

The MEN# signals are deactivated when the write cycle is complete. The cycle is complete when CAS# for that bank is sampled active. For bank 0, C01# is used to indicate that a write is in progress. MEN0# is held active when C01# is active. When CAS00# is sampled active, CIP# is checked to determine if another valid write to the same bank has occured. If so, MEN0# stays active until CAS00# is sampled active. This function keeps the write address path open during consecutive writes to the same bank.

The WE# state machine is very similar to that of the MEN# state machine. When a write cycle starts, WE0# is activated in the same manner as MEN0#. The write enable signals, however, must stay active one clock longer than the MEN# signals. Therefore, the WE# signal is not deactivated until C01# is sampled inactive.

WIP# is generated in part by combinatorial logic so that it can be active in the same clock as the C01# and C11# signals. WIP# must be active in this clock to ensure that a write miss is completed before a refresh cycle takes place. WIP# must also be held active one clock after C01# and C02# are sampled inactive. This timing ensures the proper sequence for subsequent read cycles. The logic equation and state machine for WIP# are shown in Figure 32.

6.5 Burst Address Logic

The burst address logic generates the B1MA0 and B0MA0 signals. These signals are connected directly to the low order address inputs of the DRAMs. Because of the direct connection, these signals must perform several different functions. They must multiplex the low order row and column addresses, multiplex the write and read addresses and generate the burst address during read cycles.

These functions are performed separately for each bank by two PLDs. Each PLD generates two identical signals to reduce the drive requirements. These signals are connected directly to two bytes of the DRAM array. The signals are generated partly by combinatorial logic and partly by the state machine.

The logic equations and state diagram for this function are shown in Figure 33. The state machine generates the burst address for read cycles. The logic equations handle the multiplexing functions.

The burst address is generated after a burst read cycle has started. Note that the i486 CPU cache need not be enabled for burst cycles to occur. Cycles such as 64-bit floating-point operand reads will burst if BRDY is returned to the processor. S0 and S3 track the state of the A3 CPU address output. When a burst read cycle starts, S1 or S2 is entered. The B0MA0 address output will then change its state when MBRDY# and DATASEL are both low. This function is the burst address for bank 0. The B1MA0 address output changes its state when MBRDY# is low and DATASEL is high. This function is the burst address for bank1. The only difference in the two PLDs is the value of DATASEL used to determine the time of which the burst address changes its state.

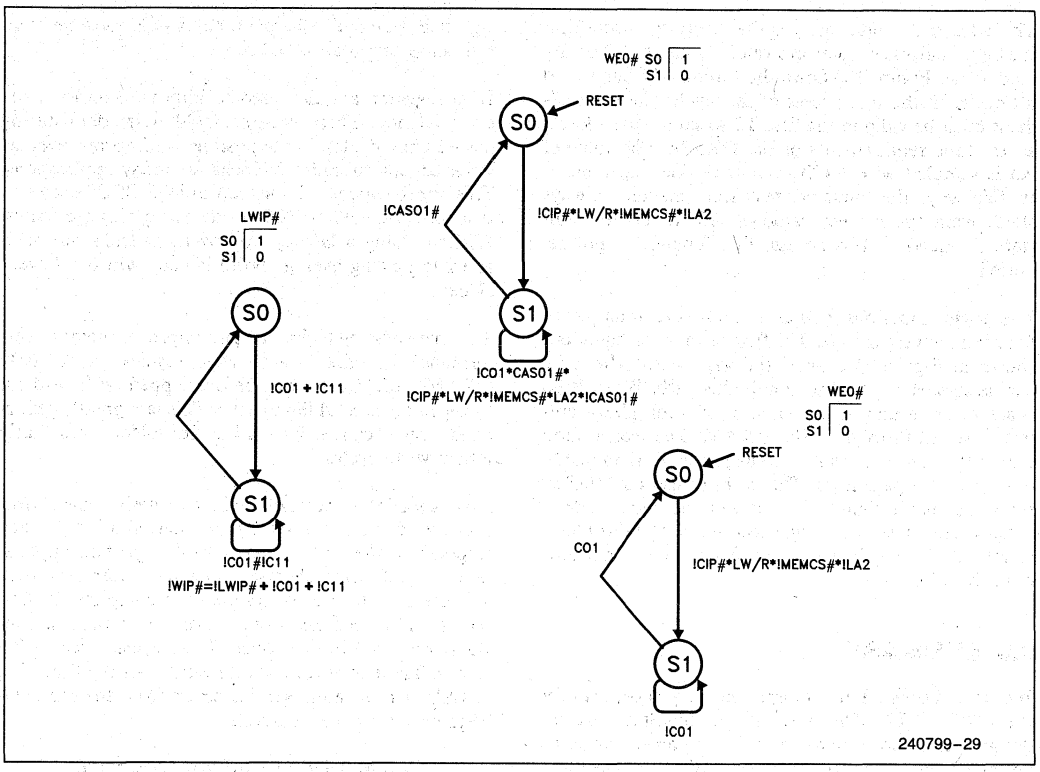
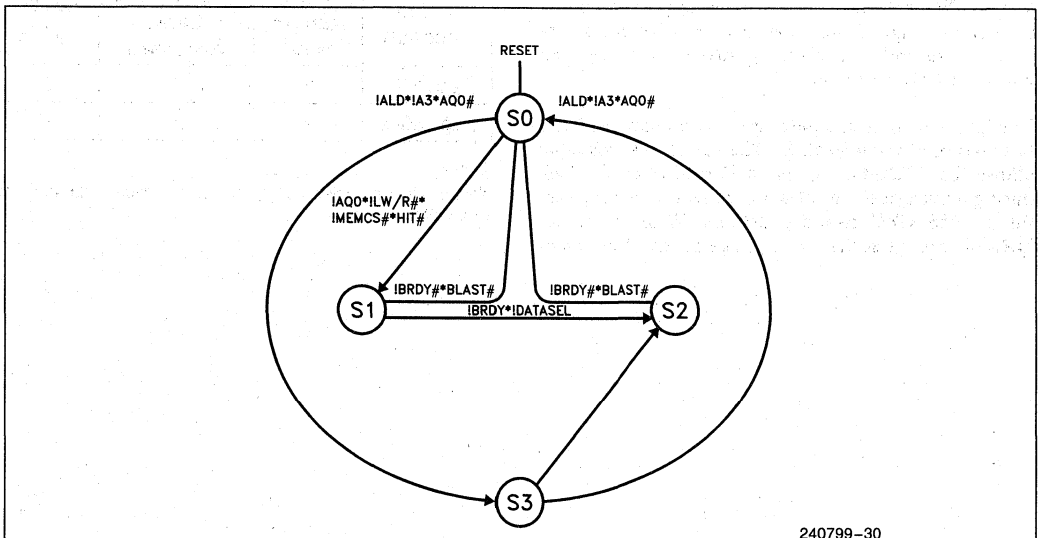


Figure 32. State Machines for MEN0#, WIP#, and WE0#



!B00MA0 = !WE0# * !LA313 + WE0# * RAS# * !LA313 * WE0# * !RAS# * !B0A

RAS Precharge and Refresh Counter

Figure 33. Burst Address Generation

The S0 and S3 states are required only to ensure that the burst address outputs are valid during the T2 of any read cycle. Figure 17 shows the timing of a burst read hit cycle. In the first access of this cycle, the burst address must be valid in the first T2 to satisfy the address access time requirements of the DRAM. The value of A3 is sampled with ALD to statisfy this requirement. In this way, the burst address state machine always starts from the correct value of A3. If another wait state is added to this access, this function is not required.

The logic equations which provide the multiplexor function are very simple. The first term of the equations shown in Figure 33 enables the write path. The write enable signals are used to enable this path. When WE0 is active, for example, the value of the multiplexor output is passed through to the DRAM. The second term allows the row address A13 to be passed to the DRAM during a read page miss. This term is also qualified by the write enable signals. In this way, the write address is not disabled early during a read miss. The third term enables the burst address output from the state machine onto the address pins.

7.0 SUMMARY

We have discussed an example memory subsystem for the i486TM CPU. The material has been presented as a design guide for systems under development or as an optimization for existing systems. We have discussed several key functions which will be summarized in this section. We will also discuss some important timing restrictions. The key functions discussed include an external or second level cache, posted write cycles, and interleaved DRAM banks.

The interleaving technique is used to support the burst bus feature of the i486 CPU. The use of this technique allows the DRAM to supply a DWORD every clock during burst cycles. Interleaving proves to be very useful in i486 CPU memory designs. Without its use DRAM timings such as tPC (Page Mode Cycle time) and tCP (CAS Precharge time) would prevent zero wait state access at 33 MHz.

Data registers are also used to improve average write cycle latency. These registers hold write data during posted write cycles. Write posting can improve average write latency to under 3 clocks for many applications. This improvement is important in i486 CPU based systems because 65% to 70% of all bus cycles are writes. Without using a latency improvement technique such as write posting average write latency will be above 5 clocks.

The write posting technique also improves memory performance in other ways. Write cycles, particularly DRAM page misses, can be overlapped with read hit cycles in the second level cache. This fact greatly reduces the delay caused by read cycles which immediatly follow write cycles.

Analysis of this memory subsystem design has shown that use of these features has resulted in a low latency response to the CPU. Over several important applications the following characteristics have been recorded. The average clock cycles required to complete the first read is 3.5 clocks. Subsequent cycles of a burst are always processed in one clock. Write cycles average 2.5 clocks. These average counts result from the following DRAM access rates. Read accesses from the cache always occur in zero wait states.

Table 3. Dram Function Latencies

DRAM Function	First Access Burst	Subsequent Burst Accesses	Write Cycles
Page Hit	3	1	2
Page Miss	7	1	5*

NOTE:
*Write miss latencies occur only during cycles subsequent to a write miss cycle.

7.1 Timing Restrictions

A few DRAM timing restrictions must be mentioned. These timings become critical at 33 MHz. These timings are critical due primarily to the latency of the first cycle of a read page hit. Since three clocks are used the following timing restrictions exist.

tRAC = Data access time from RAS# active

tCAA = Data access time from column address valid

tCAC = Data access time from CAS# active

tRP = RAS# precharge time

At 33 MHz

tRAC = 71.5 ns

tCAA = 37.5 ns

tCAC = 34 ns

tRP = 60.6 ns

At 25 MHz

tRAC = 101.5 ns

tCAA = 51 ns

tCAC = 61.5 ns

tRP = 80 ns

APPENDIX A
PLD CODES AND SCHEMATICS

A.1 PLD DEVICES

Many design examples in this manual use PLDs (Programmable Logic Devices) which can be programmed by the user to implement random logic. A PLD device can be used as a state machine or a signal decoder, for example. The advantages of PLDs include the following:

1. PLD pinout is determined by the designer, which can simplify board layout by moving signals as required.
2. PLDs are inexpensive as compared to dedicated bus controllers.

Intel EPLDs (Erasable Programmable Logic Devices) have the following additional advantages:

1. Programmability/erasability allows EPLD functions to be changed easily, simplifying prototype development.
2. Since EPLDs are implemented in CMOS technology, they can consume an order of magnitude less power than bipolar PLDs. Power-conscious applications can benefit greatly from using EPLDs.
3. Since the EPROM cell size is an order of magnitude smaller than an equivalent bipolar fuse, EPLDs can implement more functions in the same package. This higher integration can result in a lower overall component count for a design. The added flexibility can also mean that an extremely low number of "raw" (unprogrammed) devices need to be stocked versus bipolar PLDs.
4. Once an EPLD design has been tested, plastic OTP (One-Time Programmable) versions of the device can be used in a production environment.

PLDs have the following tradeoffs:

1. Most PLDs do not have buried (not connected to outputs) registers. For some state machine applications, this means using an otherwise available output pin to store the current state.
2. The drive capability of CMOS EPLDs may be insufficient for some applications. While the trend is towards use of CMOS throughout a system, in cases where high current levels are required, some additional buffering may be required with EPLDs.

A PLD consists logically of a programmable AND array whole output terms feed a fixed OR array. Any sum-of-products equations, within the limits of the number of PLD inputs, outputs, and equation terms, can be realized by specifying the correct AND array connections. Figure B-1 shows an example of two PLD equations and the corresponding logic array. Note that every horizontal line in the AND array represents a multi-input AND gate; every vertical line represents a possible input to the AND gate. An X at the intersection of a horizontal line and a vertical line represents a connection from the input to the AND gate.

The sum-of-products is then routed to a configurable macrocell. The macrocell in Figure B-2 can be configured as a combinational output or registered output. The output can be active high or active low. A separate AND term controls the output buffer.

Designing with PLDs consists of determining where Xs must be placed in the AND array and how to configure the macrocell. This task is simplified by logic compilers, such as iPLS II (Intel's Programmable Logic Software II) or ABEL. Logic compilers accept input in the form of sum-of-product equations and translate the input into a JEDEC programming file that can be used by programming hardware/software.

Intel PLDs are described in the *Programmable Logic Handbook.* Three Intel PLDs have been used in this manual to implement state machine and decode functions. These PLDs include:

- 85C220—fast 20-pin superset of 16 x 8 type bipolar and CMOS PLDs.
- 85C224—fast 24-pin superset of 20 x 8 type bipolar and CMOS PLDs.
- 85C508—fast address decode PLD with integral transparent latches.

The 85C220 and 85C224 PLDs are both available at clock speeds to support fast state-machines in i486 systems. The 85C508 provides a fast Enable-to-Output time with a minimal system setup time.

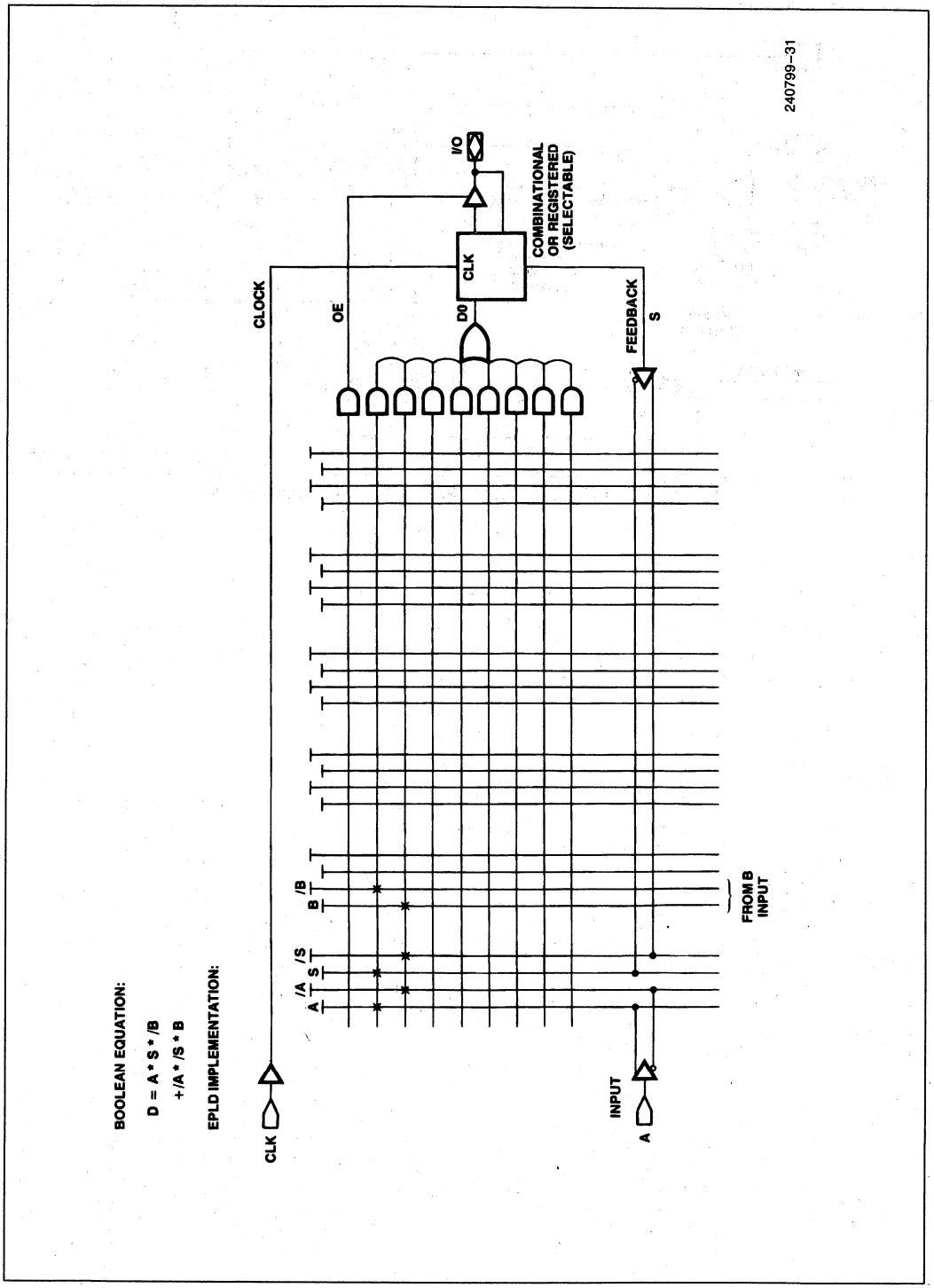


Figure A-1. PLD Equation and Device Implementation

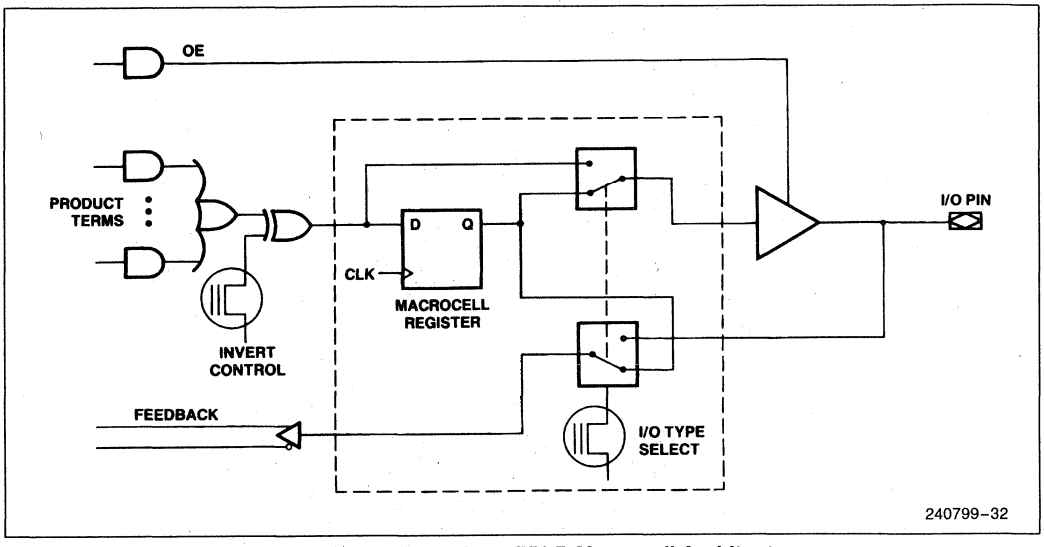


Figure A-2. 85C220/85C224 EPLD Macrocell Architecture

```
module   SC_MODE_DRAM_CTRL_4   flag '-r4'

title    'STATIC COLUMN MODE DRAM CONTROLLER - PLD 4, INTEL CORPORATION'
" This pld generates MRDY and MBRDY
" Implemented with Intel 85C224 EPLD.

  SCk  device          'E224';

  x          =    .X.;              " ABEL 'don't care' symbol
  c          =    .C.;              " ABEL 'clocking input' symbol

" Inputs

  CLK pin 1;  "P4 input CLK"
  M~          pin  2; "Miss Indicator
  CIP~        pin  3; "Cycle OK
  MEMCS~      pin  4; "Latched A2.
  HIT~        pin  5; "DRAM Page Hit Signal
  RFACK       pin  6; "Refresh acknowledge"
  ADS~        pin  7; "CPU ADS~
  W_R         pin  8; "CPU W/R
  RESET       pin  9; "System Reset
  dum1        pin 10; "Write in progress
  BOFF~       pin 11; "CPU Backoff input
  WIP~        pin 14; "CPU Burst Last output
  CAS~        pin 15; "Row Address strobe
  BLAST~      pin 22;
  RAS~        pin 23; "Any CAS# signal

" Output

  dum0        pin 16;
  MT          pin 17; " BRDY state miss tracking
  MRDY~       pin 18; " Memory RDY (modified with other RDYs)
  DALE~       pin 19; " Decode Latch enable
  LWR         pin 20; " Internally latched W/R# for rdy
  BRDY~       pin 21; " Processor BRDY~

state_diagram [MRDY~]

  state [1]:   if (!RFACK & !ADS~ & W_R & !RAS~ & M~) # (!CIP~ & LWR &
               !MEMCS~ & !RFACK & M~) then [0] else [1];

  state [0]:   goto [1];

state_diagram [BRDY ~, MT]

  state [1, 1]: if !CIP~ & !HIT~ & !MEMCS~ & !LWR & !RFACK & WIP~ & !RAS~
                then [0, 1] else if !CIP~ & !MEMCS~ & HIT~ & !LWR #
                !CIP~ & !MEMCS~ & RAS~ & !LWR then [1, 0];
```

240799-33

5

```
  state [1, 0]: if RESET then [1, 1] else
                  If WIP~ & !RFACK & !CAS~ then [0, 1];

  state [0, 1]: if RESET # !BOFF~ # !BLAST ~ then [1, 1] else [0, 1];

state_diagram [DALE~]

  state [0]:    if RESET then [0] else
                if !ADS~ then [1] else [0];

  state [1]:    if RESET # !BOFF~ then [0] else
                if !CIP~ then [0] else [1];

state_diagram [LWR]

  state [0]:    if RESET then [0] else
                if !ADS~ & W_R then [1] else [0];

  state [1]:    if RESET # !BOFF~ then [0] else
                if !ADS~ & !W_R then [0] else [1];

test_vectors

([CLK,M~,CIP~,MEMCS~,HIT~,RFACK,ADS~,W_R,RESET,WIP~,BOFF~,BLAST~]
 - >     [RAS~,MRDY~,DALE~,LWR,BRDY~])

" C M A M H R A W R W B B R     M D L B
" L ~ Q E I F D_E I O L A     R A W R
" K   0 M T A S R S P F A S     D L R D
"       ~ C ~ F ~     E     F S ~     Y E     Y
"         S   K       T   ~ T           ~ ~     ~
"        ~                  ~
"
"
"

   [c, x, x, x, x, x, 1, x, 1, x, x, x, x] - > [x, x, x, x];
   [c, x, 1, 1, x, x, 1, x, 1, x, x, x, x] - > [1, 0, 0, 1];
   [c, 1, 1, 1, x, 0, 1, x, 0, 0, 1, 1, 1] - > [1, 0, 0, 1];
   [c, 1, 1, 1, x, 0, 1, x, 0, 0, 1, 1, 1] - > [1, 0, 0, 1];
   [c, 1, 1, 1, x, 0, 0, 1, 0, 0, 1, 1, 1] - > [1, 1, 1, 1];
   [c, 1, 0, 0, 0, 0, 1, x, 0, 0, 1, 1, 1] - > [0, 0, x, 1];
   [c, 1, 0, 0, 0, 0, 1, x, 0, 1, 1, 1, 0] - > [1, 0, 1, 1];
   [c, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0] - > [0, 1, 1, 1];
   [c, 1, 0, 0, 0, 0, 1, x, 0, 1, 1, 1, 0] - > [1, 0, x, 1];
   [c, 1, 1, x, 0, 0, 0, 1, 0, 1, 1, 1, 0] - > [0, 1, 1, 1];
   [c, 1, 0, 0, 0, 0, 1, x, 0, 1, 1, 1, 0] - > [1, 0, x, 1];
   [c, 1, 1, x, 0, 0, 1, x, 0, 1, 1, 1, 0] - > [1, 0, x, 1];
   [c, 1, 1, 1, x, 0, 1, x, 0, 1, 1, 1, 0] - > [1, 0, x, 1];

end SC_MODE_DRAM_CTRL_4;
```

240799-34

```
module   SC_MODE_DRAM_CTRL_3   flag '-r4'

title     'STATIC COLUMN MODE DRAM CONTROLLER - PLD 3, INTEL CORPORATION'
" This PLD generates RAS
" Implemented with the Intel 85C220 EPLD.

  SC3     device     'E0320';

  x          =    .X.;              " ABEL 'don't care' symbol
  c          =    .C.;              " ABEL 'clocking input' symbol

" Inputs

  CLK pin   1;  "P4 input CLK"
  M~          pin  2; "Refresh Acknowledge
  CIP~        pin  3; "Cycle OK
  MEMCS~      pin  4; "Latched A2.
  HIT~        pin  5; "DRAM Page Hit Signal
  RFACK       pin  6; "Backoff input to P4"
  PCHG        pin  7; "RAS precharge count
  WIP~        pin  8; "Write in Progress
  RESET       pin  9; "System Reset
  Q1          pin 12; "RAS refresh count
" Output

  RAS2~       pin 13; "
  RAS1~       pin 14; " RAS byte 0,2
  EP          pin 15; " state variable
  EP1 pin 16; " state variable
  RAS0~       pin 17; " RAS byte 1,3
  RAS3~       pin 18; "
  CSWIP~      pin 19; "

state_diagram [RAS0~,RAS1~,EP]

  state [1, 1, 0]:    if RESET then [1, 1, 0] else
                      if !CIP~ & !CSWIP~ & !PCHG then [0, 0, 0] else
                      if RFACK & WIP~ then [1, 1, 1] else
                      [1, 1, 0];

  state [0, 0, 0]:    if RESET then [1, 1, 0] else
                      if RFACK then [0, 0, 1] else
                      if !CIP~ & HIT~ & !MEMCS~ then [1, 1, 0]
                      else [0, 0, 0];

  state [0, 0, 1]:    if RESET then [1, 1, 0] else
                      if !RFACK & !PCHG then [1, 1, 0] else
                      if RFACK & !WIP~ # !RFACK & PCHG then
                      [0, 0, 1] else if RFACK & WIP~ & !Q1 then [1, 1, 1];

  state [1, 1, 1]:    if RESET then [1, 1, 0] else
                      if !PCHG then [0, 0, 1] else [1, 1, 1];
```

240799-35

5

```
  state [0, 1, 0]:    goto [1, 1, 0];
  state [0, 1, 1]:    goto [1, 1, 0];
  state [1, 0, 0]:    goto [1, 1, 0];
  state [1, 0, 1]:    goto [1, 1, 0];

state_diagram [RAS2~,RAS3~,EP1]

  state [1, 1, 0]:    if RESET then [1, 1, 0] else
                      if !CIP~ & !CSWIP~ & !PCHG then [0, 0, 0] else
                      if RFACK & WIP~ then [1, 1, 1] else
                      [1, 1, 0];

  state [0, 0, 0]:    if RESET then [1, 1, 0] else
                      if RFACK then [0, 0, 1] else
                      if !CIP~ & HIT~ & !MEMCS~ then [1, 1, 0]
                      else [0, 0, 0];

  state [0, 0, 1]:    if RESET then [1, 1, 0] else
                      if !RFACK & !PCHG then [1, 1, 0] else
                      if RFACK & !WIP~ # !RFACK & PCHG then
                      [0, 0, 1] else if RFACK & WIP~ & !Q1 then [1, 1, 1];

  state [1, 1, 1]:    if RESET then [1, 1, 0] else
                      if !PCHG then [0, 0, 1] else [1, 1, 1];

  state [0, 1, 0]:    goto [1, 1, 0];
  state [0, 1, 1]:    goto [1, 1, 0];
  state [1, 0, 0]:    goto [1, 1, 0];
  state [1, 0, 1]:    goto [1, 1, 0];

equations

  !CSWIP~ = (!MEMCS~ # !WIP~)& !RESET;

test_vectors

  ([CLK,M~,CIP~,MEMCS~,HIT~,RFACK,PCHG,WIP~,Q1,RESET] -> 
  [RAS0~,RAS1~,EP,RAS2~,RAS3~,EP1])

"  C M A M H R P W Q R  R R E R R E
"  L ~ Q E I F C I 1 E  A A P A A P
"  K  0 M T A H P  S  S S  S S 1
"     ~ C ~ C G ~  E  0 1  2 3
"       S  K      T
"        ~
"
"
"

  [c, x, x, x, x, x, 1, x, x, 1] -> [x, x, x, x, x, x];
  [c, x, x, x, x, x, 1, x, x, 1] -> [1, 1, 0, 1, 1, 0];
  [c, x, x, x, x, x, 1, x, x, 1] -> [1, 1, 0, 1, 1, 0];
  [c, x, x, x, x, x, 1, x, x, 1] -> [1, 1, 0, 1, 1, 0];
```

240799-36

```
    [c, x, x, x, x, x, 1, x, x, 1] -> [1, 1, 0, 1, 1, 0];
    [c, x, x, x, x, x, 1, x, x, 1] -> [1, 1, 0, 1, 1, 0];
    [c, x, x, x, x, x, 1, x, x, 1] -> [1, 1, 0, 1, 1, 0];
    [c, 1, 1, x, x, 0, 0, 0, 0, 0] -> [1, 1, 0, 1, 1, 0];
    [c, 1, 1, x, x, 0, 0, 0, 0, 0] -> [1, 1, 0, 1, 1, 0];
    [c, 1, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 0, 0, 0];
    [c, 1, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 0, 0, 0];
    [c, 1, 1, x, x, 0, 0, 0, 0, 0] -> [0, 0, 0, 0, 0, 0];
    [c, 1, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 0, 0, 0];
    [c, 1, 1, x, x, 0, 0, 0, 0, 0] -> [0, 0, 0, 0, 0, 0];
    [c, 1, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 0, 0, 0];
    [c, 1, 1, x, x, 0, 0, 0, 0, 0] -> [0, 0, 0, 0, 0, 0];
    [c, 1, 1, x, x, 0, 0, 0, 0, 0] -> [0, 0, 0, 0, 0, 0];
    [c, 1, 1, x, x, 0, 0, 0, 0, 0] -> [0, 0, 0, 0, 0, 0];

end SC_MODE_DRAM_CTRL_3;
```

240799-37

5

```
module   SC_MODE_DRAM_CTRL_1   flag '-r4'

title    'STATIC COLUMN MODE DRAM CONTROLLER - PLD 1, INTEL CORPORATION
" Cycle Tracking Logic
" Implemented with Intel 85C224 EPLD.
  SCy  device     'E224';

  x          =    .X.;                    " ABEL 'don't care' symbol
  c          =    .C.;                    " ABEL 'clocking input' symbol

" Inputs

  CLK pin 1; "P4 input CLK"
  BLAST~     pin  2; "P4 BLAST output
  MEMCS~     pin  3; "Memory Chip Select
  AHOLD      pin  4; "Address HOLD input to P4"
  HIT~       pin  5; "DRAM Page Hit Signal
  BOFF~      pin  6; Backoff input to P4"
  ADS~       pin  7; "Address Status output of P4"
  RFRQ       pin  8; "Refresh Request Signal
  RESET      pin  9; "System Reset
  BRDY~      pin 10; "Processor burst ready pin.
  MRDY~      pin 11; "Memory ready
  RAS~       pin 14; "Row Address Strobe
  EP         pin 23; "Refresh indicator - count on RAS~ low

" Output

  RFACK~     pin 15; "Refresh acknowledge
  CIP~       pin 16; " ADS~ active indicator
  M~         pin 17; " AQ0~ Miss state indicator
  CT         pin 18; " AHOLD with ADS~ indicator
  PCHG       pin 19; " Precharge state indicator
  Q1         pin 20; " Precharge state indicator
  ALD pin 21; " Address Latch Disable
  adlst~     pin 22; " ADL state variable

state_diagram [CIP~, M~]

  state [1, 1]: if RESET then [1, 1] else
                  if AHOLD # !RFACK~ # EP then [1, 1] else
                  if !ADS~ # CT then [0, 1] else [1, 1];

  state [0, 1]:     if RESET # !BOFF~ # MEMCS~
                      then [1, 1] else
                      if HIT~ & !RAS~ & !MRDY~ then [0, 0] else
                      If (!MRDY~ # (!BRDY~ & !BLAST~)) then [1, 1]
                      else [0, 1];

  state [0, 0]:     if RESET # !BOFF~ then [1, 1] else
                      if !PCHG & (CT # !ADS~) then [0, 1] else
                      if !PCHG & !CT then [1, 1] else
                      [0, 0];
```

240799-38

```
  state [1, 0]: goto [1, 1];

state_diagram [PCHG, Q1]

  state [0, 0]:     if RESET then [0, 0] else
                      if !RAS~ then [1, 0] else
                      if RAS~ & !RFACK~ then [0, 1] else [0, 0];

  state [1, 0]:     if RESET then [0, 0] else
                      if RAS~ & !EP then [0, 0] else
                      if RFACK~ & EP & !RAS~ then     [1, 1] else
                      if RAS~ & EP then [0, 1] else [1, 0];

  state [0, 1]: goto [1, 0];

  state [1, 1]: goto [0, 0];

state_diagram [CT]

  state [0]:        if RESET then [0] else
                    if !ADS~ & (AHOLD # !RFACK~ # !M~ # EP) then [1] else [0];

  state [1]:        if RESET # !BOFF~ then [0] else
                    if !CIP~ & M~ then [0] else [1];

state_diagram [RFACK~]

  state[1]:         if RESET then [1] else
                    if !CIP~ & RFRQ & !MRDY~ & !HIT~ then [0] else
                    if !CIP~ & RFRQ & (!BRDY~ & !BLAST~) #
                    RFRQ & CIP~ & ADS~ then [0] else [1];

  state[0]:         if RESET # !BOFF~ then [1] else
                    if RAS~ then [1] else [0];

state_diagram [ALD, adlst~]

  state [0, 1]:     if RESET then [0, 1] else
                      if !ADS~ # !CIP~ & !MEMCS~ then [1, 0] else [0, 1];

  state [1, 0]:     if RESET then [0,1] else
                      if !CIP~ & MEMCS~ then [0, 1] else
                      if HIT~ & !MRDY~ then [1, 1] else
                      if !HIT~ & !MRDY~ then [0, 1] else
                      if !BRDY~ & !BLAST~ then [0, 1] else [1, 0];

  state [1, 1]:     if RESET then [0, 1] else
                      if !CIP~ & (!PCHG # MEMCS~) then [0, 1] else [1, 1];

  state [0, 0]: goto [0, 1];

test_vectors
```

240799-39

5

```
   ([CLK,BLAST~,MEMCS~,AHOLD,HIT~,BOFF~,ADS~,RFRQ,RESET] - >
   [BRDY~,MRDY~,RAS~,EP,RFACK~,CIP~,M~,CT,PCHG,Q1,ALD,adlst~])

"  C B M A H B A R R B R R E   R A M C P Q A a
"  L L E H I O D F E R D A P   F Q ~ T C 1 L d
"  K A M O T F S R S D Y S ~   A 0     H   D I
"    S C L ~ F ~ Q E Y ~ ~     C ~   G     s
"    T S D   ~     T ~         K         t
"    ~ ~                                 ~
"
"
"

   [c, x, x, x, x, x, 1, x, 1, x, x, x, x] - > [1, 1, 1, 0, 0, 0, 0, 1];
   [c, x, x, x, x, x, 1, x, 1, x, x, x, x] - > [1, 1, 1, 0, 0, 0, 0, 1];
   [c, 1, x, 0, x, 1, 1, 0, 1, 0, 1, 1, 0] - > [1, 1, 1, 0, 0, 0, 0, 1];
   [c, 1, x, 0, x, 1, 0, 0, 0, 0, 1, 1, 0] - > [1, 0, 1, 0, 0, 0, 1, 0];
   [c, 1, x, 0, x, 1, 1, 0, 1, 0, 1, 1, 0] - > [1, 0, 1, 0, 0, 0, 1, 0];
   [c, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0] - > [1, 0, 1, 0, 0, 0, 1, 0];
   [c, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0] - > [1, 0, 1, 0, 1, 0, 1, 0];
   [c, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0] - > [1, 0, 1, 0, 1, 0, 1, 0];
   [c, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0] - > [1, 0, 1, 0, 1, 0, 1, 0];
   [c, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0] - > [1, 0, 1, 0, 1, 0, 1, 0];
   [c, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0] - > [1, 0, 1, 0, 1, 0, 1, 0];
   [c, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0] - > [1, 1, 1, 0, 1, 0, 0, 1];
   [c, x, x, 0, x, 1, 1, 0, 1, 0, 1, 0, 0] - > [1, 1, 1, 0, 1, 0, 0, 1];

end SC_MODE_DRAM_CTRL_1;
```

240799-40

```
module  SC_MODE_DRAM_CTRL_7  flag '-r4'

title  'STATIC COLUMN MODE DRAM CONTROLLER - PLD 7, INTEL CORPORATION'
" This PLD generates DATASL and WE
" Implemented with the Intel 85C220 EPLD.

  SC7  device      'E0320';

  x          =    .X.;              " ABEL 'don't care' symbol
  c          =    .C.;              " ABEL 'clocking input' symbol

" Inputs

  CLK pin 1; "P4 input CLK"
  BRDY~      pin  2; "Burst Ready
  CIP~       pin  3; "Cycle OK
  MEMCS~     pin  4; "memory select
  LA2 pin 5; "Latched A2.
  CAS00~     pin  6; "CAS output Bank1
  CAS10~     pin  7; "CAS output Bank1
  LW_R       pin  8; "CPU W/R latched~
  RESET      pin  9; "System Reset
  BLAST~     pin 12; "CPU BLAST~ output
  BOFF~      pin 13; "CPU Backoff input
  HIT~       pin 19;
" Output

  DATASEL    pin 14; " Bank select for reads
  RS~        pin 15; " state variable
  RALE~      pin 16; " state variable
  WE~        pin 17; " Write Enable posted writes
  BSEL       pin 18; " Selects read or write data path

state_diagram [DATASEL, RS~]

  state [1, 1]: if RESET then [1, 1] else
                  if !CIP~ & !LA2 & !LW_R & !MEMCS~ then [0, 0] else
                  if !CIP~ & LA2 & !LW_R & !MEMCS~ then [1, 0] else [1, 1];

  state [1, 0]: if RESET # !BOFF~ # (!BRDY~ & !BLAST~) then [1, 1] else
                  if !BRDY~ & BLAST~ then [0, 0] else [1, 0];

  state [0, 0]: if RESET # !BOFF~ # (!BRDY~ & !BLAST~) then [1, 1] else
                  if !BRDY~ & BLAST~ then [1, 0] else [0, 0];

  state [0, 1]: goto [1, 1];

state_diagram [WE~]

  state [1]:    if RESET then [1] else
                if LW_R & !CIP~ & !MEMCS~ then [0] else [1];
```

240799-41

```
    state [0]:    if RESET # !BOFF~ then [1] else
                  if LW_R & !CIP~ & !MEMCS~ then [0] else
                  if CAS00~ + CAS10~ then [1];

state_diagram [RALE~]

    state [0]:    if RESET then [0] else
                  if !CIP~ & HIT~ & !MEMCS~ then [1] else [0];

    state [1]:    if RESET # !BOFF~ then [0] else
                  if !HIT~ then [0] else [1];

end SC_MODE_DRAM_CTRL_7;
```

240799-42

```
module  SC_MODE_DRAM_CTRL_11 flag '-r4'

title  'STATIC COLUMN MODE DRAM CONTROLLER - PLD 11, INTEL CORPORATION'
" This PLD generates the mux enables write enables and WIP#
" Implemented with the Intel 85C220 EPLD.

  SCw  device     'E0320';

  x          =    .X.;              " ABEL 'don't care' symbol
  c          =    .C.;              " ABEL 'clocking input' symbol

" Inputs

  CLK pin 1; "P4 input CLK"
  LA2 pin 2; "Latched A2.
  CIP~       pin  3; "Cycle OK
  MEMCS~     pin  4; "Memory Chip select.
  RESET      pin  5; "DRAM Page Hit Signal
  LW_R       pin  6; "latched CPU W/R#
  C01 pin 7; "Write indication Bank0
  CAS01~     pin  8; "
  C11 pin 9; "Write indication Bank1
  CAS11~     pin 19; "

" Output

  WIP~       pin 12; "New Wip signal comb
  MEN0~      pin 13; "Mux enables
  WE0~       pin 14; "
  LWIP~      pin 15; "Latched WIP~
  dum        pin 16; "
  WE1~       pin 17; "
  MEN1~      pin 18; " Mux enable Bank1

state_diagram [WE0~]

  state [1]:   if RESET then [1] else
               if !CIP~ & LW_R & !MEMCS~ & !LA2 then [0];

  state [0]:   if RESET then [1] else
               if !C01 then [0] else
               if C01 then [1];

  state_diagram [WE1~]

  state [1]:   if RESET then [1] else
               if !CIP~ & LW_R & !MEMCS~ & LA2 then [0];

  state [0]:   if RESET then [1] else
               if !C11 then [0] else
               if C11 then [1];

state_diagram [LWIP~]
```

240799-43

```
    state [1]:    if !C01 # !C11 then [0] else [1];

    state [0]:    if RESET then [1] else
                  if !C01 # !C11 then [0] else [1];

state_diagram [MEN0~]

    state [1]:    if RESET then [1] else
                  if !CIP~ & LW_R & !MEMCS~ & !LA2 then [0];

    state [0]:    if RESET then [1] else
                  if !C01 & CAS01~ then [0] else
                  if !CIP~ & LW_R & !MEMCS~ & !LA2 & !CAS01~ then [0] else
                  if !CAS01~ then [1];

state_diagram [MEN1~]

    state [1]:    if RESET then [1] else
                  if !CIP~ & LW_R & !MEMCS~ & LA2 then [0];

    state [0]:    if RESET then [1] else
                  if !C11 & CAS11~ then [0] else
                  if !CIP~ & LW_R & !MEMCS~ & !LA2 & !CAS11~ then [0] else
                  if !CAS11~ then [1];

equations

                  !WIP~ = !LWIP~ # !C01 # !C11;

"test_vectors

"([CLK,M IO~,CIP~,MEMCS~,HIT~,RFACK,ADS~,W_R,RESET,CAS0~,BOFF~,BLAST~]
"-> [RAS~,MRDY~,DALE~,LWR,BRDY~])
"
" C M A M H R A W R C B B R  M D L B
" L_Q E I F D_E A O L A  R A W R
" K I 0 M T A S R S S F A S  D L R D
"  0 ~ C ~ F ~  E 0 F S ~  Y E  Y
"    S  K  T ~ ~ T  ~ ~  ~
"    ~  ~
"
"
"
"
" [c, x, x, x, x, x, 1, x, 1, x, x, x, x] -> [x, x, x, x];
" [c, x, 1, 1, x, x, 1, x, 1, x, x, x, x] -> [1, 0, 0, 1];
" [c, 1, 1, 1, x, 1, 1, x, 0, 1, 0, 1, 1] -> [1, 0, 0, 1];
" [c, 1, 1, 1, x, 1, 1, x, 0, 1, 0, 1, 1] -> [1, 0, 0, 1];
" [c, 1, 1, 1, x, 1, 0, 1, 0, 1, 0, 1, 1] -> [1, 1, 1, 1];
" [c, 1, 0, 0, 0, 1, 1, x, 0, 1, 0, 1, 1] -> [0, 0, x, 1];
" [c, 1, 0, 0, 0, 1, 1, x, 0, 1, 0, 1, 0] -> [1, 0, x, 1];
" [c, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0] -> [0, 1, 1, 1];
" [c, 1, 0, 0, 0, 1, 1, x, 0, 0, 0, 1, 0] -> [1, 0, x, 1];
```

240799-44

```
" [c, 1, 1, x, 0, 1, 0, 1, 0, 1, 0, 1, 0] -> [0, 1, 1, 1];
" [c, 1, 0, 0, 0, 1, 1, x, 0, 1, 0, 1, 0] -> [1, 0, x, 1];
" [c, 1, 1, x, 0, 1, 1, x, 0, 1, 0, 1, 0] -> [1, 0, x, 1];
" [c, 1, 1, 1, x, 1, 1, x, 0, 1, 0, 1, 0] -> [1, 0, x, 1];
"
end SC_MODE_DRAM_CTRL_11;
```

240799-45

5

```
module   SC_MODE_DRAM_CTRL_11 flag '-r4'

title   'STATIC COLUMN MODE DRAM CONTROLLER - PLD 11, INTEL CORPORATION'
" This PLD generates the mux enables write enables and WIP#
" Implemented with the Intel 85C220 EPLD.

  SCw   device   'E0320';

  x             =     .X.;              " ABEL 'don't care' symbol
  c             =     .C.;              " ABEL 'clocking input' symbol

" Inputs

  CLK pin 1; "P4 input CLK"
  LA2 pin 2; "Latched A2.
  CIP~          pin  3; "Cycle OK
  MEMCS~        pin  4; "Memory Chip select.
  RESET         pin  5; "DRAM Page Hit Signal
  LW_R          pin  6; "latched CPU W/R#
  C01 pin 7; "Write indication Bank0
  CAS01~        pin  8; "
  C11 pin  9; "Write indication Bank1
  CAS11~        pin 19; "

" Output

  WIP~          pin 12; "New Wip signal comb
  MEN0~         pin 13; " Mux enables
  WE0~          pin 14; "
  LWIP~         pin 15; "Latched WIP~
  dum           pin 16; "
  WE1~          pin 17; "
  MEN1~         pin 18; " Mux enable Bank1

state_diagram [WE0~]

  state [1]:    if RESET then [1] else
                if !CIP~ & LW_R & !MEMCS~ & !LA2 then [0];

  state [0]:    if RESET then [1] else
                if !C01 then [0] else
                if C01 then [1];

state_diagram [WE1~]

  state [1]:    if RESET then [1] else
                if !CIP~ & LW_R & !MEMCS~ & LA2 then [0];

  state [0]:    if RESET then [1] else
                if !C11 then [0] else
                if C11 then [1];

state_diagram [LWIP~]
```

240799-46

```
   state [1]:    if !C01 # !C11 then [0] else [1];

   state [0]:    if RESET then [1] else
                 if !C01 # !C11 then [0] else [1];

state_diagram [MEN0~]

   state [1]:    if RESET then [1] else
                 if !CIP~ & LW_R & !MEMCS~ & !LA2 then [0];

   state [0]:    if RESET then [1] else
                 if !C01 & CAS01 ~ then [0] else
                 if !CIP~ & LW_R & !MEMCS~ & !LA2 & !CAS01~ then [0] else
                 if !CAS01~ then [1];

state_diagram [MEN1~]

   state [1]:    if RESET then [1] else
                 if !CIP~ & LW_R & !MEMCS~ & LA2 then [0];

   state [0]:    if RESET then [1] else
                 if !C11 & CAS11~ then [0] else
                 if !CIP~ & LW_R & !MEMCS~ & !LA2 & !CAS11~ then [0] else
                 if !CAS11~then [1];

equations

    !WIP~ = !LWIP~ # !C01 # !C11;

"test_vectors

"([CLK,M_IO~,CIP~,MEMCS~,HIT~,RFACK,ADS~,W_R,RESET,CAS0~,BOFF~]
" -> [BLAST~,RAS~,MRDY~,DALE~,LWR,BRDY~])
"
C M A M H R A W R C B B R  M D L B
" L _ Q E I F D _ E A O L A  R A W R
" K I 0 M T A S R S S F A S  D L R D
"  O ~ C ~ F ~  E 0 F S ~  Y E  Y
"    S  K   T ~ ~T   ~ ~  ~
"     ~        ~
"
"
"
"
" [c, x, x, x, x, x, 1, x, 1, x, x, x, x] -> [x, x, x, x];
" [c, x, 1, 1, x, x, 1, x, 1, x, x, x, x] -> [1, 0, 0, 1];
" [c, 1, 1, 1, x, 1, 1, x, 0, 1, 0, 1, 1] -> [1, 0, 0, 1];
" [c, 1, 1, 1, x, 1, 1, x, 0, 1, 0, 1, 1] -> [1, 0, 0, 1];
" [c, 1, 1, 1, x, 1, 0, 1, 0, 1, 0, 1, 1] -> [1, 1, 1, 1];
" [c, 1, 0, 0, 0, 1, 1, x, 0, 1, 0, 1, 1] -> [0, 0, x, 1];
" [c, 1, 0, 0, 0, 1, 1, x, 0, 1, 0, 1, 0] -> [1, 0, x, 1];
" [c, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0] -> [0, 1, 1, 1];
" [c, 1, 0, 0, 0, 1, 1, x, 0, 0, 0, 1, 0] -> [1, 0, x, 1];
```

240799-47

```
" [c, 1, 1, x, 0, 1, 0, 1, 0, 1, 0, 1, 0] -> [0, 1, 1, 1];
" [c, 1, 0, 0, 0, 1, 1, x, 0, 1, 0, 1, 0] -> [1, 0, x, 1];
" [c, 1, 1, x, 0, 1, 1, x, 0, 1, 0, 1, 0] -> [1, 0, x, 1];
" [c, 1, 1, 1, x, 1, 1, x, 0, 1, 0, 1, 0] -> [1, 0, x, 1];
"

end SC_MODE_DRAM_CTRL_11;
```

240799-48

```
module   SC_MODE_DRAM_CTRL_8   flag '-r4'

title   'STATIC COLUMN MODE DRAM CONTROLLER - PLD 8, INTEL CORPORATION'
" This PLD generates CAS1 (CAS for bank 1)
" Implemented with the Intel 85C220 EPLD.

   SC8   device   'E0320';

   x              =     .X.;                  " ABEL 'don't care' symbol
   c              =     .C.;                  " ABEL 'clocking input' symbol

" Inputs

   CLK pin 1; "P4 input CLK"
   RFACK          pin   2; "Refresh Acknowledge
   CIP~           pin   3; "Cycle OK
   LA2 pin 4; "Latched A2.
   HIT~           pin   5; "DRAM Page Hit Signal
   BOFF~          pin   6; "Backoff input to P4"
   LW_R~          pin   7; "
   RAS~           pin   8; "
   RESET          pin   9; "System Reset
   RDY~           pin 12; " Processor RDY#
   MEMCS~         pin 13; " Memory Chip Select
   BRDY~          pin 18; " Processor BREADY#
   BLAST~         pin 19; " Processor BLAST#

" Output

   CAS10~         pin 14; " CAS1 byte 0,2
   C1             pin 15; " state variable
   C2             pin 16; " state variable
   CAS11~         pin 17; " CAS1 byte 1,3

state_diagram [CAS10~,CAS11~,C1,C2]

   state [1, 1, 1, 1]:  if RESET # !BOFF~ then [1, 1, 1, 1] else
                          if !RFACK & !CIP~ & LA2 & LW_R~ & !MEMCS~ then
                          [1, 1, 0, 1] else if !RFACK & !CIP~ & !LW R~ & !RAS~
                          & !HIT~ & !MEMCS~ # (RFACK & RAS~) then
                          [0, 0, 1, 1] else [1, 1, 1, 1];

   state [1, 1, 0, 1]:  if RESET # !BOFF~ then [1, 1, 1, 1] else
                        if !RAS~ & RDY~ then [0, 0, 0, 0] else
                        [1, 1, 0, 1];

   state [0, 0, 0, 0]:  if RESET # !BOFF~ then [1, 1, 1, 1] else
                          if !CIP~ & LA2 & LW_R~ & !MEMCS~ then [1, 1, 0, 0] else
                          if CIP~ # (!CIP~ & (MEMCS~ # !LW_R~)) # (!CIP~ & LW_R~&
                          !LA2) then [1, 1, 1, 1] else [0, 0, 0, 0];

   state [1, 1, 0, 0]:  if !RAS~ then [0, 0, 0, 0] else [1, 1, 0, 0];
```

240799-49

5

```
  state [0, 0, 1, 1]:  if RESET # !BOFF~ then [1, 1, 1, 1] else
                       if !BRDY~ & !BLAST~ & !RFACK then
                       [1, 1, 1, 1] else
                         if !BRDY~ & BLAST~ & LA2 then [1, 1, 1, 0] else
                         if !BRDY~ & BLAST~ & !LA2 then [0, 0, 1, 0] else
                         if RFACK then [0, 0, 1, 0] else
                         if BRDY~ & !RFACK then [0, 0, 1, 1];

  state [1, 1, 1, 0]:  if RESET then [1, 1, 1, 1] else
                         if !BOFF~ then [1, 1, 1, 0] else
                         if !BRDY~ & BLAST~ then [0, 0, 1, 1] else
                         if !BRDY~ & !BLAST~ then [1, 1, 1, 1] else
                         [1, 1, 1, 0];

  state [0, 0, 1, 0]:  if RESET # !BOFF~ then [1, 1, 1, 1] else
                         if !BRDY~ & BLAST~ then [1, 1, 1, 0] else
                         if !BRDY~ & !BLAST~ # BRDY~ then [1, 1, 1, 1];

test_vectors

  ([CLK,RFACK,CIP~,LA2,HIT,~BOFF~,LW_R~,RAS~,RESET,RDY~,MEMCS~,BRDY~]
-> [BLAST~,CAS10~,C1,C2,CAS11~])

" C R A L H B L R R R M B B    C C C C
" L F Q A I O W A E D E R L    A 1 2 A
" K A 0 2 T F R S S Y M D A    S   S
"  C ~  ~ F ~  ~ E ~ C Y S    0   0
"  K    ~    T  S   T  0   1
"              ~   ~  ~   ~
"
"
"

  [c, x, x, x, x, x, 1, x, 1, x, x, x, x] -> [x, x, x, x];
  [c, x, 1, 1, x, x, 1, x, 1, x, x, x, x] -> [1, 1, 1, x];
  [c, x, 1, 1, x, 1, 1, 0, 0, 1, 0, 1, 1] -> [1, 1, 1, 1];
  [c, 0, 1, 1, x, 1, 1, 0, 0, 1, 0, 1, 1] -> [1, 1, 1, 1];
  [c, 0, 1, 1, x, 1, 1, 0, 0, 1, 0, 1, 1] -> [1, 1, 1, 1];
  [c, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1] -> [1, 0, 1, 1];
  [c, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1] -> [1, 0, 1, 1];
  [c, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1] -> [0, 0, 0, 0];
  [c, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1] -> [1, 0, 0, 1];
  [c, 0, 1, x, 0, 1, 1, 0, 0, 1, 0, 1, 1] -> [0, 0, 0, 0];
  [c, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1] -> [1, 0, 0, 1];
  [c, 0, 1, x, 0, 1, 1, 0, 0, 1, 0, 1, 1] -> [0, 0, 0, 0];
  [c, 0, 1, 0, x, 1, 1, 0, 0, 1, 0, 1, 1] -> [1, 1, 1, 1];
end SC_MODE_DRAM_CTRL_8;
```

240799-50

```
module   PG_MODE_DRAM_CTRL_2   flag '-r4'

title     'PAGE MODE DRAM CONTROLLER - PLD 2, INTEL CORPORATION'
" This PLD generates CAS0
" Implemented with the Intel 85C220 EPLD.

  SC2   device   'E0320';

  x          =    .X.;              " ABEL 'don't care' symbol
  c          =    .C.;              " ABEL 'clocking input' symbol

" Inputs

  CLK pin 1; "P4 input CLK"
  RFACK      pin  2; "Refresh Acknowledge
  CIP~       pin  3; "Cycle OK
  LA2 pin 4; "Latched A2.
  HIT~       pin  5; "DRAM Page Hit Signal
  BOFF~      pin  6; "Backoff input to P4"
  LW_R~      pin  7; "
  RAS~       pin  8; "
  RESET      pin  9; "System Reset
  RDY~       pin 12; "Processor RDY#
  MEMCS~     pin 13; "Memory Chip Select
  BRDY~      pin 18; "Processor BREADY#
  BLAST~     pin 19; "Processor BLAST#

" Output

  CAS10~     pin 14; " CAS1 byte 0,2
  C1         pin 15; " state variable
  C2         pin 16; " state variable
  CAS11~     pin 17; " CAS1 byte 1,3

state_diagram [CAS10~, CAS11~, C1, C2]

  state [1, 1, 1, 1]:  if RESET # !BOFF~ then [1, 1, 1, 1] else
                         if !RFACK & !CIP~ & !LA2 & LW R~ & !MEMCS~ then
                         [1, 1, 0, 1] else if !RFACK & !CIP~ & !LW_R~ & !RAS~
                         & !HIT~ & !MEMCS~ # (RFACK & RAS~) then
                         [0, 0, 1, 1] else [1, 1, 1, 1];

  state [1, 1, 0, 1]:  if RESET # !BOFF~ then [1, 1, 1, 1] else
                       if !RAS~ & RDY~ then [0, 0, 0, 0] else
                       [1, 1, 0, 1];

  state [0, 0, 0, 0]:  if RESET # !BOFF~ then [1, 1, 1, 1] else
                       if !CIP~ & !LA2 & LW_R~ & !MEMCS~ then [1, 1, 0, 0] else
                       if CIP~ # (!CIP~ & (MEMCS~ # !LW_R)) # (!CIP~ & LW_R~ &
                       LA2) then [1, 1, 1, 1] else [0, 0, 0, 0];

  state [1, 1, 0, 0]:  if !RAS~ then [0, 0, 0, 0] else [1, 1, 0, 0];
```

240799-51

5

```
    state [0, 0, 1, 1]:  if RESET # !BOFF~ then [1, 1, 1, 1] else
                         if !BRDY~ & !BLAST~ & !RFACK then
                         [1, 1, 1, 1] else
                           if !BRDY~ & BLAST~ & !LA2 then [1, 1, 1, 0] else
                           if !BRDY~ & BLAST~ & LA2 then [0, 0, 1, 0] else
                           if RFACK then [0, 0, 1, 0] else
                           if BRDY~ & !RFACK then [0, 0, 1, 1];

    state [1, 1, 1, 0]:  if RESET then [1, 1, 1, 1] else
                           if !BOFF~ then [1, 1, 1, 0] else
                           if !BRDY~ & BLAST~ then [0, 0, 1, 1] else
                           if !BRDY~ & !BLAST~ then [1, 1, 1, 1] else
                           [1, 1, 1, 0];

    state [0, 0, 1, 0]:  if RESET # !BOFF~ then [1, 1, 1, 1] else
                           if !BRDY~ & BLAST ~ then [1, 1, 1, 0] else
                           if !BRDY~ & !BLAST~ # BRDY~ then [1, 1, 1, 1];

test_vectors

   ([CLK,RFACK,CIP~,LA2,HIT~,BOFF~,LW_R~,RAS~,RESET,RDY~,MEMCS~,BRDY~]
   - >      [BLAST~,CAS10~,C1,C2,CAS11~])

" C R A L H B L R R R M B B     C C C C
" L F Q A I O W A E D E R L     A 1 2 A
" K A 0 2 T F R S S Y M D A     S     S
"   C ~    ~ F ~ ~ E ~ C Y S   0     0
"   K        ~    T    S    T    0    1
"                    ~    ~    ~    ~
"
"
"

   [c, x, x, x, x, x, 1, x, 1, x, x, x, x] - > [x, x, x, x];
   [c, x, 1, 0, x, x, 1, x, 1, x, x, x, x] - > [1, 1, 1, 1];
   [c, 0, 1, 0, x, 1, 1, 0, 0, 1, 0, 1, 1] - > [1, 1, 1, 1];
   [c, 0, 1, 0, x, 1, 1, 0, 0, 1, 0, 1, 1] - > [1, 1, 1, 1];
   [c, 0, 1, 0, x, 1, 1, 0, 0, 1, 0, 1, 1] - > [1, 1, 1, 1];
   [c, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1] - > [1, 0, 1, 1];
   [c, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1] - > [1, 0, 1, 1];
   [c, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1] - > [0, 0, 0, 0];
   [c, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1] - > [1, 0, 0, 1];
   [c, 0, 1, x, 0, 1, 1, 0, 0, 1, 0, 1, 1] - > [0, 0, 0, 0];
   [c, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1] - > [1, 0, 0, 1];
   [c, 0, 1, x, 0, 1, 1, 0, 0, 1, 0, 1, 1] - > [0, 0, 0, 0];
   [c, 0, 1, 1, x, 1, 1, 0, 0, 1, 0, 1, 1] - > [1, 1, 1, 1];
end PG_MODE_DRAM_CTRL_2;
```

240799-52

```
module   SC_MODE_DRAM_CTRL_15 flag '-r4'

title  'STATIC COLUMN MODE DRAM CONTROLLER - PLD 15, INTEL CORPORATION'
" This PLD combines ready signals
" Implemented with the Intel 85C220 EPLD.

  SC15K device   'E0320';

  x          =    .X.;              " ABEL 'don't care' symbol
  c          =    .C.;              " ABEL 'clocking input' symbol

" Inputs

  MEMCS~    pin  1; "
  JRDY~     pin  2; "
  MRDY~     pin  3; "
  BRDY~     pin  4; "
  ALD pin 5; "
  CKEN~     pin  6; "
  SKEN~     pin  7; "
  BRDYO~    pin  8; "
  M~        pin  9; "miss indicator for CIP~
  CIP~      pin 11; " Cycle indicator

" Output

  WEN~      pin 12; "Write enable for write latches
  RDY~      pin 13; "to 486
  MRDYCS~   pin 14; "
  MALD~     pin 15; "Modified ALD for FF's
  dum10     pin 16; "
  PBRDY~    pin 17; "
  KEN~      pin 18; "
  DRDY~     pin 19; "

equations

  !MALD~ = (!MEMCS~ & !ALD);

  !RDY~ = (!MRDY~ & M~ & !MEMCS~) # !JRDY~;

  !MRDYCS~ = (!MRDY~ & M~ & !MEMCS~);

  !WEN~ = !CIP~ & M~;

  !DRDY~ = !BRDY~ # !MRDYCS~;

  KEN~ = SKEN~ & CKEN~;

  PBRDY~ = BRDY~ & BRDYO~;
"test_vectors
```

240799-53

5

```
" ([CLK,RESET] - >
" [RESETO])

" C R    R
" L E    E
" K S    S
"    E    E
"    T    T
"         O
"
"
"

" [c, 0] - > [x];
" [c, 0] - > [0];
" [c, 0] - > [0];
" [c, 0] - > [0];
" [c, 0] - > [0];
"[c, 0] - > [0];
" [c, 0] - > [0];
" [c, 1] - > [1];
" [c, 1] - > [1];
" [c, 1] - > [1];
" [c, 1] - > [1];
" [c, 1] - > [1];
" [c, 1] - > [1];
" [c, 1] - > [1];
" [c, 1] - > [1];
" [c, 0] - > [0];
" [c, 0] - > [0];
" [c, 0] - > [0];

end SC_MODE_DRAM_CTRL_15;
```

240799-54

```
module   SC_MODE_DRAM_CTRL_17 flag '-r4'

title  'STATIC COLUMN MODE DRAM CONTROLLER - PLD 17, INTEL CORPORATION'
" This PLD generates the A0 signal for bank 1
" Implemented with the Intel 85C224 EPLD.

  SC17 device  'E224';

  x          =    .X.;              " ABEL 'don't care' symbol
  c          =    .C.;              " ABEL 'clocking input' symbol

" Inputs

  CLK pin 1; "P4 input CLK"
  BRDY~      pin  2; "Burst Ready
  CIP~       pin  3; "Cycle OK
  MEMCS~     pin  4; "memory select
  LA313      pin  5; "Latched A2.
  DATASEL    pin  6; "Refresh acknowledge"
  RAS~       pin  7; "Row address strobe
  LW_R       pin  8; "CPU W/R latched~
  RESET      pin  9; "System Reset
  BLAST~     pin 10; "CPU BLAST~ output
  A3         pin 11; "CPU Backoff input
  ALD pin 14; "Address Latch disable
  dum1       pin 15;
  WE1~       pin 22; "Write enable
  dum2       pin 23; "Address Latch disable

" Output

  B10MA0     pin 21; "Bank 1 A0
  B1A pin 20; "Burst A3 bank0
  CS0~       pin 19; " state variable
  dun pin 18; " state variable
  dum        pin 17; " Burst A3 bank1
  B11MA0     pin 16; "Bank 1 A0

state_diagram [B1A, CS0~]

  state [1, 1]: if RESET then [1, 1] else
                  if CIP~ & !ALD & !A3 then [0, 1] else
                  if !CIP~ & !ALD & !A3 then [0, 1] else
                  if !CIP~ & !LW_R & !MEMCS~ & WE1~ then [1, 0] else [1, 1];

  state [0, 1]: if RESET then [1, 1] else
                  if CIP~ & !ALD & A3 then [1, 1] else
                  if !CIP~ & !ALD & A3 then [1, 1] else
                  if !CIP~ & !LW_R & !MEMCS~ & WE1~ then [0, 0] else [0, 1];

  state [1, 0]: if RESET # (!BRDY~ & !BLAST~) then [1, 1] else
                  if !BRDY~ & DATASEL then [0, 0] else [1, 0];
```

240799-55

5

```
    state [0, 0]: if RESET # (!BRDY~ & !BLAST~) then [1, 1] else
                  if !BRDY~ & DATASEL then [1, 0] else [0, 0];

equations

!B10MA0 = !WE1~ & !LA313 # WE1~ & RAS~ & !LA313 # WE1~ & !RAS~ & !B1A;

!B11MA0 = !WE1~ & !LA313 # WE1~ & RAS~ & !LA313 # WE1~ & !RAS~ & !B1A;

end SC_MODE_DRAM_CTRL_17;
```

240799-56

```
module  SC_MODE_DRAM_CTRL_6  flag '-r4'

title  'STATIC COLUMN MODE DRAM CONTROLLER - PLD 6, INTEL CORPORATION'
" This PLD generates A0 for bank 0
" Implemented with the Intel 85C224 EPLD.

  SC6  device  'E224';

  x          =    .X.;              " ABEL 'don't care' symbol
  c          =    .C.;              " ABEL 'clocking input' symbol

" Inputs

  CLK pin 1; "P4 input CLK"
  BRDY~      pin  2; "Burst Ready
  CIP~       pin  3; "Cycle OK
  MEMCS~     pin  4; "memory select
  LA313      pin  5; "Latched A2.
  DATASEL    pin  6; "Refresh acknowledge"
  RAS~       pin  7; "Row address strobe
  LW_R       pin  8; "CPU W/R latched~
  RESET      pin  9; "System Reset
  BLAST~     pin 10; "CPU BLAST~ output
  A3         pin 11; "CPU Backoff input
  ALD pin 14; "Address Latch disable
  dum1       pin 15;
  WE0~       pin 22; "Write enable
  dum2       pin 23; "Address Latch disable

" Output

  B00MA0     pin 21; "Bank 0 A0
  B0A pin 20; " Burst A3 bank 0
  CS0~       pin 19; " state variable
  dun pin 18; " state variable
  dum        pin 17; " Burst A3 bank1
  B01MA0     pin 16; "Bank 0 A0

state_diagram [B0A, CS0~]

  state [1, 1]: if RESET then [1, 1] else
                  if CIP~ & !ALD & !A3 then [0, 1] else
                  if !CIP~ & !ALD & !A3 then [0, 1] else
                  if !CIP~ & !LW_R & !MEMCS~ & WE0~ then [1, 0] else [1, 1];

  state [0, 1]: if RESET then [1, 1] else
                  if CIP~ & !ALD & A3 then [1, 1] else
                  if !CIP~ & !ALD & A3 then [1, 1] else
                  if !CIP~ & !LW_R & !MEMCS~ & WE0~ then [0, 0] else [0, 1];

  state [1, 0]: if RESET # (!BRDY~ & !BLAST~) then [1, 1] else
                  if !BRDY~ & !DATASEL then [0, 0] else [1, 0];
```

240799-57

5

```
state [0, 0]: if RESET # (!BRDY~ & !BLAST~) then [1, 1] else
                  if !BRDY~ & !DATASEL then [1, 0] else [0, 0];

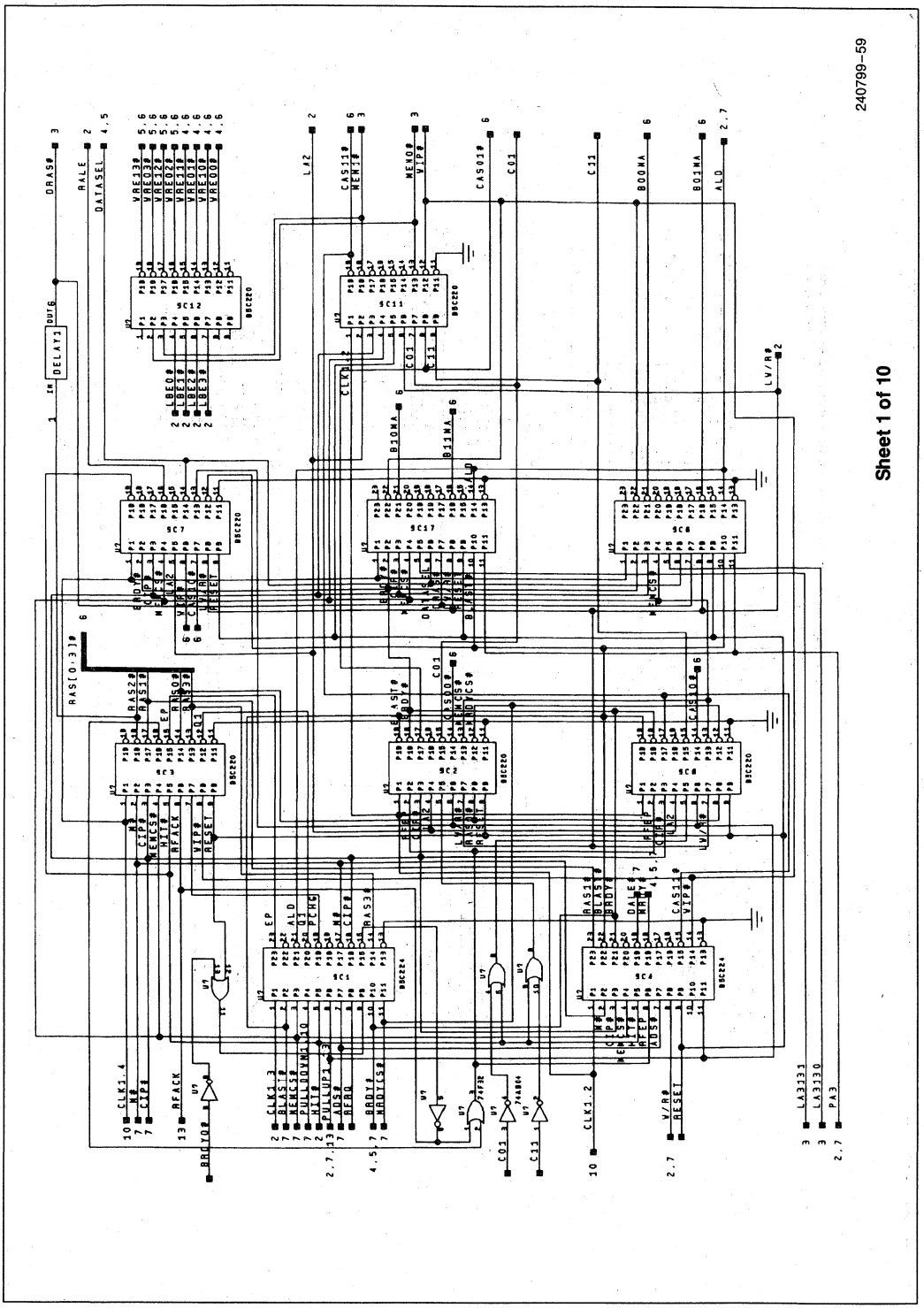
equations

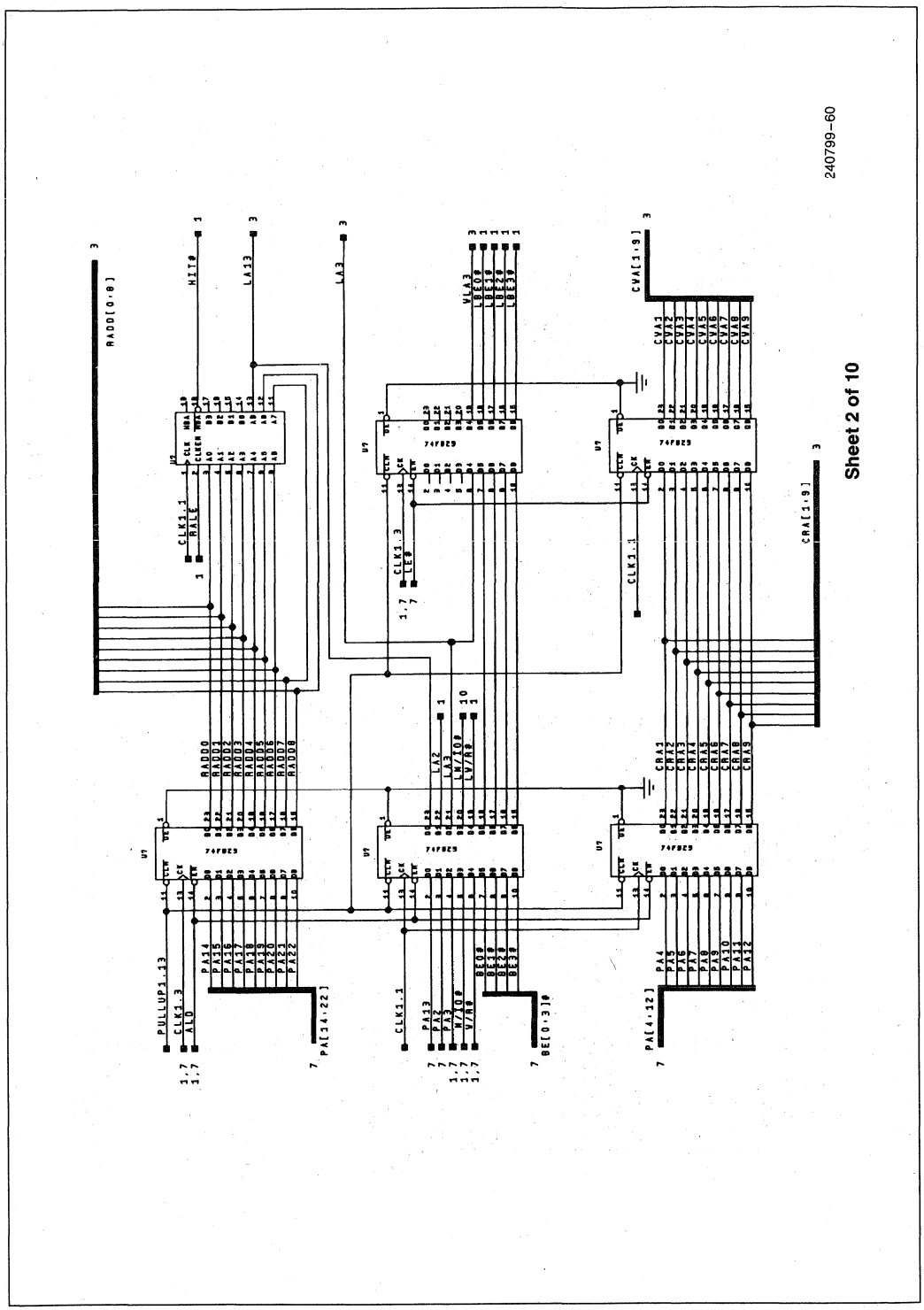
!B00MA0 = !WE0~ & !LA313 # WE0~ & RAS~ & !LA313 # WE0~ & !RAS~ & !B0A;

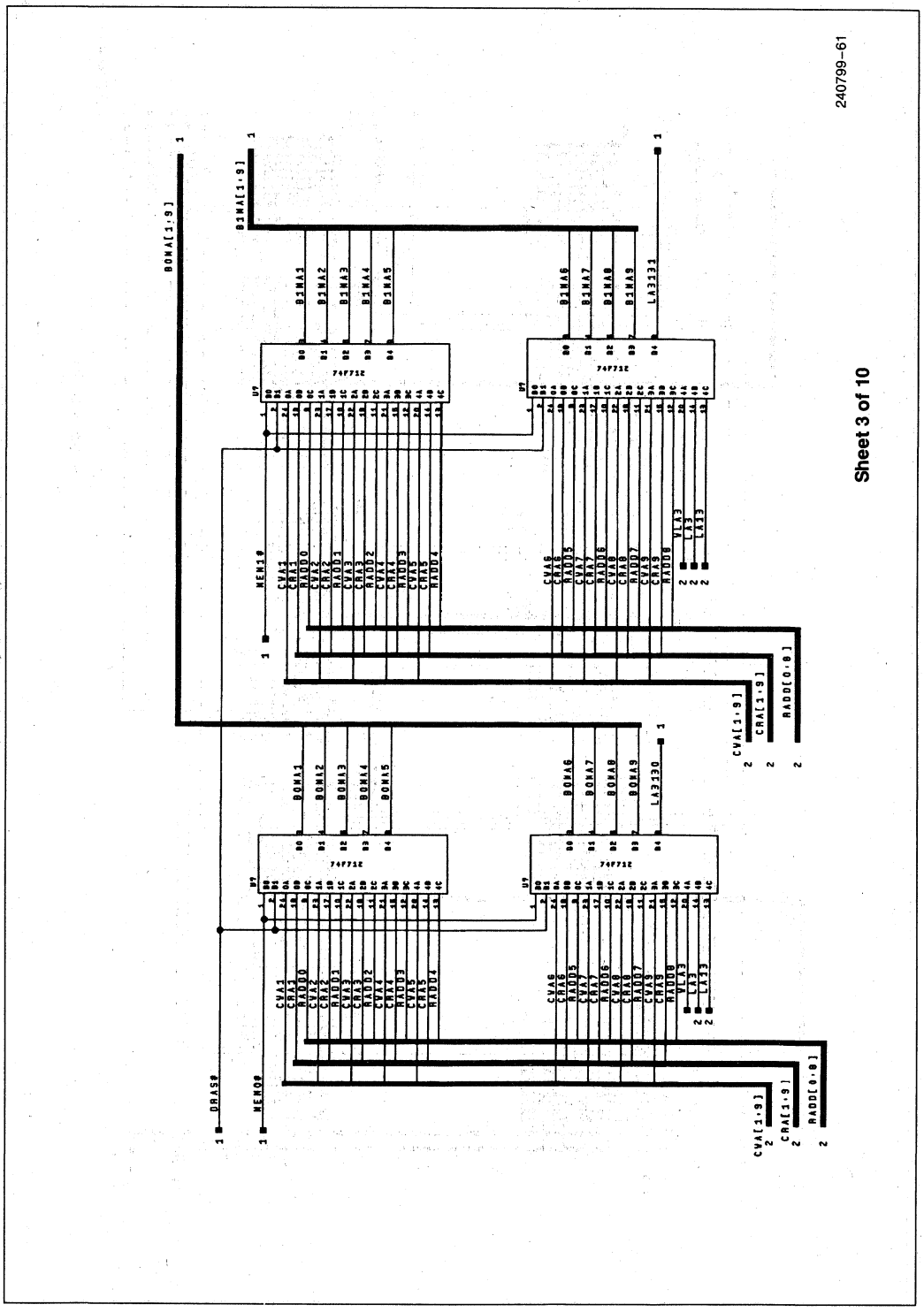
!B01MA0 = !WE0~ & !LA313 # WE0~ & RAS~ & !LA313 # WE0~ & !RAS~ & !B0A;

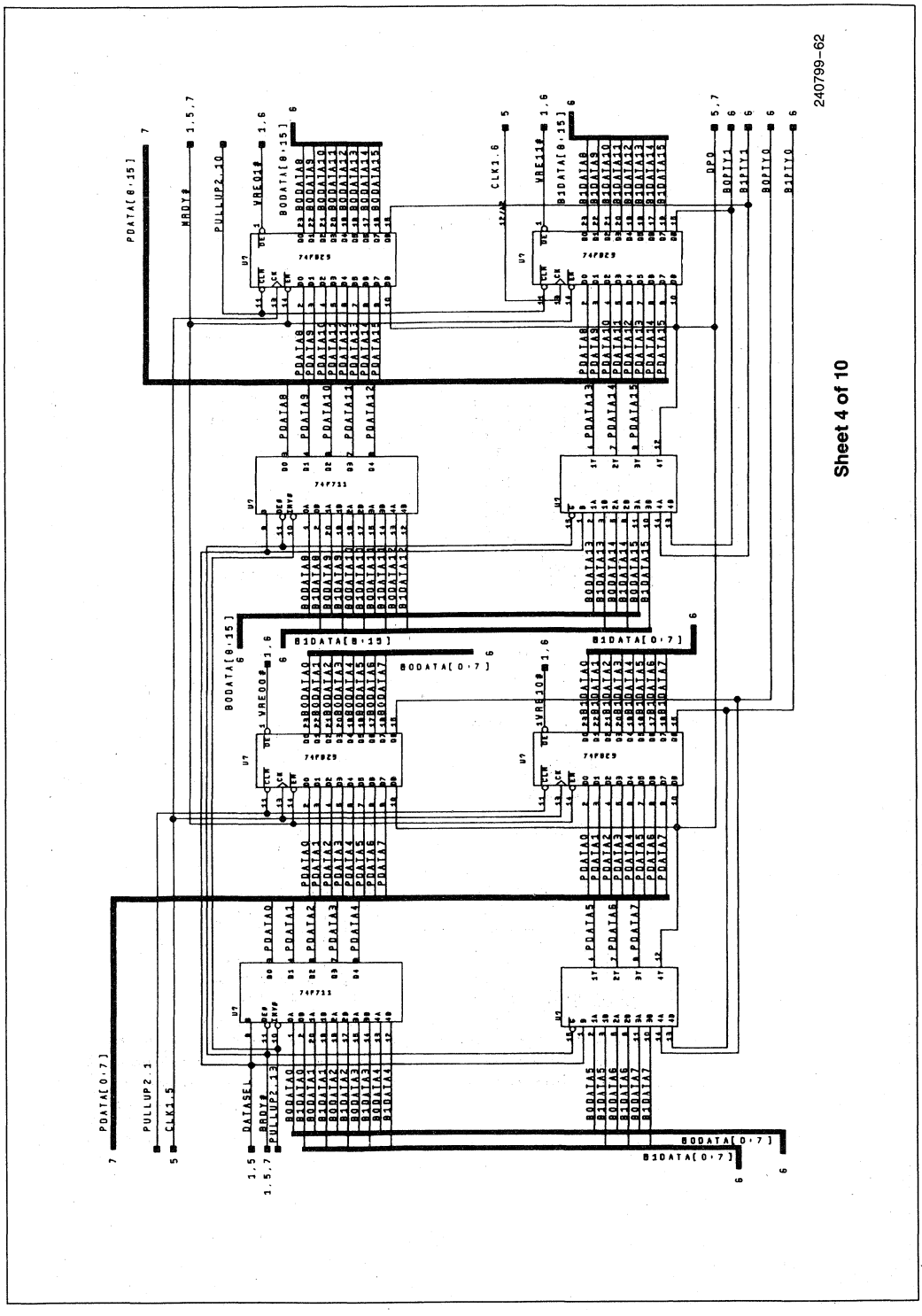
end SC_MODE_DRAM_CTRL_6;
```

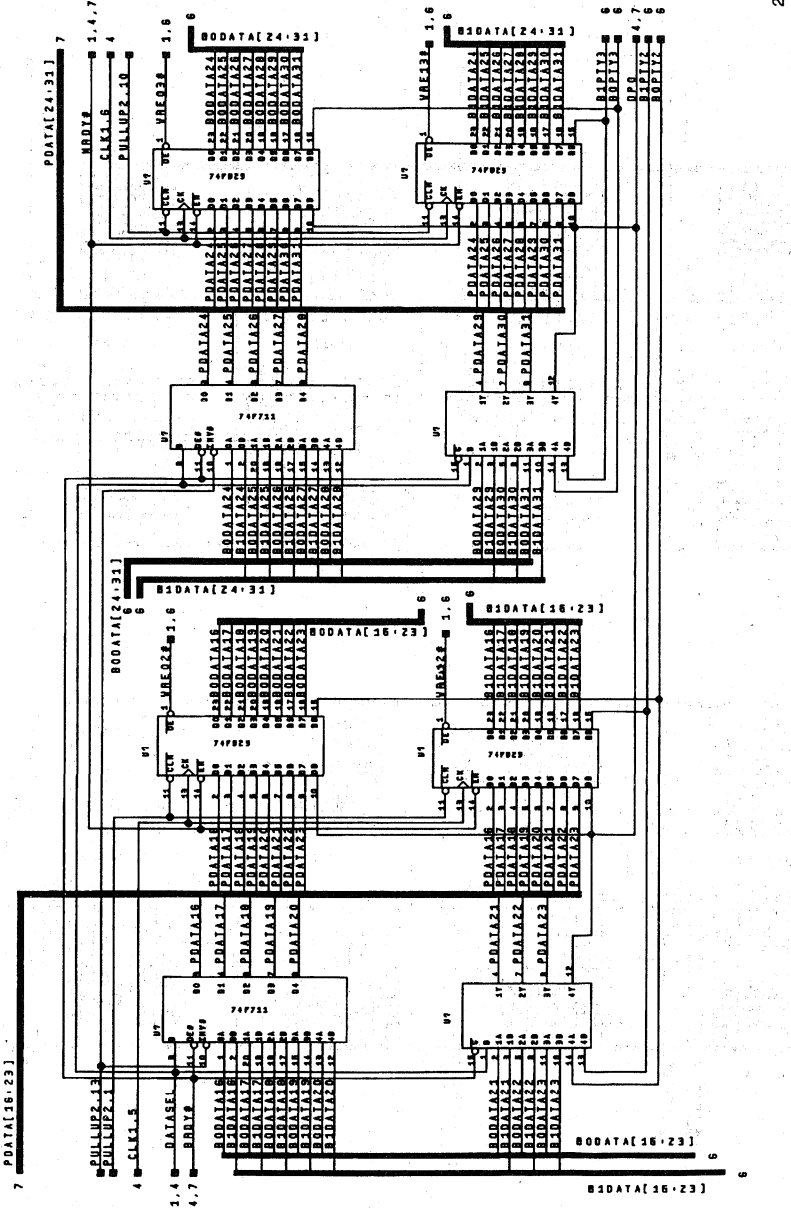
240799-58

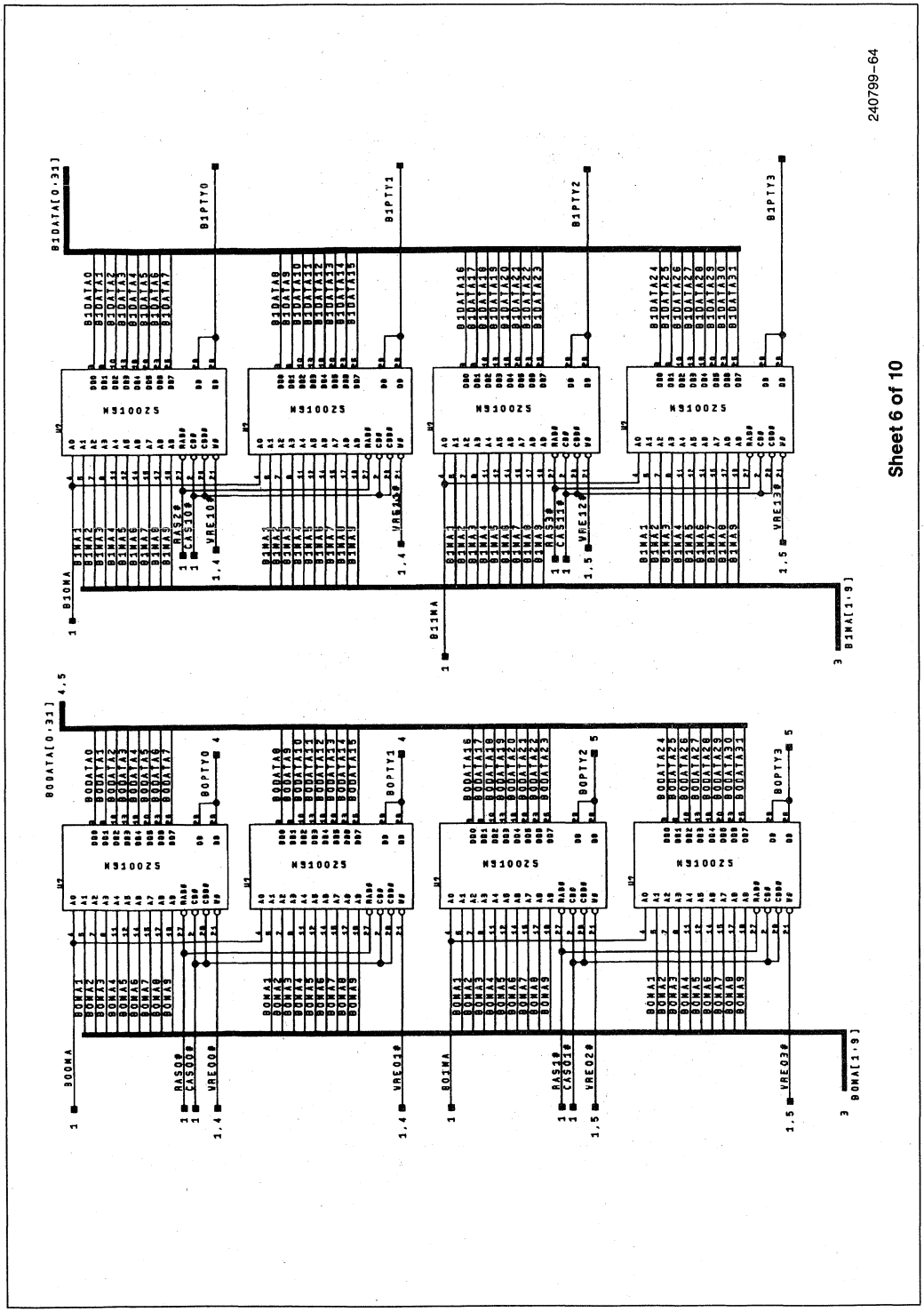

240799-59
Sheet 1 of 10

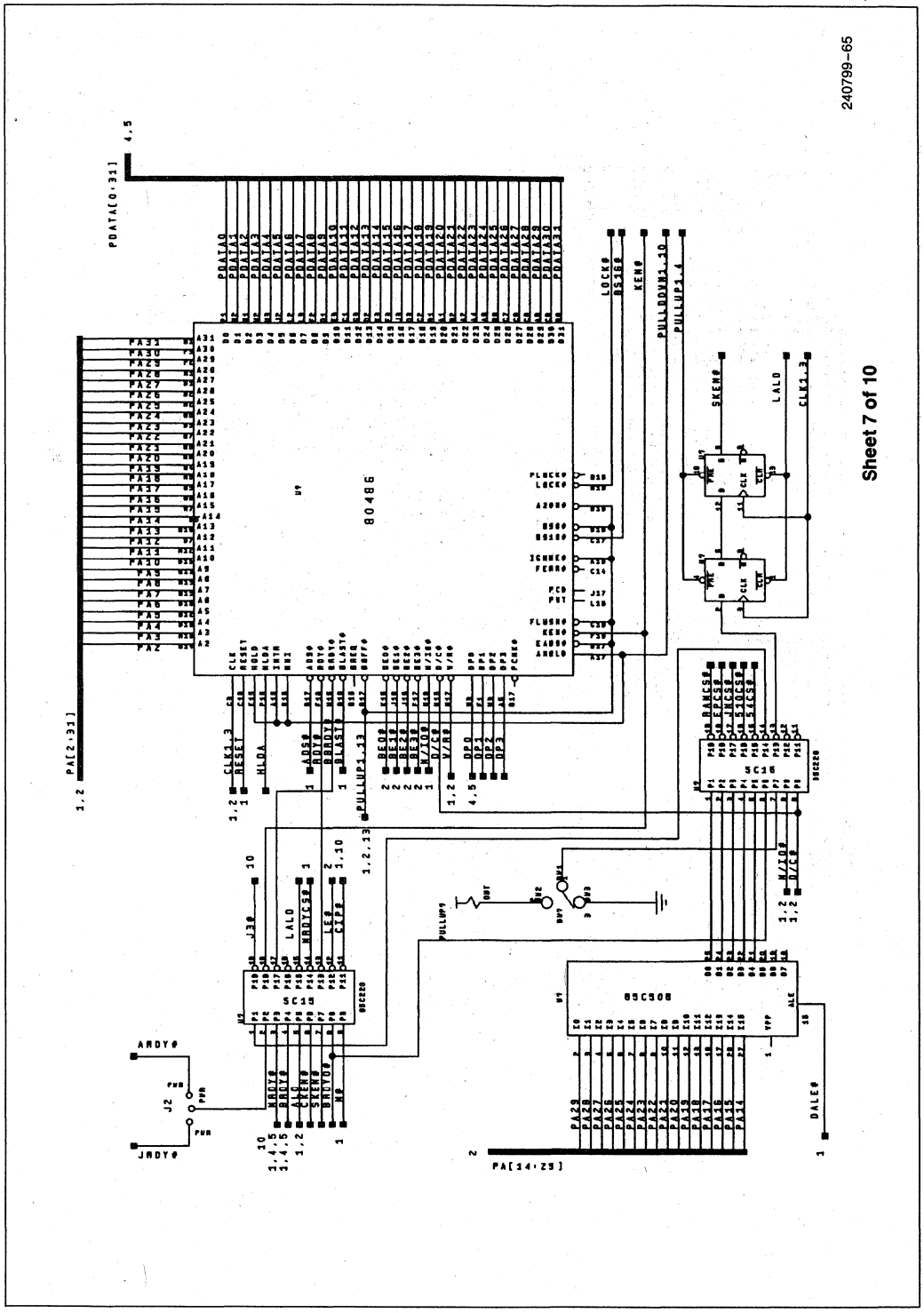

240799-60
Sheet 2 of 10

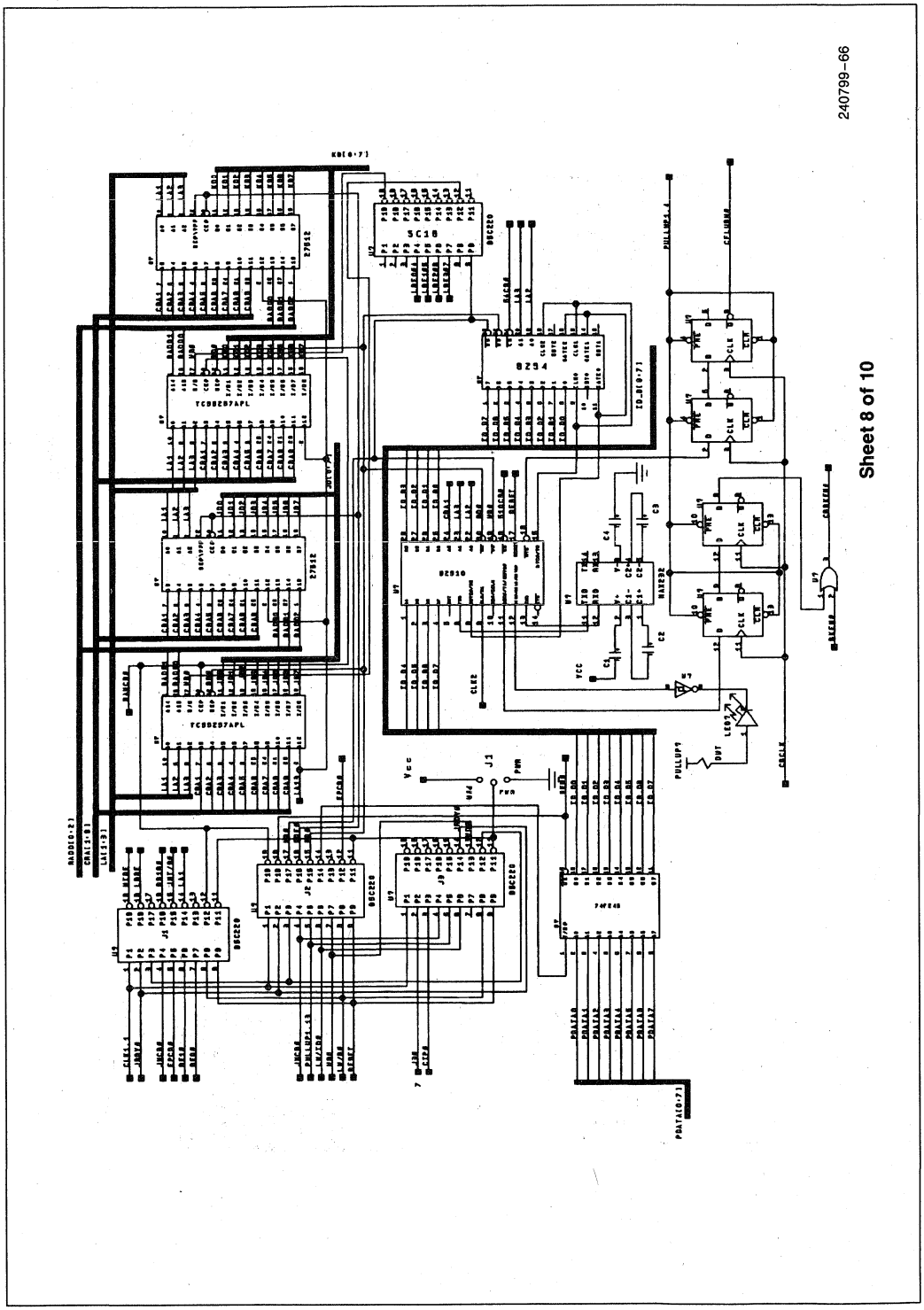

240799-61
Sheet 3 of 10
BOMA[1:9]
B1MA[1:9]
DRAS#
MEM0#
MEM1#
CVA[1:9]
CRA[1:9]
RADD[0:8]
74F732
LA3330
LA3331
LA33
VLA3

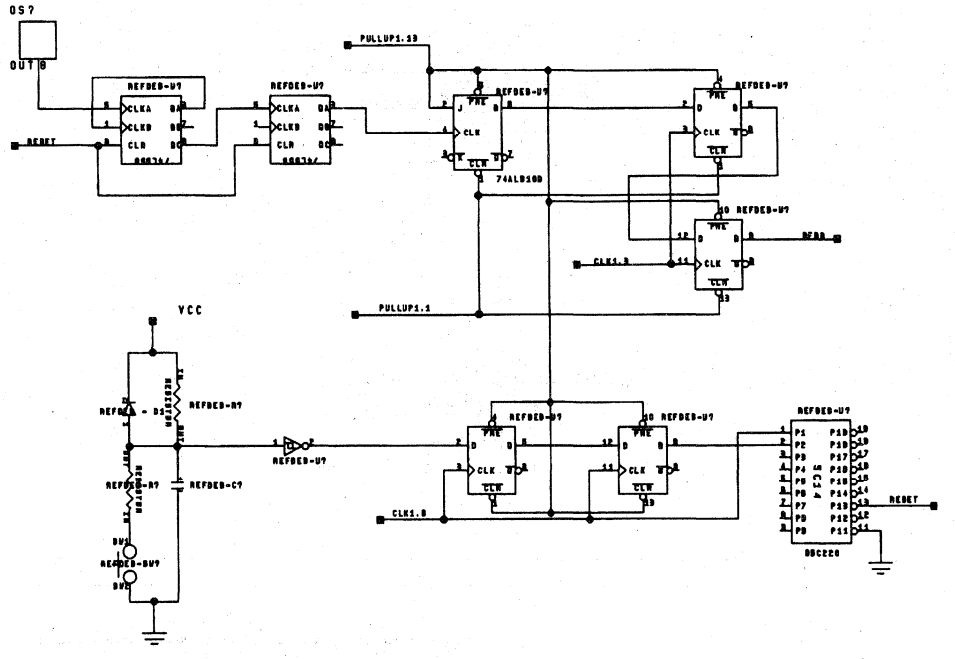

240799-62
Sheet 4 of 10
PDATA[0:15]
MRDY#
PULLUP2.10
VRE01#
BODATA[0:15]
CLK1.6
VRE11#
B1DATA[0:15]
DPO
BOPTY1
B1PTY1
BOPTY0
B1PTY0
74FBE9
74F733
B1DATA[8:15]
BODATA[0:7]
VRE00#
VRE10#
B1DATA[0:7]
PDATA[0:7]
PULLUP2.1
CLK1.5
DATASEL
BRDY#
PULLUP2.13


240799-63
Sheet 5 of 10
PDATA[24:31]
BODATA[24:31]
B1DATA[24:31]
PDATA[16:23]
BODATA[16:23]
B1DATA[16:23]
74FB29
74F733


240799-64
Sheet 6 of 10

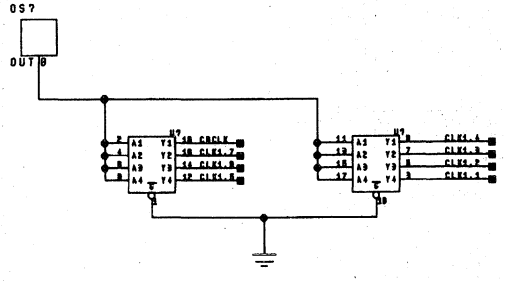

240799-65
Sheet 7 of 10
80486
82C206
82C606
SC15
PA[31:2]
PA[14:2]
PDATA[0:31]
ARDY#
JRDY#
DALE#
SKEN#
LALD
CLK4.3
LOCK#
KEN#
PULLUP#
PULLUP#3.10
PULLUP#1.4
RAMCS#
EPCS#
UHCS#
SIOCS#
RCCS#
M/IO#
D/C#
W/R#
HLDA
RESET
ADS#
RDY#
BRDY#
BLAST#
BREQ
DP0
DP1
DP2
DP3
LALD
BRDYCS#
LE#
CTP#
CPEN#
BRDYO#
HLD
ALE


240799-66
Sheet 8 of 10



240799-67

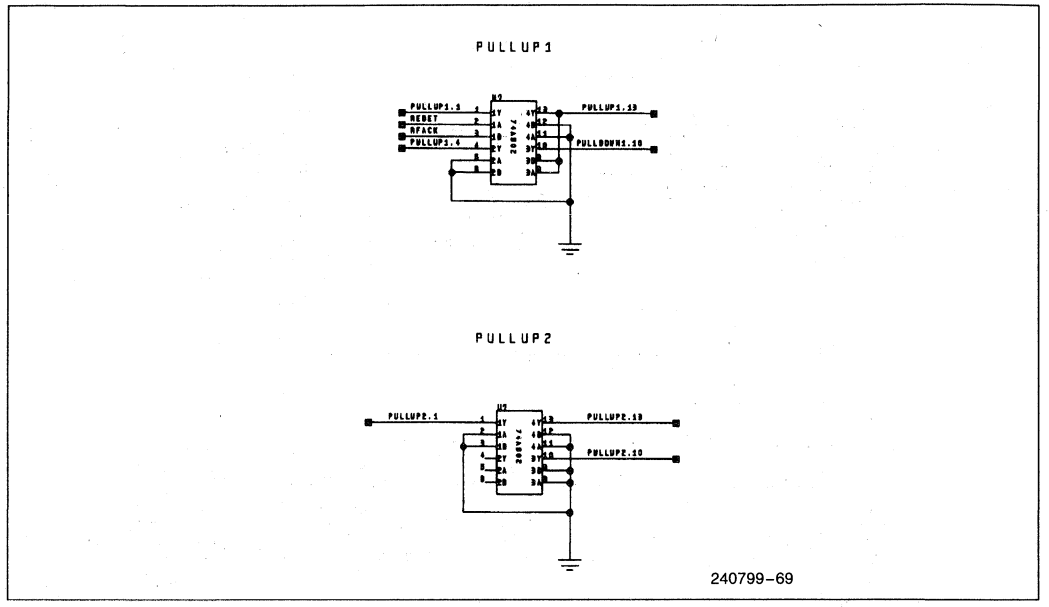
Sheet 9 of 10

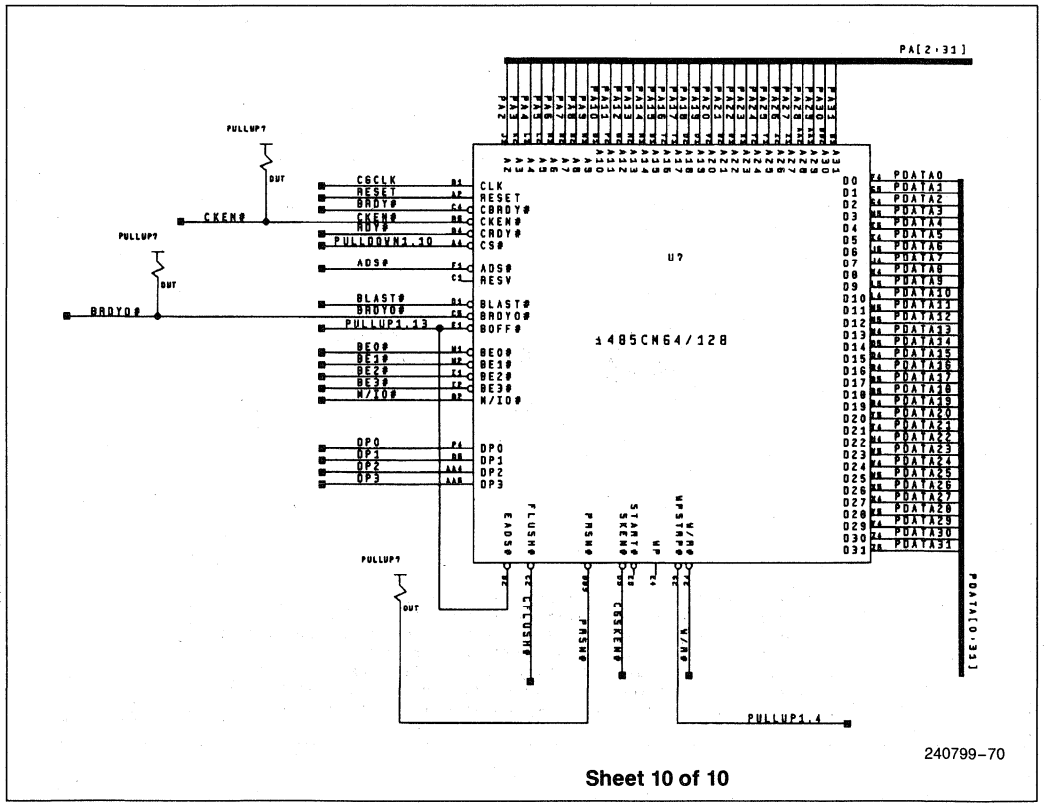


NOTE:

CLK1.1 -> U10 PIN13, U13 PIN13, U12 PIN13, J1,J2,J3 PIN1
CLK1.2 -> SC4 PIN1, SC8 PIN1, SC11 PIN1, SC2 PIN1
CLK1.3 -> PROC. PINC3, SC1 PIN1, U8 PIN13, U87 PIN13, U83 PIN3&11
CLK1.4 -> SC17 PIN1, SC7 PIN1, SC6 PIN1, SC3 PIN1
CLK1.5 -> U30 PIN13, U25 PIN13, U29 PIN13, U21PIN13, U24 PIN13
CLK1.6 -> U20 PIN13,U27PIN13,U33 PIN13, U85 PIN3&11, U86 PIN1
CLK1.7 -> U45 PIN1, U46 PIN10, U47 PIN1, U49 PIN2, U56 PIN6, U57 PIN23

240799-68





Sheet 10 of 10

386™ DX MICROPROCESSOR
HIGH PERFORMANCE 32-BIT CHMOS MICROPROCESSOR WITH INTEGRATED MEMORY MANAGEMENT

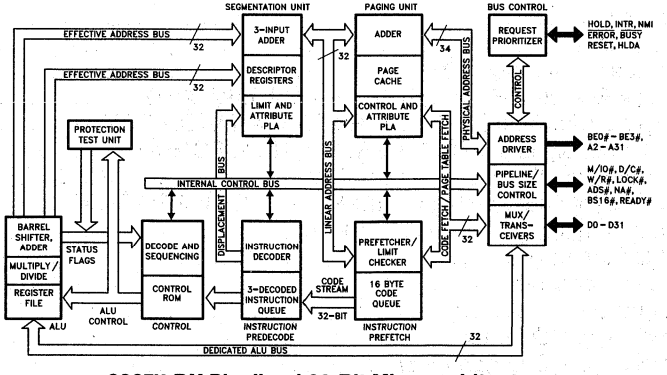
- **Flexible 32-Bit Microprocessor**
  - **— 8, 16, 32-Bit Data Types**
  - **— 8 General Purpose 32-Bit Registers**
- **Very Large Address Space**
  - **— 4 Gigabyte Physical**
  - **— 64 Terabyte Virtual**
  - **— 4 Gigabyte Maximum Segment Size**
- **Integrated Memory Management Unit**
  - **— Virtual Memory Support**
  - **— Optional On-Chip Paging**
  - **— 4 Levels of Protection**
  - **— Fully Compatible with 80286**
- **Object Code Compatible with All 8086 Family Microprocessors**
- **Virtual 8086 Mode Allows Running of 8086 Software in a Protected and Paged System**
- **Hardware Debugging Support**
- **Optimized for System Performance**
  - **— Pipelined Instruction Execution**
  - **— On-Chip Address Translation Caches**
  - **— 20, 25 and 33 MHz Clock**
  - **— 40, 50 and 66 Megabytes/Sec Bus Bandwidth**
- **High Speed Numerics Support via 387 DX™ Coprocessor**
- **Complete System Development Support**
  - **— Software: C, PL/M, Assembler System Generation Tools**
  - **— Debuggers: PSCOPE, ICE™-386**
- **High Speed CHMOS III and CHMOS IV Technology**
- **132 Pin Grid Array Package**

(See Packaging Specification, Order #231369)

The 386™ DX Microprocessor is an advanced 32-bit microprocessor designed for applications needing very high performance and optimized for multitasking operating systems. The 32-bit registers and data paths support 32-bit addresses and data types. The processor addresses up to four gigabytes of physical memory and 64 terabytes (2**46) of virtual memory. The integrated memory management and protection architecture includes address translation registers, advanced multitasking hardware and a protection mechanism to support operating systems. In addition, the 386 DX allows the simultaneous running of multiple operating systems. Instruction pipelining, on-chip address translation, and high bus bandwidth ensure short average instruction execution times and high system throughput.

The 386 DX offers new testability and debugging features. Testability features include a self-test and direct access to the page translation cache. Four new breakpoint registers provide breakpoint traps on code execution or data accesses, for powerful debugging of even ROM-based systems.

Object-code compatibility with all 8086 family members (8086, 8088, 80186, 80188, 80286) means the 386 DX offers immediate access to the world's largest microprocessor software base.



231630-49

386™ DX Pipelined 32-Bit Microarchitecture

386™ DX and 387™ DX are Trademarks of Intel Corporation.
UNIX™ is a Trademark of AT&T Bell Labs.
MS-DOS is a Trademark of MICROSOFT Corporation.

October 1990
Order Number: 231630-008

TABLE OF CONTENTS

CONTENTS

PAGE

1. PIN ASSIGNMENT 5-290
1.1 Pin Description Table 5-291

2. BASE ARCHITECTURE 5-293
2.1 Introduction 5-293
2.2 Register Overview 5-293
2.3 Register Descriptions 5-294
2.4 Instruction Set 5-300
2.5 Addressing Modes 5-303
2.6 Data Types 5-305
2.7 Memory Organization 5-307
2.8 I/O Space 5-308
2.9 Interrupts 5-309
2.10 Reset and Initialization 5-312
2.11 Testability 5-313
2.12 Debugging Support 5-313

3. REAL MODE ARCHITECTURE 5-317
3.1 Real Mode Introduction 5-317
3.2 Memory Addressing 5-318
3.3 Reserved Locations 5-319
3.4 Interrupts 5-319
3.5 Shutdown and Halt 5-319

4. PROTECTED MODE ARCHITECTURE 5-319
4.1 Introduction 5-319
4.2 Addressing Mechanism 5-320
4.3 Segmentation 5-321
4.4 Protection 5-331
4.5 Paging 5-337
4.6 Virtual 8086 Environment 5-341

5. FUNCTIONAL DATA 5-346
5.1 Introduction 5-346
5.2 Signal Description 5-346
5.2.1 Introduction 5-346
5.2.2 Clock (CLK2) 5-347
5.2.3 Data Bus (D0 through D31) 5-347
5.2.4 Address Bus (BEO# through BE3#, A2 through A31) 5-347
5.2.5 Bus Cycle Definition Signals (W/R#, D/C#, M/IO, LOCK#) 5-348
5.2.6 Bus Control Signals (ADS#, READY#, NA#, BS16#) 5-349
5.2.7 Bus Arbitration Signals (HOLD, HLDA) 5-350
5.2.8 Coprocessor Interface Signals (PEREQ, BUSY#, ERROR#) 5-350
5.2.9 Interrupt Signals (INTR, NMI, RESET) 5-351
5.2.10 Signal Summary 5-352
5.3. Bus Transfer Mechanism 5-352
5.3.1 Introduction 5-352
5.3.2 Memory and I/O Spaces 5-353
5.3.3 Memory and I/O Organization 5-354
5.3.4 Dynamic Data Bus Sizing 5-354
5.3.5 Interfacing with 32- and 16-bit Memories 5-355
5.3.6 Operand Alignment 5-356

CONTENTS

PAGE

5. FUNCTIONAL DATA (Continued)
5.4 Bus Functional Description 5-356
5.4.1 Introduction 5-356
5.4.2 Address Pipelining 5-359
5.4.3 Read and Write Cycles 5-361
5.4.4 Interrupt Acknowledge (INTA) Cycles 5-372
5.4.5 Halt Indication Cycle 5-373
5.4.6 Shutdown Indication Cycle 5-374
5.5 Other Functional Descriptions 5-375
5.5.1 Entering and Exiting Hold Acknowledge 5-375
5.5.2 Reset during Hold Acknowledge 5-375
5.5.3 Bus Activity During and Following Reset 5-375
5.6 Self-test Signature 5-377
5.7 Component and Revision Identifiers 5-377
5.8 Coprocessor Interface 5-379
5.8.1 Software Testing for Coprocessor Presence 5-379

6. INSTRUCTION SET 5-380
6.1 Instruction Encoding and Clock Count Summary 5-380
6.2 Instruction Encoding Details 5-395

7. DESIGNING FOR ICE™-386 DX EMULATOR USE 5-402

8. MECHANICAL DATA 5-404
8.1 Introduction 5-404
8.2 Package Dimensions and Mounting 5-404
8.3 Package Thermal Specification 5-407

9. ELECTRICAL DATA 5-408
9.1 Introduction 5-408
9.2 Power and Grounding 5-408
9.3 Maximum Ratings 5-409
9.4 D.C. Specifications 5-409
9.5 A.C. Specifications 5-410

10. REVISION HISTORY 5-422

NOTE:
This is revision 008; This supercedes all previous revisions.

1. PIN ASSIGNMENT

The 386 DX pinout as viewed from the top side of the component is shown by Figure 1-1. Its pinout as viewed from the Pin side of the component is Figure 1-2.

V_{CC} and GND connections must be made to multiple V_{CC} and V_{SS} (GND) pins. Each V_{CC} and V_{SS} must be connected to the appropriate voltage level. The circuit board should include V_{CC} and GND planes for power distribution and all V_{CC} and V_{SS} pins must be connected to the appropriate plane.

NOTE:
Pins identified as "N.C." should remain completely unconnected.

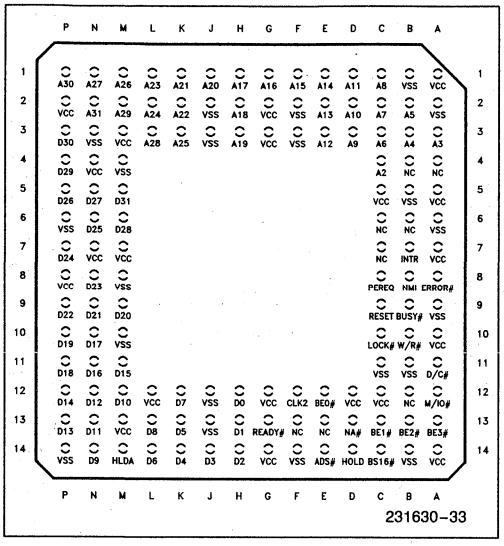


Figure 1-1. 386™ DX PGA Pinout—View from Top Side

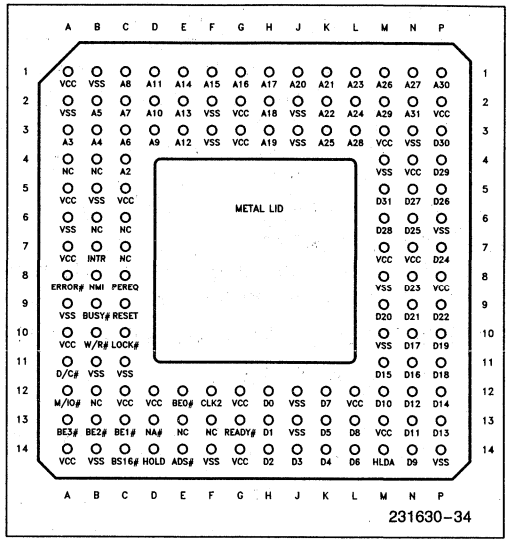


Figure 1-2. 386™ DX PGA Pinout—View from Pin Side

Table 1-1. 386™ DX PGA Pinout—Functional Grouping

Signal/Pin		Signal/Pin		Signal/Pin		Signal/Pin		Signal/Pin		Signal/Pin	
A2	C4	A24	L2	D6	L14	D28	M6	V_{CC}	C12	V_{SS}	F2
A3	A3	A25	K3	D7	K12	D29	P4		D12		F3
A4	B3	A26	M1	D8	L13	D30	P3		G2		F14
A5	B2	A27	N1	D9	N14	D31	M5		G3		J2
A6	C3	A28	L3	D10	M12	D/C#	A11		G12		J3
A7	C2	A29	M2	D11	N13	ERROR#	A8		G14		J12
A8	C1	A30	P1	D12	N12	HLDA	M14		L12		J13
A9	D3	A31	N2	D13	P13	HOLD	D14		M3		M4
A10	D2	ADS#	E14	D14	P12	INTR	B7		M7		M8
A11	D1	BE0#	E12	D15	M11	LOCK#	C10		M13		M10
A12	E3	BE1#	C13	D16	N11	M/IO#	A12		N4		N3
A13	E2	BE2#	B13	D17	N10	NA#	D13		N7		P6
A14	E1	BE3#	A13	D18	P11	NMI	B8		P2		P14
A15	F1	BS16#	C14	D19	P10	PEREQ	C8		P8	W/R#	B10
A16	G1	BUSY#	B9	D20	M9	READY#	G13	V_{SS}	A2	N.C.	A4
A17	H1	CLK2	F12	D21	N9	RESET	C9		A6		B4
A18	H2	D0	H12	D22	P9	V_{CC}	A1		A9		B6
A19	H3	D1	H13	D23	N8		A5		B1		B12
A20	J1	D2	H14	D24	P7		A7		B5		C6
A21	K1	D3	J14	D25	N6		A10		B11		C7
A22	K2	D4	K14	D26	P5		A14		B14		E13
A23	L1	D5	K13	D27	N5		C5		C11		F13

1.1 PIN DESCRIPTION TABLE

The following table lists a brief description of each pin on the 386 DX. The following definitions are used in these descriptions:

#	The named signal is active LOW.
I	Input signal.
O	Output signal.
I/O	Input and Output signal.
—	No electrical connection.

For a more complete description refer to Section 5.2 Signal Description.

Symbol	Type	Name and Function
CLK2	I	**CLK2** provides the fundamental timing for the 386 DX.
$D_{31}-D_0$	I/O	**DATA BUS** inputs data during memory, I/O and interrupt acknowledge read cycles and outputs data during memory and I/O write cycles.
$A_{31}-A_2$	O	**ADDRESS BUS** outputs physical memory or port I/O addresses.
BE0#–BE3#	O	**BYTE ENABLES** indicate which data bytes of the data bus take part in a bus cycle.
W/R#	O	**WRITE/READ** is a bus cycle definition pin that distinguishes write cycles from read cycles.
D/C#	O	**DATA/CONTROL** is a bus cycle definition pin that distinguishes data cycles, either memory or I/O, from control cycles which are: interrupt acknowledge, halt, and instruction fetching.
M/IO#	O	**MEMORY I/O** is a bus cycle definition pin that distinguishes memory cycles from input/output cycles.
LOCK#	O	**BUS LOCK** is a bus cycle definition pin that indicates that other system bus masters are denied access to the system bus while it is active.
ADS#	O	**ADDRESS STATUS** indicates that a valid bus cycle definition and address (W/R#, D/C#, M/IO#, BE0#, BE1#, BE2#, BE3# and $A_{31}-A_2$) are being driven at the 386 DX pins.
NA#	I	**NEXT ADDRESS** is used to request address pipelining.
READY#	I	**BUS READY** terminates the bus cycle.
BS16#	I	**BUS SIZE 16** input allows direct connection of 32-bit and 16-bit data buses.
HOLD	I	**BUS HOLD REQUEST** input allows another bus master to request control of the local bus.

1.1 PIN DESCRIPTION TABLE (Continued)

Symbol	Type	Name and Function
HLDA	O	**BUS HOLD ACKNOWLEDGE** output indicates that the 386 DX has surrendered control of its local bus to another bus master.
BUSY#	I	**BUSY** signals a busy condition from a processor extension.
ERROR#	I	**ERROR** signals an error condition from a processor extension.
PEREQ	I	**PROCESSOR EXTENSION REQUEST** indicates that the processor extension has data to be transferred by the 386 DX.
INTR	I	**INTERRUPT REQUEST** is a maskable input that signals the 386 DX to suspend execution of the current program and execute an interrupt acknowledge function.
NMI	I	**NON-MASKABLE INTERRUPT REQUEST** is a non-maskable input that signals the 386 DX to suspend execution of the current program and execute an interrupt acknowledge function.
RESET	I	**RESET** suspends any operation in progress and places the 386 DX in a known reset state. See **Interrupt Signals** for additional information.
N/C	—	**NO CONNECT** should always remain unconnected. Connection of a N/C pin may cause the processor to malfunction or be incompatible with future steppings of the 386 DX.
V_{CC}	I	**SYSTEM POWER** provides the +5V nominal D.C. supply input.
V_{SS}	I	**SYSTEM GROUND** provides 0V connection from which all inputs and outputs are measured.

2. BASE ARCHITECTURE

2.1 INTRODUCTION

The 386 DX consists of a central processing unit, a memory management unit and a bus interface.

The central processing unit consists of the execution unit and instruction unit. The execution unit contains the eight 32-bit general purpose registers which are used for both address calculation, data operations and a 64-bit barrel shifter used to speed shift, rotate, multiply, and divide operations. The multiply and divide logic uses a 1-bit per cycle algorithm. The multiply algorithm stops the iteration when the most significant bits of the multiplier are all zero. This allows typical 32-bit multiplies to be executed in under one microsecond. The instruction unit decodes the instruction opcodes and stores them in the decoded instruction queue for immediate use by the execution unit.

The memory management unit (MMU) consists of a segmentation unit and a paging unit. Segmentation allows the managing of the logical address space by providing an extra addressing component, one that allows easy code and data relocatability, and efficient sharing. The paging mechanism operates beneath and is transparent to the segmentation process, to allow management of the physical address space. Each segment is divided into one or more 4K byte pages. To implement a virtual memory system, the 386 DX supports full restartability for all page and segment faults.

Memory is organized into one or more variable length segments, each up to four gigabytes in size. A given region of the linear address space, a segment, can have attributes associated with it. These attributes include its location, size, type (i.e. stack, code or data), and protection characteristics. Each task on an 386 DX can have a maximum of 16,381 segments of up to four gigabytes each, thus providing 64 terabytes (trillion bytes) of virtual memory to each task.

The segmentation unit provides four-levels of protection for isolating and protecting applications and the operating system from each other. The hardware enforced protection allows the design of systems with a high degree of integrity.

The 386 DX has two modes of operation: Real Address Mode (Real Mode), and Protected Virtual Address Mode (Protected Mode). In Real Mode the 386 DX operates as a very fast 8086, but with 32-bit extensions if desired. Real Mode is required primarily to setup the processor for Protected Mode operation. Protected Mode provides access to the sophisticated memory management, paging and privilege capabilities of the processor.

Within Protected Mode, software can perform a task switch to enter into tasks designated as Virtual 8086 Mode tasks. Each such task behaves with 8086 semantics, thus allowing 8086 software (an application program, or an entire operating system) to execute. The Virtual 8086 tasks can be isolated and protected from one another and the host 386 DX operating system, by the use of paging, and the I/O Permission Bitmap.

Finally, to facilitate high performance system hardware designs, the 386 DX bus interface offers address pipelining, dynamic data bus sizing, and direct Byte Enable signals for each byte of the data bus. These hardware features are described fully beginning in Section 5.

2.2 REGISTER OVERVIEW

The 386 DX has 32 register resources in the following categories:

- General Purpose Registers
- Segment Registers
- Instruction Pointer and Flags
- Control Registers
- System Address Registers
- Debug Registers
- Test Registers.

The registers are a superset of the 8086, 80186 and 80286 registers, so all 16-bit 8086, 80186 and 80286 registers are contained within the 32-bit 386 DX.

Figure 2-1 shows all of 386 DX base architecture registers, which include the general address and data registers, the instruction pointer, and the flags register. The contents of these registers are task-specific, so these registers are automatically loaded with a new context upon a task switch operation.

The base architecture also includes six directly accessible segments, each up to 4 Gbytes in size. The segments are indicated by the selector values placed in 386 DX segment registers of Figure 2-1. Various selector values can be loaded as a program executes, if desired.

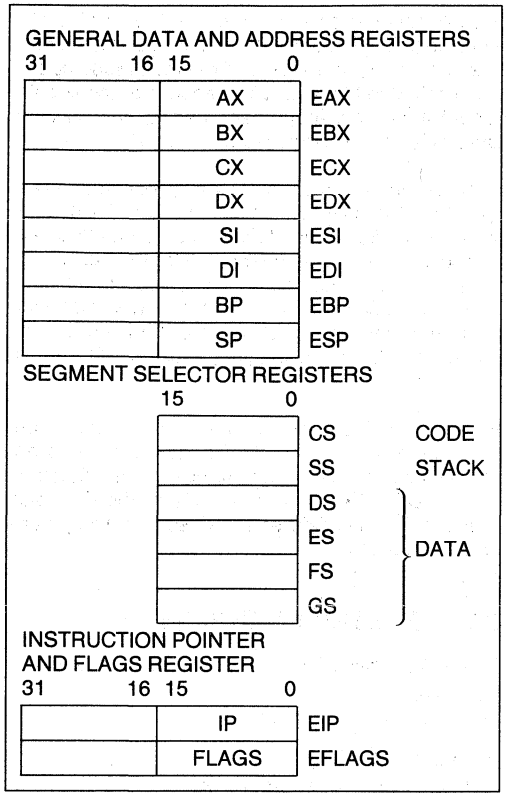


Figure 2-1. 386™ DX Base Architecture Registers

The selectors are also task-specific, so the segment registers are automatically loaded with new context upon a task switch operation.

The other types of registers, Control, System Address, Debug, and Test, are primarily used by system software.

2.3 REGISTER DESCRIPTIONS

2.3.1 General Purpose Registers

General Purpose Registers: The eight general purpose registers of 32 bits hold data or address quantities. The general registers, Figure 2-2, support data operands of 1, 8, 16, 32 and 64 bits, and bit fields of 1 to 32 bits. They support address operands of 16 and 32 bits. The 32-bit registers are named EAX, EBX, ECX, EDX, ESI, EDI, EBP, and ESP.

The least significant 16 bits of the registers can be accessed separately. This is done by using the 16-bit names of the registers AX, BX, CX, DX, SI, DI, BP, and SP. When accessed as a 16-bit operand, the upper 16 bits of the register are neither used nor changed.

Finally 8-bit operations can individually access the lowest byte (bits 0–7) and the higher byte (bits 8–15) of general purpose registers AX, BX, CX and DX. The lowest bytes are named AL, BL, CL and DL, respectively. The higher bytes are named AH, BH, CH and DH, respectively. The individual byte accessibility offers additional flexibility for data operations, but is not used for effective address calculation.

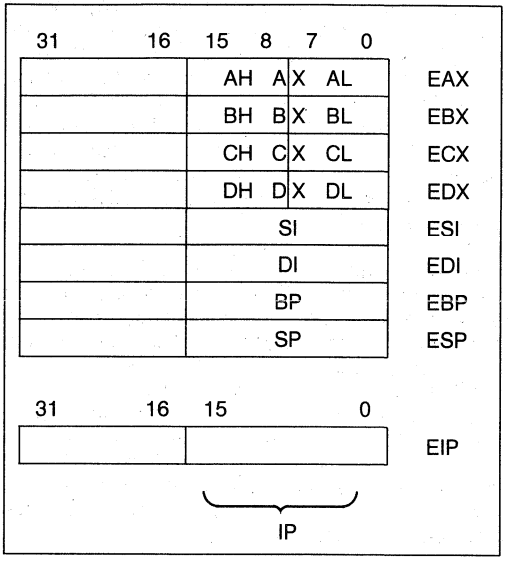


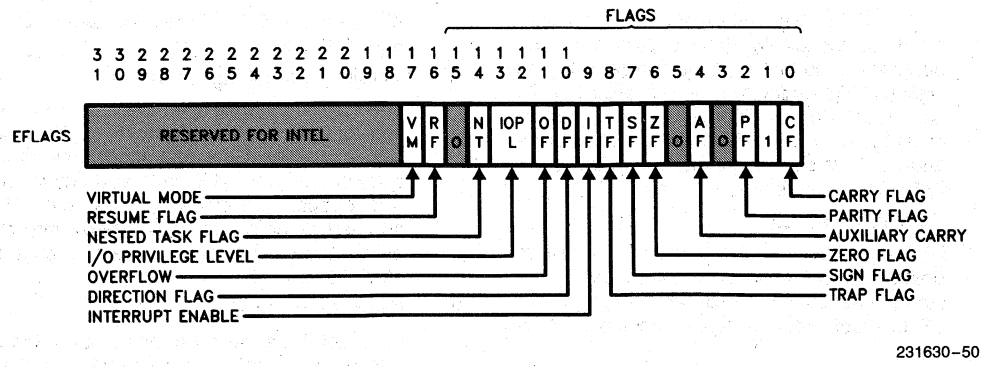
Figure 2-2. General Registers and Instruction Pointer

2.3.2 Instruction Pointer

The instruction pointer, Figure 2-2, is a 32-bit register named EIP. EIP holds the offset of the next instruction to be executed. The offset is always relative to the base of the code segment (CS). The lower 16 bits (bits 0–15) of EIP contain the 16-bit instruction pointer named IP, which is used by 16-bit addressing.

2.3.3 Flags Register

The Flags Register is a 32-bit register named EFLAGS. The defined bits and bit fields within EFLAGS, shown in Figure 2-3, control certain operations and indicate status of the 386 DX. The lower 16 bits (bit 0–15) of EFLAGS contain the 16-bit flag register named FLAGS, which is most useful when executing 8086 and 80286 code.



NOTE:
0 indicates Intel reserved: do not define; see section 2.3.10.

Figure 2-3. Flags Register

VM (Virtual 8086 Mode, bit 17)

The VM bit provides Virtual 8086 Mode within Protected Mode. If set while the 386 DX is in Protected Mode, the 386 DX will switch to Virtual 8086 operation, handling segment loads as the 8086 does, but generating exception 13 faults on privileged opcodes. The VM bit can be set only in Protected Mode, by the IRET instruction (if current privilege level = 0) and by task switches at any privilege level. The VM bit is unaffected by POPF. PUSHF always pushes a 0 in this bit, even if executing in virtual 8086 Mode. The EFLAGS image pushed during interrupt processing or saved during task switches will contain a 1 in this bit if the interrupted code was executing as a Virtual 8086 Task.

RF (Resume Flag, bit 16)

The RF flag is used in conjunction with the debug register breakpoints. It is checked at instruction boundaries before breakpoint processing. When RF is set, it causes any debug fault to be ignored on the next instruction. RF is then automatically reset at the successful completion of every instruction (no faults are signalled) except the IRET instruction, the POPF instruction, (and JMP, CALL, and INT instructions causing a task switch). These instructions set RF to the value specified by the memory image. For example, at the end of the breakpoint service routine, the IRET instruction can pop an EFLAG image having the RF bit set and resume the program's execution at the breakpoint address without generating another breakpoint fault on the same location.

NT (Nested Task, bit 14)

This flag applies to Protected Mode. NT is set to indicate that the execution of this task is nested within another task. If set, it indicates that the current nested task's Task State Segment (TSS) has a valid back link to the previous task's TSS. This bit is set or reset by control transfers to other tasks. The value of NT in EFLAGS is tested by the IRET instruction to determine whether to do an inter-task return or an intra-task return. A POPF or an IRET instruction **will** affect the setting of this bit according to the image popped, at any privilege level.

IOPL (Input/Output Privilege Level, bits 12-13)

This two-bit field applies to Protected Mode. IOPL indicates the numerically maximum CPL (current privilege level) value permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O Permission Bitmap. It also indicates the maximum CPL value allowing alteration of the IF (INTR Enable Flag) bit when new values are popped into the EFLAG register. POPF and IRET instruction can alter the IOPL field when executed at CPL = 0. Task switches can always alter the IOPL field, when the new flag image is loaded from the incoming task's TSS.

OF (Overflow Flag, bit 11)

OF is set if the operation resulted in a signed overflow. Signed overflow occurs when the operation resulted in carry/borrow **into** the sign bit (high-order bit) of the result but did not result in a carry/borrow **out of** the high-order bit, or vice-versa. For 8/16/32 bit operations, OF is set according to overflow at bit 7/15/31, respectively.

DF (Direction Flag, bit 10)

DF defines whether ESI and/or EDI registers postdecrement or postincrement during the string instructions. Postincrement occurs if DF is reset. Postdecrement occurs if DF is set.

IF (INTR Enable Flag, bit 9)

The IF flag, when set, allows recognition of external interrupts signalled on the INTR pin. When IF is reset, external interrupts signalled on the INTR are not recognized. IOPL indicates the maximum CPL value allowing alteration of the IF bit when new values are popped into EFLAGS or FLAGS.

TF (Trap Enable Flag, bit 8)

TF controls the generation of exception 1 trap when single-stepping through code. When TF is set, the 386 DX generates an exception 1 trap after the next instruction is executed. When TF is reset, exception 1 traps occur only as a function of the breakpoint addresses loaded into debug registers DR0–DR3.

SF (Sign Flag, bit 7)

SF is set if the high-order bit of the result is set, it is reset otherwise. For 8-, 16-, 32-bit operations, SF reflects the state of bit 7, 15, 31 respectively.

ZF (Zero Flag, bit 6)

ZF is set if all bits of the result are 0. Otherwise it is reset.

AF (Auxiliary Carry Flag, bit 4)

The Auxiliary Flag is used to simplify the addition and subtraction of packed BCD quantities. AF is set if the operation resulted in a carry out of bit 3 (addition) or a borrow into bit 3 (subtraction). Otherwise AF is reset. AF is affected by carry out of, or borrow into bit 3 only, regardless of overall operand length: 8, 16 or 32 bits.

PF (Parity Flags, bit 2)

PF is set if the low-order eight bits of the operation contains an even number of "1's" (even parity). PF is reset if the low-order eight bits have odd parity. PF is a function of only the low-order eight bits, regardless of operand size.

CF (Carry Flag, bit 0)

CF is set if the operation resulted in a carry out of (addition), or a borrow into (subtraction) the high-order bit. Otherwise CF is reset. For 8-, 16- or 32-bit operations, CF is set according to carry/borrow at bit 7, 15 or 31, respectively.

Note in these descriptions, "set" means "set to 1," and "reset" means "reset to 0."

2.3.4 Segment Registers

Six 16-bit segment registers hold segment selector values identifying the currently addressable memory segments. Segment registers are shown in Figure 2-4. In Protected Mode, each segment may range in size from one byte up to the entire linear and physi-

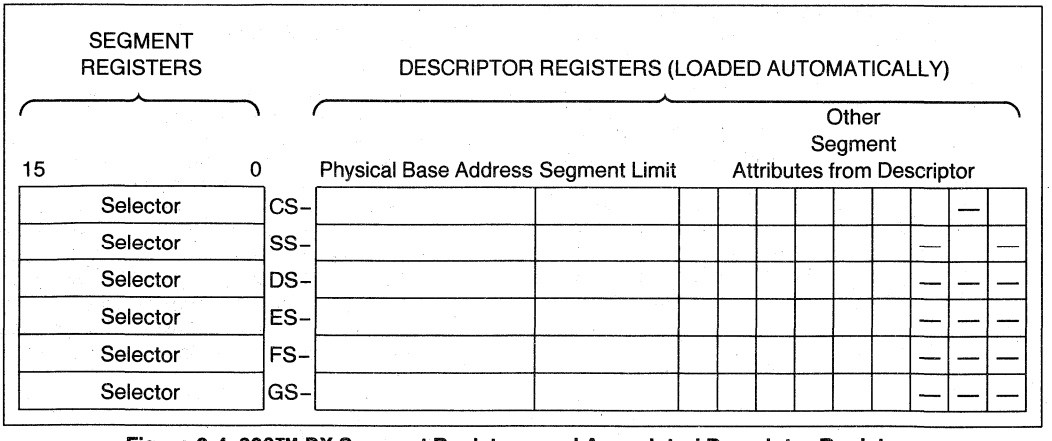


Figure 2-4. 386™ DX Segment Registers, and Associated Descriptor Registers

cal space of the machine, 4 Gbytes (2^{32} bytes). If a maximum sized segment is used (limit = FFFFFFFFH) it should be Dword aligned (i.e., the least two significant bits of the segment base should be zero). This will avoid a segment limit violation (exception 13) caused by the wrap around. In Real Address Mode, the maximum segment size is fixed at 64 Kbytes (2^{16} bytes).

The six segments addressable at any given moment are defined by the segment registers CS, SS, DS, ES, FS and GS. The selector in CS indicates the current code segment; the selector in SS indicates the current stack segment; the selectors in DS, ES, FS and GS indicate the current data segments.

2.3.5 Segment Descriptor Registers

The segment descriptor registers are not programmer visible, yet it is very useful to understand their content. Inside the 386 DX, a descriptor register (programmer invisible) is associated with each programmer-visible segment register, as shown by Figure 2-4. Each descriptor register holds a 32-bit segment base address, a 32-bit segment limit, and the other necessary segment attributes.

When a selector value is loaded into a segment register, the associated descriptor register is automatically updated with the correct information. In Real Address Mode, only the base address is updated directly (by shifting the selector value four bits to the left), since the segment maximum limit and attributes are fixed in Real Mode. In Protected Mode, the base address, the limit, and the attributes are all updated per the contents of the segment descriptor indexed by the selector.

Whenever a memory reference occurs, the segment descriptor register associated with the segment being used is automatically involved with the memory reference. The 32-bit segment base address becomes a component of the linear address calculation, the 32-bit limit is used for the limit-check operation, and the attributes are checked against the type of memory reference requested.

2.3.6 Control Registers

The 386 DX has three control registers of 32 bits, CR0, CR2 and CR3, to hold machine state of a global nature (not specific to an individual task). These registers, along with System Address Registers described in the next section, hold machine state that affects all tasks in the system. To access the Control Registers, load and store instructions are defined.

CR0: Machine Control Register (includes 80286 Machine Status Word)

CR0, shown in Figure 2-5, contains 6 defined bits for control and status purposes. The low-order 16 bits of CR0 are also known as the Machine Status Word, MSW, for compatibility with 80286 Protected Mode. LMSW and SMSW instructions are taken as special aliases of the load and store CR0 operations, where only the low-order 16 bits of CR0 are involved. For compatibility with 80286 operating systems the 386 DX LMSW instructions work in an identical fashion to the LMSW instruction on the 80286. (i.e. It only operates on the low-order 16-bits of CR0 and it ignores the new bits in CR0.) New 386 DX operating systems should use the MOV CR0, Reg instruction.

The defined CR0 bits are described below.

PG (Paging Enable, bit 31)

the PG bit is set to enable the on-chip paging unit. It is reset to disable the on-chip paging unit.

R (reserved, bit 4)

This bit is reserved by Intel. When loading CR0 care should be taken to not alter the value of this bit.

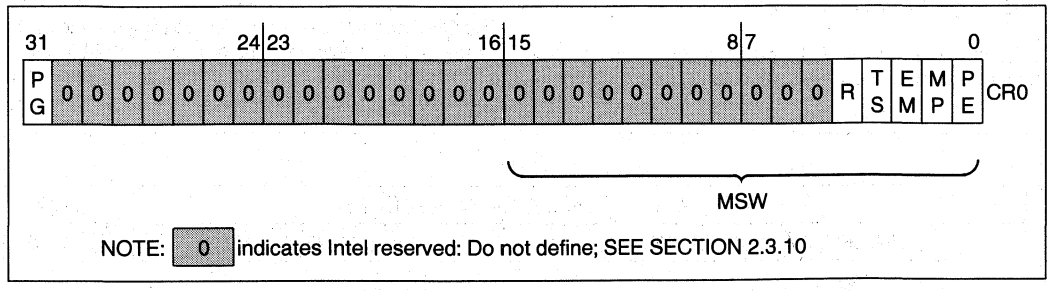


Figure 2-5. Control Register 0

TS (Task Switched, bit 3)

TS is automatically set whenever a task switch operation is performed. If TS is set, a coprocessor ESCape opcode will cause a Coprocessor Not Available trap (exception 7). The trap handler typically saves the 387 DX coprocessor context belonging to a previous task, loads the 387 DX coprocessor state belonging to the current task, and clears the TS bit before returning to the faulting coprocessor opcode.

EM (Emulate Coprocessor, bit 2)

The EMulate coprocessor bit is set to cause all coprocessor opcodes to generate a Coprocessor Not Available fault (exception 7). It is reset to allow coprocessor opcodes to be executed on an actual 387 DX coprocessor (this is the default case after reset). Note that the WAIT opcode is not affected by the EM bit setting.

MP (Monitor Coprocessor, bit 1)

The MP bit is used in conjunction with the TS bit to determine if the WAIT opcode will generate a Coprocessor Not Available fault (exception 7) when TS = 1. When both MP = 1 and TS = 1, the WAIT opcode generates a trap. Otherwise, the WAIT opcode does not generate a trap. Note that TS is automatically set whenever a task switch operation is performed.

PE (Protection Enable, bit 0)

The PE bit is set to enable the Protected Mode. If PE is reset, the processor operates again in Real Mode. PE may be set by loading MSW or CR0. PE can be reset only by a load into CR0. Resetting the PE bit is typically part of a longer instruction sequence needed for proper transition from Protected Mode to Real Mode. Note that for strict 80286 compatibility, PE cannot be reset by the LMSW instruction.

CR1: reserved

CR1 is reserved for use in future Intel processors.

CR2: Page Fault Linear Address

CR2, shown in Figure 2-6, holds the 32-bit linear address that caused the last page fault detected. The error code pushed onto the page fault handler's stack when it is invoked provides additional status information on this page fault.

CR3: Page Directory Base Address

CR3, shown in Figure 2-6, contains the physical base address of the page directory table. The 386 DX page directory table is always page-aligned (4 Kbyte-aligned). Therefore the lowest twelve bits of CR3 are ignored when written and they store as undefined.

A task switch through a TSS which **changes** the value in CR3, or an explicit load into CR3 with any value, will invalidate all cached page table entries in the paging unit cache. Note that if the value in CR3 does not change during the task switch, the cached page table entries are not flushed.

2.3.7 System Address Registers

Four special registers are defined to reference the tables or segments supported by the 80286 CPU and 386 DX protection model. These tables or segments are:

GDT (Global Descriptor Table),

IDT (Interrupt Descriptor Table),

LDT (Local Descriptor Table),

TSS (Task State Segment).

The addresses of these tables and segments are stored in special registers, the System Address and System Segment Registers illustrated in Figure 2-7. These registers are named GDTR, IDTR, LDTR and TR, respectively. Section 4 **Protected Mode Architecture** describes the use of these registers.

GDTR and IDTR

These registers hold the 32-bit linear base address and 16-bit limit of the GDT and IDT, respectively.

The GDT and IDT segments, since they are global to all tasks in the system, are defined by 32-bit linear addresses (subject to page translation if paging is enabled) and 16-bit limit values.

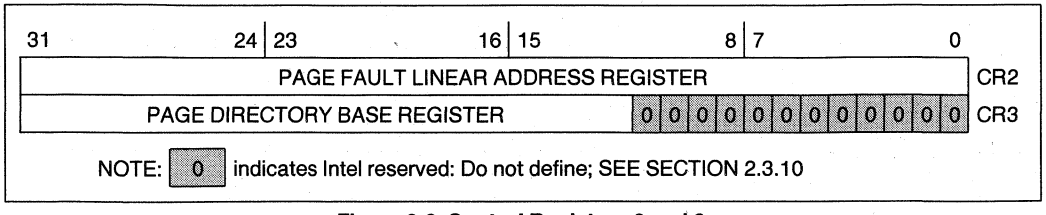


Figure 2-6. Control Registers 2 and 3

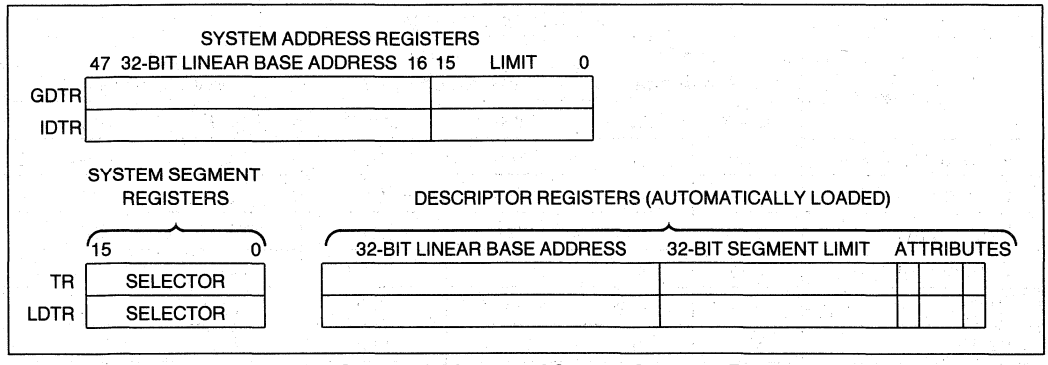


Figure 2-7. System Address and System Segment Registers

LDTR and TR

These registers hold the 16-bit selector for the LDT descriptor and the TSS descriptor, respectively.

The LDT and TSS segments, since they are task-specific segments, are defined by selector values stored in the system segment registers. Note that a segment descriptor register (programmer-invisible) is associated with each system segment register.

2.3.8 Debug and Test Registers

Debug Registers: The six programmer accessible debug registers provide on-chip support for debugging. Debug Registers DR0–3 specify the four linear breakpoints. The Debug Control Register DR7 is used to set the breakpoints and the Debug Status Register DR6, displays the current state of the breakpoints. The use of the debug registers is described in section 2.12 **Debugging support.**

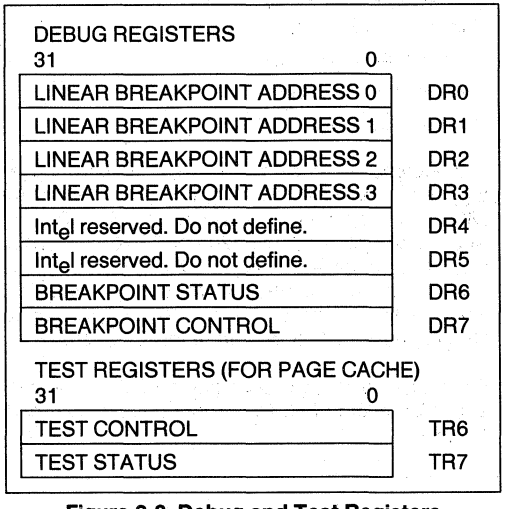


Figure 2-8. Debug and Test Registers

Test Registers: Two registers are used to control the testing of the RAM/CAM (Content Addressable Memories) in the Translation Lookaside Buffer portion of the 386 DX. TR6 is the command test register, and TR7 is the data register which contains the data of the Translation Lookaside buffer test. Their use is discussed in section 2.11 **Testability.**

Figure 2-8 shows the Debug and Test registers.

2.3.9 Register Accessibility

There are a few differences regarding the accessibility of the registers in Real and Protected Mode. Table 2-1 summarizes these differences. See Section 4 **Protected Mode Architecture** for further details.

2.3.10 Compatibility

VERY IMPORTANT NOTE:
COMPATIBILITY WITH FUTURE PROCESSORS

In the preceding register descriptions, note certain 386 DX register bits are Intel reserved. When reserved bits are called out, treat them as fully undefined. This is essential for your software compatibility with future processors! Follow the guidelines below:

1) **Do not depend on the states of any undefined bits when testing the values of defined register bits. Mask them out when testing.**
2) **Do not depend on the states of any undefined bits when storing them to memory or another register.**
3) **Do not depend on the ability to retain information written into any undefined bits.**
4) **When loading registers always load the undefined bits as zeros.**

Table 2-1. Register Usage

Register	Use in Real Mode		Use in Protected Mode		Use in Virtual 8086 Mode	
	Load	Store	Load	Store	Load	Store
General Registers	Yes	Yes	Yes	Yes	Yes	Yes
Segment Registers	Yes	Yes	Yes	Yes	Yes	Yes
Flag Register	Yes	Yes	Yes	Yes	IOPL*	IOPL*
Control Registers	Yes	Yes	PL = 0	PL = 0	No	Yes
GDTR	Yes	Yes	PL = 0	Yes	No	Yes
IDTR	Yes	Yes	PL = 0	Yes	No	Yes
LDTR	No	No	PL = 0	Yes	No	No
TR	No	No	PL = 0	Yes	No	No
Debug Control	Yes	Yes	PL = 0	PL = 0	No	No
Test Registers	Yes	Yes	PL = 0	PL = 0	No	No

NOTES:
PL = 0: The registers can be accessed only when the current privilege level is zero.
*IOPL: The PUSHF and POPF instructions are made I/O Privilege Level sensitive in Virtual 8086 Mode.

5) However, registers which have been previously stored may be reloaded without masking.

Depending upon the values of undefined register bits will make your software dependent upon the unspecified 386 DX handling of these bits. Depending on undefined values risks making your software incompatible with future processors that define usages for the 386 DX-undefined bits. AVOID ANY SOFTWARE DEPENDENCE UPON THE STATE OF UNDEFINED 386 DX REGISTER BITS.

2.4 INSTRUCTION SET

2.4.1 Instruction Set Overview

The instruction set is divided into nine categories of operations:

Data Transfer
Arithmetic
Shift/Rotate
String Manipulation
Bit Manipulation
Control Transfer
High Level Language Support
Operating System Support
Processor Control

These 386 DX instructions are listed in Table 2-2.

All 386 DX instructions operate on either 0, 1, 2, or 3 operands; where an operand resides in a register, in the instruction itself, or in memory. Most zero operand instructions (e.g. CLI, STI) take only one byte. One operand instructions generally are two bytes long. The average instruction is 3.2 bytes long. Since the 386 DX has a 16-byte instruction queue, an average of 5 instructions will be prefetched. The use of two operands permits the following types of common instructions:

Register to Register
Memory to Register
Immediate to Register
Register to Memory
Immediate to Memory.

The operands can be either 8, 16, or 32 bits long. As a general rule, when executing code written for the 386 DX (32-bit code), operands are 8 or 32 bits; when executing existing 80286 or 8086 code (16-bit code), operands are 8 or 16 bits. Prefixes can be added to all instructions which override the default length of the operands, (i.e. use 32-bit operands for 16-bit code, or 16-bit operands for 32-bit code).

For a more elaborate description of the instruction set, refer to the "386 DX Programmer's Reference Manual."

2.4.2 386™ DX Instructions

Table 2-2a. Data Transfer

GENERAL PURPOSE	
MOV	Move operand
PUSH	Push operand onto stack
POP	Pop operand off stack
PUSHA	Push all registers on stack
POPA	Pop all registers off stack
XCHG	Exchange Operand, Register
XLAT	Translate
CONVERSION	
MOVZX	Move byte or Word, Dword, with zero extension
MOVSX	Move byte or Word, Dword, sign extended
CBW	Convert byte to Word, or Word to Dword
CWD	Convert Word to DWORD
CWDE	Convert Word to DWORD extended
CDQ	Convert DWORD to QWORD
INPUT/OUTPUT	
IN	Input operand from I/O space
OUT	Output operand to I/O space
ADDRESS OBJECT	
LEA	Load effective address
LDS	Load pointer into D segment register
LES	Load pointer into E segment register
LFS	Load pointer into F segment register
LGS	Load pointer into G segment register
LSS	Load pointer into S (Stack) segment register
FLAG MANIPULATION	
LAHF	Load A register from Flags
SAHF	Store A register in Flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack
PUSHFD	Push EFlags onto stack
POPFD	Pop EFlags off stack
CLC	Clear Carry Flag
CLD	Clear Direction Flag
CMC	Complement Carry Flag
STC	Set Carry Flag
STD	Set Direction Flag

Table 2-2b. Arithmetic Instructions

ADDITION	
ADD	Add operands
ADC	Add with carry
INC	Increment operand by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
SUBTRACTION	
SUB	Subtract operands
SBB	Subtract with borrow
DEC	Decrement operand by 1
NEG	Negate operand
CMP	Compare operands
DAS	Decimal adjust for subtraction
AAS	ASCII Adjust for subtraction
MULTIPLICATION	
MUL	Multiply Double/Single Precision
IMUL	Integer multiply
AAM	ASCII adjust after multiply
DIVISION	
DIV	Divide unsigned
IDIV	Integer Divide
AAD	ASCII adjust before division

Table 2-2c. String Instructions

MOVS	Move byte or Word, Dword string
INS	Input string from I/O space
OUTS	Output string to I/O space
CMPS	Compare byte or Word, Dword string
SCAS	Scan Byte or Word, Dword string
LODS	Load byte or Word, Dword string
STOS	Store byte or Word, Dword string
REP	Repeat
REPE/ REPZ	Repeat while equal/zero
RENE/ REPNZ	Repeat while not equal/not zero

Table 2-2d. Logical Instructions

LOGICALS	
NOT	"NOT" operands
AND	"AND" operands
OR	"Inclusive OR" operands
XOR	"Exclusive OR" operands
TEST	"Test" operands

5

Table 2-2d. Logical Instructions (Continued)

SHIFTS	
SHL/SHR	Shift logical left or right
SAL/SAR	Shift arithmetic left or right
SHLD/ SHRD	Double shift left or right
ROTATES	
ROL/ROR	Rotate left/right
RCL/RCR	Rotate through carry left/right

Table 2-2e. Bit Manipulation Instructions

SINGLE BIT INSTRUCTIONS	
BT	Bit Test
BTS	Bit Test and Set
BTR	Bit Test and Reset
BTC	Bit Test and Complement
BSF	Bit Scan Forward
BSR	Bit Scan Reverse

Table 2-2f. Program Control Instructions

CONDITIONAL TRANSFERS	
SETCC	Set byte equal to condition code
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign
JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if Sign

Table 2-2f. Program Control Instructions (Continued)

UNCONDITIONAL TRANSFERS	
CALL	Call procedure/task
RET	Return from procedure
JMP	Jump
ITERATION CONTROLS	
LOOP	Loop
LOOPE/ LOOPZ	Loop if equal/zero
LOOPNE/ LOOPNZ	Loop if not equal/not zero
JCXZ	JUMP if register CX = 0
INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Return from Interrupt/Task
CLI	Clear interrupt Enable
STI	Set Interrupt Enable

Table 2-2g. High Level Language Instructions

BOUND	Check Array Bounds
ENTER	Setup Parameter Block for Entering Procedure
LEAVE	Leave Procedure

Table 2-2h. Protection Model

SGDT	Store Global Descriptor Table
SIDT	Store Interrupt Descriptor Table
STR	Store Task Register
SLDT	Store Local Descriptor Table
LGDT	Load Global Descriptor Table
LIDT	Load Interrupt Descriptor Table
LTR	Load Task Register
LLDT	Load Local Descriptor Table
ARPL	Adjust Requested Privilege Level
LAR	Load Access Rights
LSL	Load Segment Limit
VERR/ VERW	Verify Segment for Reading or Writing
LMSW	Load Machine Status Word (lower 16 bits of CR0)
SMSW	Store Machine Status Word

Table 2-2i. Processor Control Instructions

HLT	Halt
WAIT	Wait until BUSY# negated
ESC	Escape
LOCK	Lock Bus

2.5 ADDRESSING MODES

2.5.1 Addressing Modes Overview

The 386 DX provides a total of 11 addressing modes for instructions to specify operands. The addressing modes are optimized to allow the efficient execution of high level languages such as C and FORTRAN, and they cover the vast majority of data references needed by high-level languages.

2.5.2 Register and Immediate Modes

Two of the addressing modes provide for instructions that operate on register or immediate operands:

Register Operand Mode: The operand is located in one of the 8-, 16- or 32-bit general registers.

Immediate Operand Mode: The operand is included in the instruction as part of the opcode.

2.5.3 32-Bit Memory Addressing Modes

The remaining 9 modes provide a mechanism for specifying the effective address of an operand. The linear address consists of two components: the segment base address and an effective address. The effective address is calculated by using combinations of the following four address elements:

DISPLACEMENT: An 8-, or 32-bit immediate value, following the instruction.

BASE: The contents of any general purpose register. The base registers are generally used by compilers to point to the start of the local variable area.

INDEX: The contents of any general purpose register except for ESP. The index registers are used to access the elements of an array, or a string of characters.

SCALE: The index register's value can be multiplied by a scale factor, either 1, 2, 4 or 8. Scaled index mode is especially useful for accessing arrays or structures.

Combinations of these 4 components make up the 9 additional addressing modes. There is no performance penalty for using any of these addressing combinations, since the effective address calculation is pipelined with the execution of other instructions. The one exception is the simultaneous use of Base and Index components which requires one additional clock.

As shown in Figure 2-9, the effective address (EA) of an operand is calculated according to the following formula.

EA = Base Reg + (Index Reg * Scaling) + Displacement

Direct Mode: The operand's offset is contained as part of the instruction as an 8-, 16- or 32-bit displacement.
EXAMPLE: INC Word PTR [500]

Register Indirect Mode: A BASE register contains the address of the operand.
EXAMPLE: MOV [ECX], EDX

Based Mode: A BASE register's contents is added to a DISPLACEMENT to form the operands offset.
EXAMPLE: MOV ECX, [EAX + 24]

Index Mode: An INDEX register's contents is added to a DISPLACEMENT to form the operands offset.
EXAMPLE: ADD EAX, TABLE[ESI]

Scaled Index Mode: An INDEX register's contents is multiplied by a scaling factor which is added to a DISPLACEMENT to form the operands offset.
EXAMPLE: IMUL EBX, TABLE[ESI*4],7

Based Index Mode: The contents of a BASE register is added to the contents of an INDEX register to form the effective address of an operand.
EXAMPLE: MOV EAX, [ESI] [EBX]

Based Scaled Index Mode: The contents of an INDEX register is multiplied by a SCALING factor and the result is added to the contents of a BASE register to obtain the operands offset.
EXAMPLE: MOV ECX, [EDX*8] [EAX]

Based Index Mode with Displacement: The contents of an INDEX Register and a BASE register's contents and a DISPLACEMENT are all summed together to form the operand offset.
EXAMPLE: ADD EDX, [ESI] [EBP + 00FFFFF0H]

Based Scaled Index Mode with Displacement: The contents of an INDEX register are multiplied by a SCALING factor, the result is added to the contents of a BASE register and a DISPLACEMENT to form the operand's offset.
EXAMPLE: MOV EAX, LOCALTABLE[EDI*4] [EBP + 80]

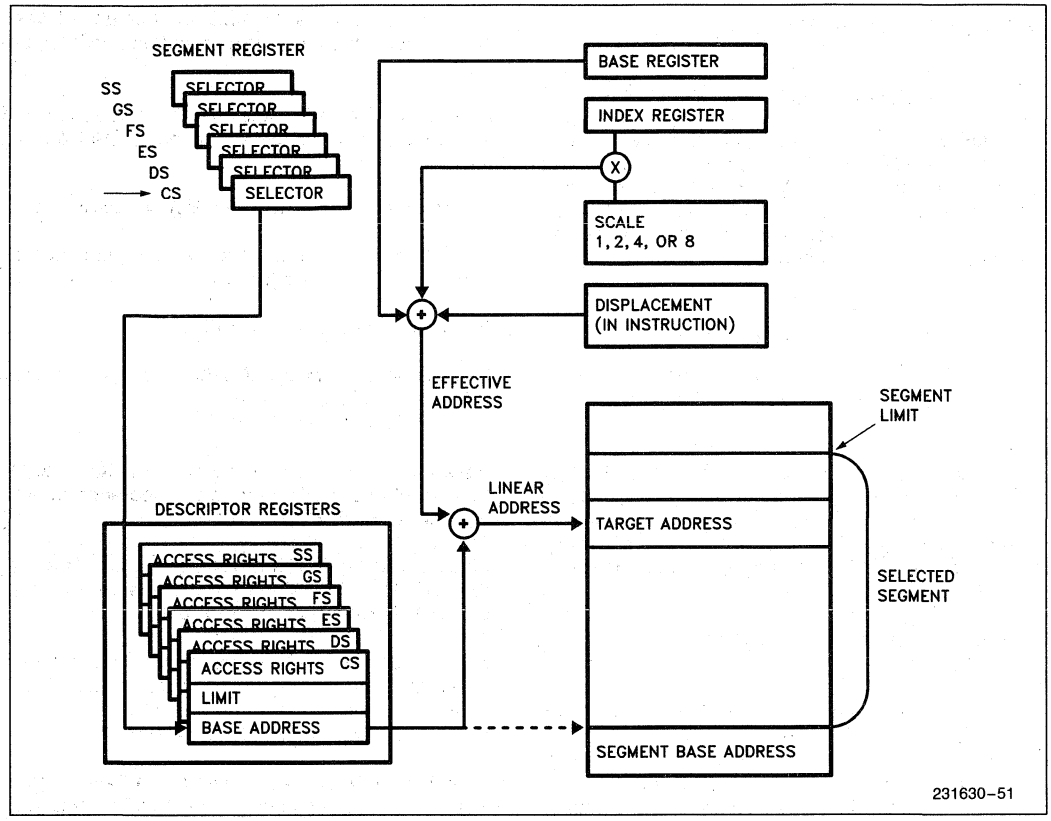


Figure 2-9. Addressing Mode Calculations

2.5.4 Differences Between 16 and 32 Bit Addresses

In order to provide software compatibility with the 80286 and the 8086, the 386 DX can execute 16-bit instructions in Real and Protected Modes. The processor determines the size of the instructions it is executing by examining the D bit in the CS segment Descriptor. If the D bit is 0 then all operand lengths and effective addresses are assumed to be 16 bits long. If the D bit is 1 then the default length for operands and addresses is 32 bits. In Real Mode the default size for operands and addresses is 16-bits.

Regardless of the default precision of the operands or addresses, the 386 DX is able to execute either 16 or 32-bit instructions. This is specified via the use of override prefixes. Two prefixes, the **Operand Size Prefix** and the **Address Length Prefix**, override the value of the D bit on an individual instruction basis. These prefixes are automatically added by Intel assemblers.

Example: The processor is executing in Real Mode and the programmer needs to access the EAX registers. The assembler code for this might be MOV EAX, 32-bit MEMORYOP, ASM386 Macro Assembler automatically determines that an Operand Size Prefix is needed and generates it.

Example: The D bit is 0, and the programmer wishes to use Scaled Index addressing mode to access an array. The Address Length Prefix allows the use of MOV DX, TABLE[ESI*2]. The assembler uses an Address Length Prefix since, with D=0, the default addressing mode is 16-bits.

Example: The D bit is 1, and the program wants to store a 16-bit quantity. The Operand Length Prefix is used to specify only a 16-bit value; MOV MEM16, DX.

Table 2-3. BASE and INDEX Registers for 16- and 32-Bit Addresses

	16-Bit Addressing	32-Bit Addressing
BASE REGISTER	BX,BP	Any 32-bit GP Register
INDEX REGISTER	SI,DI	Any 32-bit GP Register Except ESP
SCALE FACTOR	none	1, 2, 4, 8
DISPLACEMENT	0, 8, 16 bits	0, 8, 32 bits

The OPERAND LENGTH and Address Length Prefixes can be applied separately or in combination to any instruction. The Address Length Prefix does not allow addresses over 64K bytes to be accessed in Real Mode. A memory address which exceeds FFFFH will result in a General Protection Fault. An Address Length Prefix only allows the use of the additional 386 DX addressing modes.

When executing 32-bit code, the 386 DX uses either 8-, or 32-bit displacements, and any register can be used as base or index registers. When executing 16-bit code, the displacements are either 8, or 16 bits, and the base and index register conform to the 80286 model. Table 2-3 illustrates the differences.

2.6 DATA TYPES

The 386 DX supports all of the data types commonly used in high level languages:

Bit: A single bit quantity.

Bit Field: A group of up to 32 contiguous bits, which spans a maximum of four bytes.

Bit String: A set of contiguous bits, on the 386 DX bit strings can be up to 4 gigabits long.

Byte: A signed 8-bit quantity.

Unsigned Byte: An unsigned 8-bit quantity.

Integer (Word): A signed 16-bit quantity.

Long Integer (Double Word): A signed 32-bit quantity. All operations assume a 2's complement representation.

Unsigned Integer (Word): An unsigned 16-bit quantity.

Unsigned Long Integer (Double Word): An unsigned 32-bit quantity.

Signed Quad Word: A signed 64-bit quantity.

Unsigned Quad Word: An unsigned 64-bit quantity.

Offset: A 16- or 32-bit offset only quantity which indirectly references another memory location.

Pointer: A full pointer which consists of a 16-bit segment selector and either a 16- or 32-bit offset.

Char: A byte representation of an ASCII Alphanumeric or control character.

String: A contiguous sequence of bytes, words or dwords. A string may contain between 1 byte and 4 Gbytes.

BCD: A byte (unpacked) representation of decimal digits 0–9.

Packed BCD: A byte (packed) representation of two decimal digits 0–9 storing one digit in each nibble.

When the 386 DX is coupled with a 387 DX Numerics Coprocessor then the following common Floating Point types are supported.

Floating Point: A signed 32-, 64-, or 80-bit real number representation. Floating point numbers are supported by the 387 DX numerics coprocessor.

Figure 2-10 illustrates the data types supported by the 386 DX and the 387 DX numerics coprocessor.

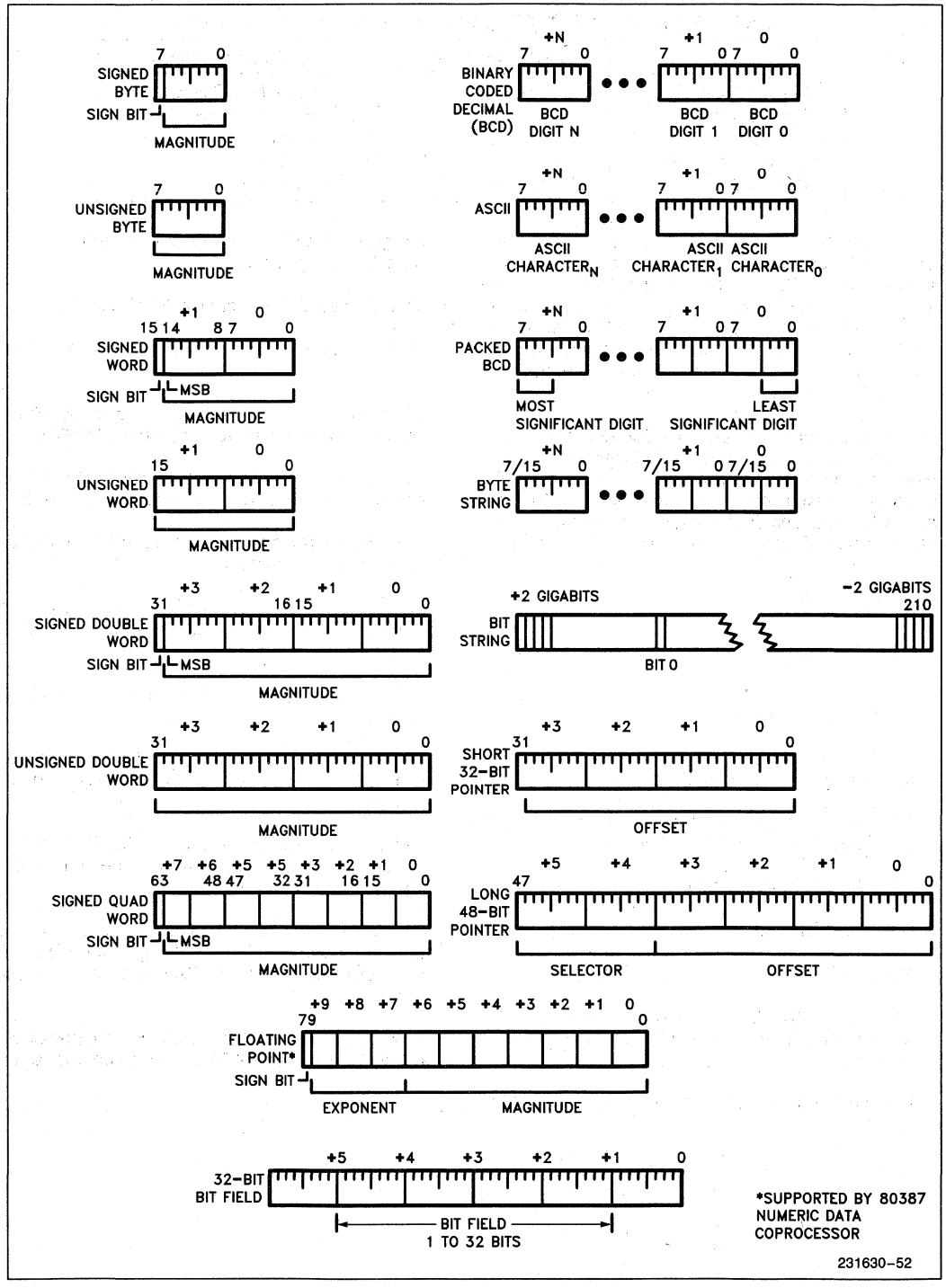


Figure 2-10. 386™ DX Supported Data Types

2.7 MEMORY ORGANIZATION

2.7.1 Introduction

Memory on the 386 DX is divided up into 8-bit quantities (bytes), 16-bit quantities (words), and 32-bit quantities (dwords). Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address, the high order byte at the high address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address, the high-order byte at the highest address. The address of a word or dword is the byte address of the low-order byte.

In addition to these basic data types, the 386 DX supports two larger units of memory: pages and segments. Memory can be divided up into one or more variable length segments, which can be swapped to disk or shared between programs. Memory can also be organized into one or more 4K byte pages. Finally, both segmentation and paging can be combined, gaining the advantages of both systems. The 386 DX supports both pages and segments in order to provide maximum flexibility to the system designer. Segmentation and paging are complementary. Segmentation is useful for organizing memory in logical modules, and as such is a tool for the application programmer, while pages are useful for the system programmer for managing the physical memory of a system.

2.7.2 Address Spaces

The 386 DX has three distinct address spaces: **logical, linear,** and **physical**. A **logical** address (also known as a **virtual** address) consists of a selector and an offset. A selector is the contents of a segment register. An offset is formed by summing all of the addressing components (BASE, INDEX, DISPLACEMENT) discussed in section 2.5.3 **Memory Addressing Modes** into an effective address. Since each task on 386 DX has a maximum of 16K ($2^{14}-1$) selectors, and offsets can be 4 gigabytes, (2^{32} bits) this gives a total of 2^{46} bits or 64 terabytes of **logical** address space per task. The programmer sees this virtual address space.

The segmentation unit translates the **logical** address space into a 32-bit **linear** address space. If the paging unit is not enabled then the 32-bit **linear** address corresponds to the **physical** address. The paging unit translates the **linear** address space into the **physical** address space. The **physical address** is what appears on the address pins.

The primary difference between Real Mode and Protected Mode is how the segmentation unit performs the translation of the **logical** address into the **linear** address. In Real Mode, the segmentation unit shifts the selector left four bits and adds the result to the offset to form the **linear** address. While in Protected Mode every selector has a **linear** base address associated with it. The **linear base** address is stored in one of two operating system tables (i.e. the Local Descriptor Table or Global Descriptor Table). The selector's **linear base** address is added to the offset to form the final **linear** address.

Figure 2-11 shows the relationship between the various address spaces.

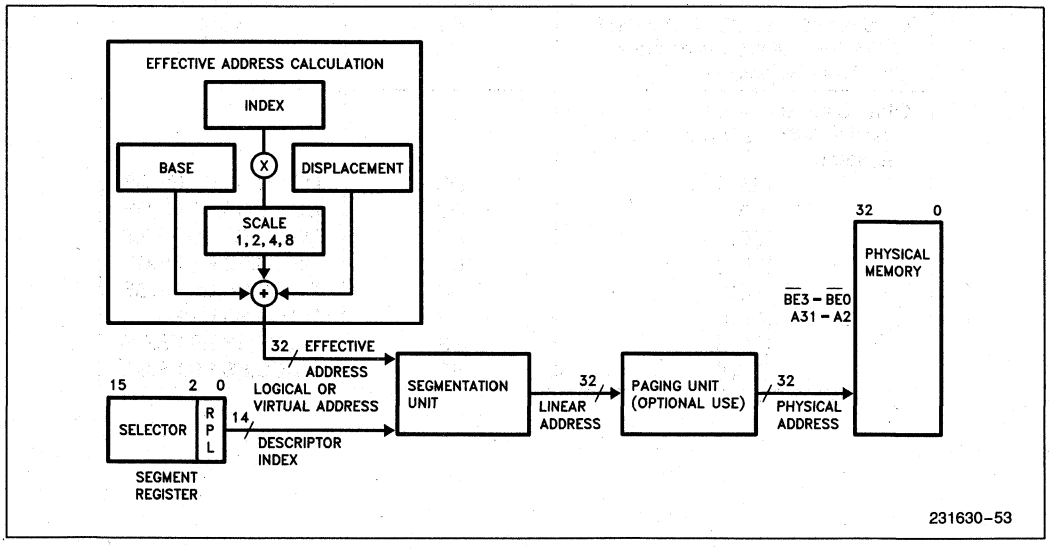


Figure 2-11. Address Translation

2.7.3 Segment Register Usage

The main data structure used to organize memory is the segment. On the 386 DX, segments are variable sized blocks of linear addresses which have certain attributes associated with them. There are two main types of segments: code and data, the segments are of variable size and can be as small as 1 byte or as large as 4 gigabytes (2^{32} bytes).

In order to provide compact instruction encoding, and increase processor performance, instructions do not need to explicitly specify which segment register is used. A default segment register is automatically chosen according to the rules of Table 2-4 (Segment Register Selection Rules). In general, data references use the selector contained in the DS register; Stack references use the SS register and Instruction fetches use the CS register. The contents of the Instruction Pointer provides the offset. Special segment override prefixes allow the explicit use of a given segment register, and override the implicit rules listed in Table 2-4. The override prefixes also allow the use of the ES, FS and GS segment registers.

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all 6 segments could have the base address set to zero and create a system with a four gigabyte linear address space. This creates a system where the virtual address space is the same as the linear address space. Further details of segmentation are discussed in section 4.1.

2.8 I/O SPACE

The 386 DX has two distinct physical address spaces: Memory and I/O. Generally, peripherals are placed in I/O space although the 386 DX also supports memory-mapped peripherals. The I/O space consists of 64K bytes, it can be divided into 64K 8-bit ports, 32K 16-bit ports, or 16K 32-bit ports, or any combination of ports which add up to less than 64K bytes. The 64K I/O address space refers to physical memory rather than linear address since I/O instructions do not go through the segmentation or paging hardware. The M/IO# pin acts as an additional address line thus allowing the system designer to easily determine which address space the processor is accessing.

Table 2-4. Segment Register Selection Rules

Type of Memory Reference	Implied (Default) Segment Use	Segment Override Prefixes Possible
Code Fetch	CS	None
Destination of PUSH, PUSHF, INT, CALL, PUSHA Instructions	SS	None
Source of POP, POPA, POPF, IRET, RET instructions	SS	None
Destination of STOS, MOVS, REP STOS, REP MOVS Instructions (DI is Base Register)	ES	None
Other Data References, with Effective Address Using Base Register of:		
[EAX]	DS	DS,CS,SS,ES,FS,GS
[EBX]	DS	DS,CS,SS,ES,FS,GS
[ECX]	DS	DS,CS,SS,ES,FS,GS
[EDX]	DS	DS,CS,SS,ES,FS,GS
[ESI]	DS	DS,CS,SS,ES,FS,GS
[EDI]	DS	DS,CS,SS,ES,FS,GS
[EBP]	SS	DS,CS,SS,ES,FS,GS
[ESP]	SS	DS,CS,SS,ES,FS,GS

The I/O ports are accessed via the IN and OUT I/O instructions, with the port address supplied as an immediate 8-bit constant in the instruction or in the DX register. All 8- and 16-bit port addresses are zero extended on the upper address lines. The I/O instructions cause the M/IO# pin to be driven low.

I/O port addresses 00F8H through 00FFH are reserved for use by Intel.

2.9 INTERRUPTS

2.9.1 Interrupts and Exceptions

Interrupts and exceptions alter the normal program flow, in order to handle external events, to report errors or exceptional conditions. The difference between interrupts and exceptions is that interrupts are used to handle asynchronous external events while exceptions handle instruction faults. Although a program can generate a software interrupt via an INT N instruction, the processor treats software interrupts as exceptions.

Hardware interrupts occur as the result of an external event and are classified into two types: maskable or non-maskable. Interrupts are serviced after the execution of the current instruction. After the interrupt handler is finished servicing the interrupt, execution proceeds with the instruction immediately **after** the interrupted instruction. Sections 2.9.3 and 2.9.4 discuss the differences between Maskable and Non-Maskable interrupts.

Exceptions are classified as faults, traps, or aborts depending on the way they are reported, and whether or not restart of the instruction causing the exception is supported. **Faults** are exceptions that are detected and serviced **before** the execution of the faulting instruction. A fault would occur in a virtual memory system, when the processor referenced a page or a segment which was not present. The operating system would fetch the page or segment from disk, and then the 386 DX would restart the instruction. **Traps** are exceptions that are reported immediately **after** the execution of the instruction which caused the problem. User defined interrupts are examples of traps. **Aborts** are exceptions which do not permit the precise location of the instruction causing the exception to be determined. Aborts are used to report severe errors, such as a hardware error, or illegal values in system tables.

Thus, when an interrupt service routine has been completed, execution proceeds from the instruction immediately following the interrupted instruction. On the other hand, the return address from an exception fault routine will always point at the instruction causing the exception and include any leading instruction prefixes. Table 2-5 summarizes the possible interrupts for the 386 DX and shows where the return address points.

The 386 DX has the ability to handle up to 256 different interrupts/exceptions. In order to service the interrupts, a table with up to 256 interrupt vectors must be defined. The interrupt vectors are simply pointers to the appropriate interrupt service routine. In Real Mode (see section 3.1), the vectors are 4 byte quantities, a Code Segment plus a 16-bit offset; in Protected Mode, the interrupt vectors are 8 byte quantities, which are put in an Interrupt Descriptor Table (see section 4.1). Of the 256 possible interrupts, 32 are reserved for use by Intel, the remaining 224 are free to be used by the system designer.

2.9.2 Interrupt Processing

When an interrupt occurs the following actions happen. First, the current program address and the Flags are saved on the stack to allow resumption of the interrupted program. Next, an 8-bit vector is supplied to the 386 DX which identifies the appropriate entry in the interrupt table. The table contains the starting address of the interrupt service routine. Then, the user supplied interrupt service routine is executed. Finally, when an IRET instruction is executed the old processor state is restored and program execution resumes at the appropriate instruction.

The 8-bit interrupt vector is supplied to the 386 DX in several different ways: exceptions supply the interrupt vector internally; software INT instructions contain or imply the vector; maskable hardware interrupts supply the 8-bit vector via the interrupt acknowledge bus sequence. Non-Maskable hardware interrupts are assigned to interrupt vector 2.

2.9.3 Maskable Interrupt

Maskable interrupts are the most common way used by the 386 DX to respond to asynchronous external hardware events. A hardware interrupt occurs when the INTR is pulled high and the Interrupt Flag bit (IF) is enabled. The processor only responds to interrupts between instructions, (REPeat String instructions, have an "interrupt window", between memory moves, which allows interrupts during long

Table 2-5. Interrupt Vector Assignments

Function	Interrupt Number	Instruction Which Can Cause Exception	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	YES	FAULT
Debug Exception	1	any instruction	YES	TRAP*
NMI Interrupt	2	INT 2 or NMI	NO	NMI
One Byte Interrupt	3	INT	NO	TRAP
Interrupt on Overflow	4	INTO	NO	TRAP
Array Bounds Check	5	BOUND	YES	FAULT
Invalid OP-Code	6	Any Illegal Instruction	YES	FAULT
Device Not Available	7	ESC, WAIT	YES	FAULT
Double Fault	8	Any Instruction That Can Generate an Exception		ABORT
Coprocessor Segment Overrun	9	ESC	NO	ABORT
Invalid TSS	10	JMP, CALL, IRET, INT	YES	FAULT
Segment Not Present	11	Segment Register Instructions	YES	FAULT
Stack Fault	12	Stack References	YES	FAULT
General Protection Fault	13	Any Memory Reference	YES	FAULT
Intel Reserved	15			
Page Fault	14	Any Memory Access or Code Fetch	YES	FAULT
Coprocessor Error	16	ESC, WAIT	YES	FAULT
Intel Reserved	17–31			
Two Byte Interrupt	0–255	INT n	NO	TRAP

* Some debug exceptions may report both traps on the previous instruction, and faults on the next instruction.

string moves). When an interrupt occurs the processor reads an 8-bit vector supplied by the hardware which identifies the source of the interrupt, (one of 224 user defined interrupts). The exact nature of the interrupt sequence is discussed in section 5.

The IF bit in the EFLAG registers is reset when an interrupt is being serviced. This effectively disables servicing additional interrupts during an interrupt service routine. However, the IF may be set explicitly by the interrupt handler, to allow the nesting of interrupts. When an IRET instruction is executed the original state of the IF is restored.

2.9.4 Non-Maskable Interrupt

Non-maskable interrupts provide a method of servicing very high priority interrupts. A common example of the use of a non-maskable interrupt (NMI) would be to activate a power failure routine. When the NMI input is pulled high it causes an interrupt with an internally supplied vector value of 2. Unlike a normal hardware interrupt, no interrupt acknowledgment sequence is performed for an NMI.

While executing the NMI servicing procedure, the 386 DX will not service further NMI requests, until an interrupt return (IRET) instruction is executed or the processor is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. The IF bit is cleared at the beginning of an NMI interrupt to inhibit further INTR interrupts.

2.9.5 Software Interrupts

A third type of interrupt/exception for the 386 DX is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the nth vector in the interrupt table.

A special case of the two byte software interrupt INT n is the one byte INT 3, or breakpoint interrupt. By inserting this one byte instruction in a program, the user can set breakpoints in his program as a debugging tool.

A final type of software interrupt, is the single step interrupt. It is discussed in section 2.12.

2.9.6 Interrupt and Exception Priorities

Interrupts are externally-generated events. Maskable Interrupts (on the INTR input) and Non-Maskable Interrupts (on the NMI input) are recognized at instruction boundaries. When NMI and maskable INTR are **both** recognized at the **same** instruction boundary, the 386 DX invokes the NMI service routine first. If, after the NMI service routine has been invoked, maskable interrupts are still enabled, then the 386 DX will invoke the appropriate interrupt service routine.

Table 2-6a. 386™ DX Priority for Invoking Service Routines in Case of Simultaneous External Interrupts

1. NMI
2. INTR

Exceptions are internally-generated events. Exceptions are detected by the 386 DX if, in the course of executing an instruction, the 386 DX detects a problematic condition. The 386 DX then immediately invokes the appropriate exception service routine. The state of the 386 DX is such that the instruction causing the exception can be restarted. If the exception service routine has taken care of the problematic condition, the instruction will execute without causing the same exception.

It is possible for a single instruction to generate several exceptions (for example, transferring a single operand could generate two page faults if the operand location spans two "not present" pages). However, only one exception is generated upon each attempt to execute the instruction. Each exception service routine should correct its corresponding exception, and restart the instruction. In this manner, exceptions are serviced until the instruction executes successfully.

As the 386 DX executes instructions, it follows a consistent cycle in checking for exceptions, as shown in Table 2-6b. This cycle is repeated as each instruction is executed, and occurs in parallel with instruction decoding and execution.

Table 2-6b. Sequence of Exception Checking

Consider the case of the 386 DX having just completed an instruction. It then performs the following checks before reaching the point where the next instruction is completed:

1. Check for Exception 1 Traps from the instruction just completed (single-step via Trap Flag, or Data Breakpoints set in the Debug Registers).
2. Check for Exception 1 Faults in the next instruction (Instruction Execution Breakpoint set in the Debug Registers for the next instruction).
3. Check for external NMI and INTR.
4. Check for Segmentation Faults that prevented fetching the entire next instruction (exceptions 11 or 13).
5. Check for Page Faults that prevented fetching the entire next instruction (exception 14).
6. Check for Faults decoding the next instruction (exception 6 if illegal opcode; exception 6 if in Real Mode or in Virtual 8086 Mode and attempting to execute an instruction for Protected Mode only (see 4.6.4); or exception 13 if instruction is longer than 15 bytes, or privilege violation in Protected Mode (i.e. not at IOPL or at CPL = 0).
7. If WAIT opcode, check if TS = 1 and MP = 1 (exception 7 if both are 1).
8. If ESCAPE opcode for numeric coprocessor, check if EM = 1 or TS = 1 (exception 7 if either are 1).
9. If WAIT opcode or ESCAPE opcode for numeric coprocessor, check ERROR# input signal (exception 16 if ERROR# input is asserted).
10. Check in the following order for each memory reference required by the instruction:
    a. Check for Segmentation Faults that prevent transferring the entire memory quantity (exceptions 11, 12, 13).
    b. Check for Page Faults that prevent transferring the entire memory quantity (exception 14).

Note that the order stated supports the concept of the paging mechanism being "underneath" the segmentation mechanism. Therefore, for any given code or data reference in memory, segmentation exceptions are generated before paging exceptions are generated.

2.9.7 Instruction Restart

The 386 DX fully supports restarting all instructions after faults. If an exception is detected in the instruction to be executed (exception categories 4 through 10 in Table 2-6b), the 386 DX invokes the appropriate exception service routine. The 386 DX is in a state that permits restart of the instruction, for all cases but those in Table 2-6c. Note that all such cases are easily avoided by proper design of the operating system.

Table 2-6c. Conditions Preventing Instruction Restart

A. An instruction causes a task switch to a task whose Task State Segment is **partially** "not present". (An entirely "not present" TSS is restartable.) Partially present TSS's can be avoided either by keeping the TSS's of such tasks present in memory, or by aligning TSS segments to reside entirely within a single 4K page (for TSS segments of 4K bytes or less).
B. A coprocessor operand wraps around the top of a 64K-byte segment or a 4G-byte segment, and spans three pages, and the page holding the middle portion of the operand is "not present." This condition can be avoided by starting **at a page boundary** any segments containing coprocessor operands if the segments are approximately 64K-200 bytes or larger (i.e. large enough for wraparound of the coprocessor operand to possibly occur).
Note that these conditions are avoided by using the operating system designs mentioned in this table.

2.9.8 Double Fault

A Double Fault (exception 8) results when the processor attempts to invoke an exception service routine for the segment exceptions (10, 11, 12 or 13), but in the process of doing so, detects an exception **other than** a Page Fault (exception 14).

A Double Fault (exception 8) will also be generated when the processor attempts to invoke the Page Fault (exception 14) service routine, and detects an exception other than a second Page Fault. In any functional system, the entire Page Fault service routine must remain "present" in memory.

Double page faults however do not raise the double fault exception. If a second page fault occurs while the processor is attempting to enter the service routine for the first time, then the processor will invoke the page fault (exception 14) handler a second time, rather than the double fault (exception 8) handler. A subsequent fault, though, will lead to shutdown.

When a Double Fault occurs, the 386 DX invokes the exception service routine for exception 8.

2.10 RESET AND INITIALIZATION

When the processor is initialized or Reset the registers have the values shown in Table 2-7. The 386 DX will then start executing instructions near the top of physical memory, at location FFFFFFF0H. When the first InterSegment Jump or Call is executed, address lines A20-31 will drop low for CS-relative memory cycles, and the 386 DX will only execute instructions in the lower one megabyte of physical memory. This allows the system designer to use a ROM at the top of physical memory to initialize the system and take care of Resets.

RESET forces the 386 DX to terminate all execution and local bus activity. No instruction execution or bus activity will occur as long as Reset is active. Between 350 and 450 CLK2 periods after Reset becomes inactive the 386 DX will start executing instructions at the top of physical memory.

Table 2-7. Register Values after Reset

Flag Word	UUUU0002H	Note 1
Machine Status Word (CR0)	UUUUUUU0H	Note 2
Instruction Pointer	0000FFF0H	
Code Segment	F000H	Note 3
Data Segment	0000H	
Stack Segment	0000H	
Extra Segment (ES)	0000H	
Extra Segment (FS)	0000H	
Extra Segment (GS)	0000H	
DX register	component and stepping ID	Note 5
All other registers	undefined	Note 4

NOTES:
1. EFLAG Register. The upper 14 bits of the EFLAGS register are undefined, VM (Bit 17) and RF (BIT) 16 are 0 as are all other defined flag bits.
2. CR0: (Machine Status Word). All of the defined fields in the CR0 are 0 (PG Bit 31, TS Bit 3, EM Bit 2, MP Bit 1, and PE Bit 0).
3. The Code Segment Register (CS) will have its Base Address set to FFFF0000H and Limit set to 0FFFFH.
4. All undefined bits are Intel Reserved and should not be used.
5. DX register always holds component and stepping identifier (see 5.7). EAX register holds self-test signature if self-test was requested (see 5.6).

2.11 TESTABILITY

2.11.1 Self-Test

The 386 DX has the capability to perform a self-test. The self-test checks the function of all of the Control ROM and most of the non-random logic of the part. Approximately one-half of the 386 DX can be tested during self-test.

Self-Test is initiated on the 386 DX when the RESET pin transitions from HIGH to LOW, and the BUSY# pin is low. The self-test takes about 2**19 clocks, or approximately 26 milliseconds with a 20 MHz 386 DX. At the completion of self-test the processor performs reset and begins normal operation. The part has successfully passed self-test if the contents of the EAX register are zero (0). If the results of EAX are not zero then the self-test has detected a flaw in the part.

2.11.2 TLB Testing

The 386 DX provides a mechanism for testing the Translation Lookaside Buffer (TLB) if desired. This particular mechanism is unique to the 386 DX and may not be continued in the same way in future processors. When testing the TLB paging must be turned off (PG = 0 in CR0) to enable the TLB testing hardware and avoid interference with the test data being written to the TLB.

There are two TLB testing operations: 1) write entries into the TLB, and, 2) perform TLB lookups. Two Test Registers, shown in Figure 2-12, are provided for the purpose of testing. TR6 is the "test command register", and TR7 is the "test data register". The fields within these registers are defined below.

C: This is the command bit. For a write into TR6 to cause an immediate write into the TLB entry, write a 0 to this bit. For a write into TR6 to cause an immediate TLB lookup, write a 1 to this bit.

Linear Address: This is the tag field of the TLB. On a TLB write, a TLB entry is allocated to this linear address and the rest of that TLB entry is set per the value of TR7 and the value just written into TR6. On a TLB lookup, the TLB is interrogated per this value and if one and only one TLB entry matches, the rest of the fields of TR6 and TR7 are set from the matching TLB entry.

Physical Address: This is the data field of the TLB. On a write to the TLB, the TLB entry allocated to the linear address in TR6 is set to this value. On a TLB lookup, the data field (physical address) from the TLB is read out to here.

PL: On a TLB write, PL=1 causes the REP field of TR7 to select which of four associative blocks of the TLB is to be written, but PL=0 allows the internal pointer in the paging unit to select which TLB block is written. On a TLB lookup, the PL bit indicates whether the lookup was a hit (PL gets set to 1) or a miss (PL gets reset to 0).

V: The valid bit for this TLB entry. All valid bits can also be cleared by writing to CR3.

D, D#: The dirty bit for/from the TLB entry.

U, U#: The user bit for/from the TLB entry.

W, W#: The writable bit for/from the TLB entry.

For D, U and W, both the attribute and its complement are provided as tag bits, to permit the option of a "don't care" on TLB lookups. The meaning of these pairs of bits is given in the following table:

X	X#	Effect During TLB Lookup	Value of Bit X after TLB Write
0	0	Miss All	Bit X Becomes Undefined
0	1	Match if X = 0	Bit X Becomes 0
1	0	Match if X = 1	Bit X Becomes 1
1	1	Match all	Bit X Becomes Undefined

For writing a TLB entry:

1. Write TR7 for the desired physical address, PL and REP values.
2. Write TR6 with the appropriate linear address, etc. (be sure to write C = 0 for "write" command).

For looking up (reading) a TLB entry:

1. Write TR6 with the appropriate linear address (be sure to write C=1 for "lookup" command).
2. Read TR7 and TR6. If the PL bit in TR7 indicates a hit, then the other values reveal the TLB contents. If PL indicates a miss, then the other values in TR7 and TR6 are indeterminate.

2.12 DEBUGGING SUPPORT

The 386 DX provides several features which simplify the debugging process. The three categories of on-chip debugging aids are:

1) the code execution breakpoint opcode (0CCH),
2) the single-step capability provided by the TF bit in the flag register, and
3) the code and data breakpoint capability provided by the Debug Registers DR0-3, DR6, and DR7.

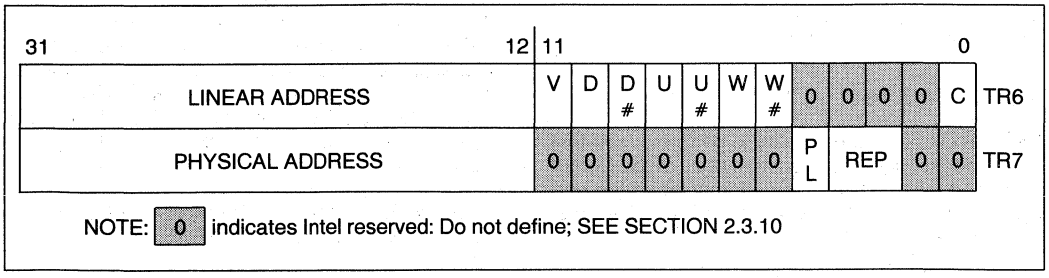


Figure 2-12. Test Registers

2.12.1 Breakpoint Instruction

A single-byte-opcode breakpoint instruction is available for use by software debuggers. The breakpoint opcode is 0CCh, and generates an exception 3 trap when executed. In typical use, a debugger program can "plant" the breakpoint instruction at all desired code execution breakpoints. The single-byte breakpoint opcode is an alias for the two-byte general software interrupt instruction, INT n, where n=3. The only difference between INT 3 (0CCh) and INT n is that INT 3 is never IOPL-sensitive but INT n is IOPL-sensitive in Protected Mode and Virtual 8086 Mode.

2.12.2 Single-Step Trap

If the single-step flag (TF, bit 8) in the EFLAG register is found to be set at the end of an instruction, a single-step exception occurs. The single-step exception is auto vectored to exception number 1. Precisely, exception 1 occurs as a trap after the instruction following the instruction which set TF. In typical practice, a debugger sets the TF bit of a flag register image on the debugger's stack. It then typically transfers control to the user program and loads the flag image with a signal instruction, the IRET instruction. The single-step trap occurs after executing one instruction of the user program.

Since the exception 1 occurs as a trap (that is, it occurs after the instruction has already executed), the CS:EIP pushed onto the debugger's stack points to the next unexecuted instruction of the program being debugged. An exception 1 handler, merely by ending with an IRET instruction, can therefore efficiently support single-stepping through a user program.

2.12.3 Debug Registers

The Debug Registers are an advanced debugging feature of the 386 DX. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution breakpoint can be placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT3 breakpoint opcode.

The 386 DX contains six Debug Registers, providing the ability to specify up to four distinct breakpoints addresses, breakpoint control options, and read breakpoint status. Initially after reset, breakpoints are in the disabled state. Therefore, no breakpoints will occur unless the debug registers are programmed. Breakpoints set up in the Debug Registers are autovectored to exception number 1.

2.12.3.1 LINEAR ADDRESS BREAKPOINT REGISTERS (DR0–DR3)

Up to four breakpoint addresses can be specified by writing into Debug Registers DR0–DR3, shown in Figure 2-13. The breakpoint addresses specified are 32-bit linear addresses. 386 DX hardware continuously compares the linear breakpoint addresses in DR0–DR3 with the linear addresses generated by executing software (a linear address is the result of computing the effective address and adding the 32-bit segment base address). Note that if paging is not enabled the linear address equals the physical address. If paging is enabled, the linear address is translated to a physical 32-bit address by the on-chip paging unit. Regardless of whether paging is enabled or not, however, the breakpoint registers hold linear addresses.

2.12.3.2 DEBUG CONTROL REGISTER (DR7)

A Debug Control Register, DR7 shown in Figure 2-13, allows several debug control functions such as enabling the breakpoints and setting up other control options for the breakpoints. The fields within the Debug Control Register, DR7, are as follows:

LENi (breakpoint length specification bits)

A 2-bit LEN field exists for each of the four breakpoints. LEN specifies the length of the associated breakpoint field. The choices for data breakpoints are: 1 byte, 2 bytes, and 4 bytes. Instruction execu-

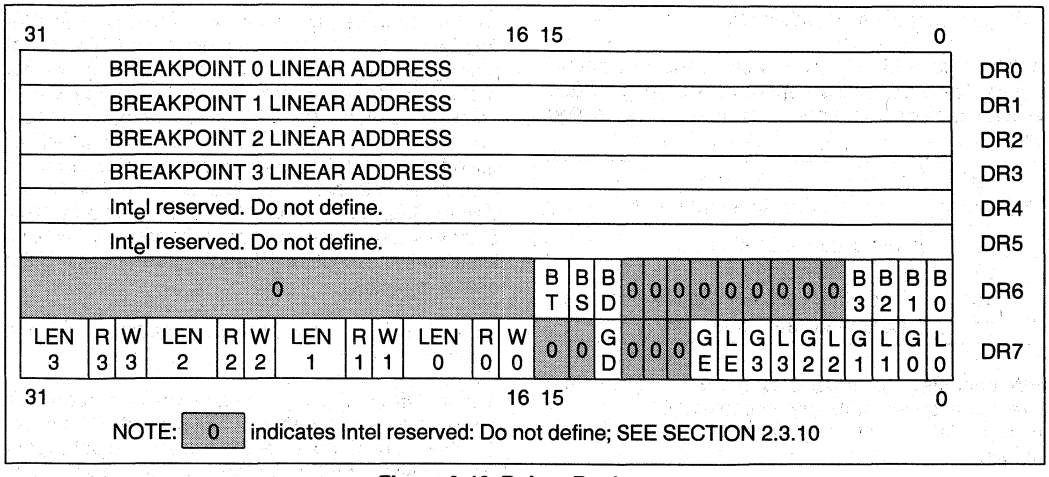


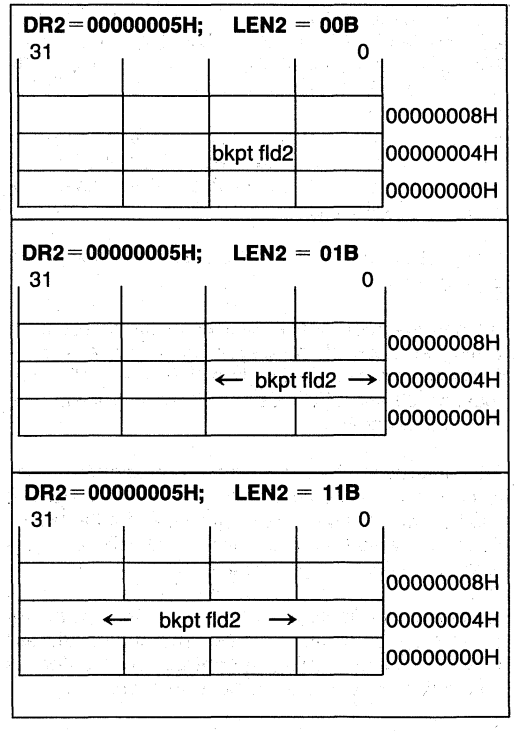
Figure 2-13. Debug Registers

tion breakpoints must have a length of 1 (LENi = 00). Encoding of the LENi field is as follows:

LENi Encoding	Breakpoint Field Width	Usage of Least Significant Bits in Breakpoint Address Register i, (i=0−3)
00	1 byte	All 32-bits used to specify a single-byte breakpoint field.
01	2 bytes	A1–A31 used to specify a two-byte, word-aligned breakpoint field. A0 in Breakpoint Address Register is not used.
10	Undefined—do not use this encoding	
11	4 bytes	A2–A31 used to specify a four-byte, dword-aligned breakpoint field. A0 and A1 in Breakpoint Address Register are not used.

The LENi field controls the size of breakpoint field i by controlling whether all low-order linear address bits in the breakpoint address register are used to detect the breakpoint event. Therefore, all breakpoint fields are aligned; 2-byte breakpoint fields begin on Word boundaries, and 4-byte breakpoint fields begin on Dword boundaries.

The following is an example of various size breakpoint fields. Assume the breakpoint linear address in DR2 is 00000005H. In that situation, the following illustration indicates the region of the breakpoint field for lengths of 1, 2, or 4 bytes.



RWi (memory access qualifier bits)

A 2-bit RW field exists for each of the four breakpoints. The 2-bit RW field specifies the type of usage which must occur in order to activate the associated breakpoint.

RW Encoding	Usage Causing Breakpoint
00	Instruction execution only
01	Data writes only
10	Undefined—do not use this encoding
11	Data reads and writes only

RW encoding 00 is used to set up an instruction execution breakpoint. RW encodings 01 or 11 are used to set up write-only or read/write data breakpoints.

Note that **instruction execution breakpoints are taken as faults** (i.e. before the instruction executes), but **data breakpoints are taken as traps** (i.e. after the data transfer takes place).

Using LENi and RWi to Set Data Breakpoint i

A data breakpoint can be set up by writing the linear address into DRi (i = 0–3). For data breakpoints, RWi can = 01 (write-only) or 11 (write/read). LEN can = 00, 01, or 11.

If a data access entirely or partly falls within the data breakpoint field, the data breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 trap will occur.

Using LENi and RWi to Set Instruction Execution Breakpoint i

An instruction execution breakpoint can be set up by writing address of the beginning of the instruction (including prefixes if any) into DRi (i = 0–3). RWi must = 00 and LEN must = 00 for instruction execution breakpoints.

If the instruction beginning at the breakpoint address is about to be executed, the instruction execution breakpoint condition has occurred, and if the breakpoint is enabled, an exception 1 fault will occur before the instruction is executed.

Note that an instruction execution breakpoint address must be equal to the **beginning** byte address of an instruction (including prefixes) in order for the instruction execution breakpoint to occur.

GD (Global Debug Register access detect)

The Debug Registers can only be accessed in Real Mode or at privilege level 0 in Protected Mode. The GD bit, when set, provides extra protection against **any** Debug Register access even in Real Mode or at privilege level 0 in Protected Mode. This additional protection feature is provided to guarantee that a software debugger (or ICE™-386) can have full control over the Debug Register resources when required. The GD bit, when set, causes an exception 1 fault if an instruction attempts to read or write any Debug Register. The GD bit is then automatically cleared when the exception 1 handler is invoked, allowing the exception 1 handler free access to the debug registers.

GE and LE (Exact data breakpoint match, global and local)

If either GE or LE is set, any data breakpoint trap will be reported exactly after completion of the instruction that caused the operand transfer. Exact reporting is provided by forcing the 386 DX execution unit to wait for completion of data operand transfers before beginning execution of the next instruction.

If exact data breakpoint match is not selected, data breakpoints may not be reported until several instructions later or may not be reported at all. When enabling a data breakpoint, it is therefore recommended to enable the exact data breakpoint match.

When the 386 DX performs a task switch, the LE bit is cleared. Thus, the LE bit supports fast task switching out of tasks, that have enabled the exact data breakpoint match for their task-local breakpoints. The LE bit is cleared by the processor during a task switch, to avoid having exact data breakpoint match enabled in the new task. Note that exact data breakpoint match must be re-enabled under software control.

The 386 DX GE bit is unaffected during a task switch. The GE bit supports exact data breakpoint match that is to remain enabled during all tasks executing in the system.

Note that **instruction execution** breakpoints are always reported exactly, whether or not exact data breakpoint match is selected.

Gi and Li (breakpoint enable, global and local)

If either Gi or Li is set then the associated breakpoint (as defined by the linear address in DRi, the length in LENi and the usage criteria in RWi) is enabled. If either Gi or Li is set, and the 386 DX detects the ith breakpoint condition, then the exception 1 handler is invoked.

When the 386 DX performs a task switch to a new Task State Segment (TSS), all Li bits are cleared. Thus, the Li bits support fast task switching out of tasks that use some task-local breakpoint

registers. The Li bits are cleared by the processor during a task switch, to avoid spurious exceptions in the new task. Note that the breakpoints must be re-enabled under software control.

All 386 DX Gi bits are unaffected during a task switch. The Gi bits support breakpoints that are active in all tasks executing in the system.

2.12.3.3 DEBUG STATUS REGISTER (DR6)

A Debug Status Register, DR6 shown in Figure 2-13, allows the exception 1 handler to easily determine why it was invoked. Note the exception 1 handler can be invoked as a result of one of several events:

1) DR0 Breakpoint fault/trap.
2) DR1 Breakpoint fault/trap.
3) DR2 Breakpoint fault/trap.
4) DR3 Breakpoint fault/trap.
5) Single-step (TF) trap.
6) Task switch trap.
7) Fault due to attempted debug register access when GD=1.

The Debug Status Register contains single-bit flags for each of the possible events invoking exception 1. Note below that some of these events are faults (exception taken before the instruction is executed), while other events are traps (exception taken after the debug events occurred).

The flags in DR6 are set by the hardware but never cleared by hardware. Exception 1 handler software should clear DR6 before returning to the user program to avoid future confusion in identifying the source of exception 1.

The fields within the Debug Status Register, DR6, are as follows:

Bi (debug fault/trap due to breakpoint 0–3)

Four breakpoint indicator flags, B0–B3, correspond one-to-one with the breakpoint registers in DR0–DR3. A flag Bi is set when the condition described by DRi, LENi, and RWi occurs.

If Gi or Li is set, and if the ith breakpoint is detected, the processor will invoke the exception 1 handler. The exception is handled as a fault if an instruction execution breakpoint occurred, or as a trap if a data breakpoint occurred.

IMPORTANT NOTE: A flag Bi is set whenever the hardware detects a match condition on **enabled** breakpoint i. Whenever a match is detected on at least one **enabled** breakpoint i, the hardware immediately sets all Bi bits corresponding to breakpoint conditions matching at that instant, whether enabled **or not.** Therefore, the exception 1 handler may see that multiple Bi bits are set, but only set Bi bits corresponding to **enabled** breakpoints (Li or Gi set) are **true** indications of why the exception 1 handler was invoked.

BD (debug fault due to attempted register access when GD bit set)

This bit is set if the exception 1 handler was invoked due to an instruction attempting to read or write to the debug registers when GD bit was set. If such an event occurs, then the GD bit is automatically cleared when the exception 1 handler is invoked, allowing handler access to the debug registers.

BS (debug trap due to single-step)

This bit is set if the exception 1 handler was invoked due to the TF bit in the flag register being set (for single-stepping). See section 2.12.2.

BT (debug trap due to task switch)

This bit is set if the exception 1 handler was invoked due to a task switch occurring to a task having a 386 DX TSS with the T bit set. (See Figure 4-15a). Note the task switch into the new task occurs normally, but before the first instruction of the task is executed, the exception 1 handler is invoked. With respect to the task switch operation, the operation is considered to be a trap.

2.12.3.4 USE OF RESUME FLAG (RF) IN FLAG REGISTER

The Resume Flag (RF) in the flag word can suppress an instruction execution breakpoint when the exception 1 handler returns to a user program at a user address which is also an instruction execution breakpoint. See section 2.3.3.

3. REAL MODE ARCHITECTURE

3.1 REAL MODE INTRODUCTION

When the processor is reset or powered up it is initialized in Real Mode. Real Mode has the same base architecture as the 8086, but allows access to the 32-bit register set of the 386 DX. The addressing mechanism, memory size, interrupt handling, are all identical to the Real Mode on the 80286.

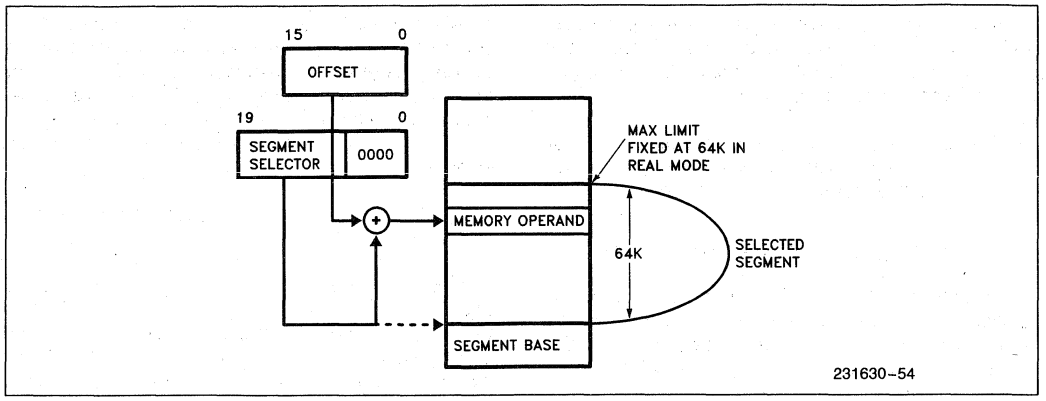


Figure 3-1. Real Address Mode Addressing

All of the 386 DX instructions are available in Real Mode (except those instructions listed in 4.6.4). The default operand size in Real Mode is 16-bits, just like the 8086. In order to use the 32-bit registers and addressing modes, override prefixes must be used. In addition, the segment size on the 386 DX in Real Mode is 64K bytes so 32-bit effective addresses must have a value less the 0000FFFFH. The primary purpose of Real Mode is to set up the processor for Protected Mode Operation.

The LOCK prefix on the 386 DX, even in Real Mode, is more restrictive than on the 80286. This is due to the addition of paging on the 386 DX in Protected Mode and Virtual 8086 Mode. Paging makes it impossible to guarantee that repeated string instructions can be LOCKed. The 386 DX can't require that all pages holding the string be physically present in memory. Hence, a Page Fault (exception 14) might have to be taken during the repeated string instruction. Therefore the LOCK prefix can't be supported during repeated string instructions.

These are the only instruction forms where the LOCK prefix is legal on the 386 DX:

Opcode	Operands (Dest, Source)
BIT Test and SET/RESET/COMPLEMENT	Mem, Reg/immed
XCHG	Reg, Mem
XCHG	Mem, Reg
ADD, OR, ADC, SBB, AND, SUB, XOR	Mem, Reg/immed
NOT, NEG, INC, DEC	Mem

An exception 6 will be generated if a LOCK prefix is placed before any instruction form or opcode not listed above. The LOCK prefix allows indivisible read/modify/write operations on memory operands using the instructions above. For example, even the ADD Reg, Mem is not LOCKable, because the Mem operand is not the destination (and therefore no memory read/modify/operation is being performed).

Since, on the 386 DX, repeated string instructions are not LOCKable, it is not possible to LOCK the bus for a long period of time. Therefore, the LOCK prefix is not IOPL-sensitive on the 386 DX. The LOCK prefix can be used at any privilege level, but only on the instruction forms listed above.

3.2 MEMORY ADDRESSING

In Real Mode the maximum memory size is limited to 1 megabyte. Thus, only address lines A2–A19 are active. (Exception, the high address lines A20–A31 are high during CS-relative memory cycles until an intersegment jump or call is executed (see section 2.10)).

Since paging is not allowed in Real Mode the linear addresses are the same as physical addresses. Physical addresses are formed in Real Mode by adding the contents of the appropriate segment register which is shifted left by four bits to an effective address. This addition results in a physical address from 00000000H to 0010FFEFH. This is compatible with 80286 Real Mode. Since segment registers are shifted left by 4 bits this implies that Real Mode segments always start on 16 byte boundaries.

All segments in Real Mode are exactly 64K bytes long, and may be read, written, or executed. The 386 DX will generate an exception 13 if a data operand or instruction fetch occurs past the end of a segment. (i.e. if an operand has an offset greater than FFFFH, for example a word with a low byte at FFFFH and the high byte at 0000H.)

Segments may be overlapped in Real Mode. Thus, if a particular segment does not use all 64K bytes another segment can be overlayed on top of the unused portion of the previous segment. This allows the programmer to minimize the amount of physical memory needed for a program.

3.3 RESERVED LOCATIONS

There are two fixed areas in memory which are reserved in Real address mode: system initialization area and the interrupt table area. Locations 00000H through 003FFH are reserved for interrupt vectors. Each one of the 256 possible interrupts has a 4-byte jump vector reserved for it. Locations FFFFFFF0H through FFFFFFFFH are reserved for system initialization.

3.4 INTERRUPTS

Many of the exceptions shown in Table 2-5 and discussed in section 2.9 are not applicable to Real Mode operation, in particular exceptions 10, 11, 14, will not happen in Real Mode. Other exceptions have slightly different meanings in Real Mode; Table 3-1 identifies these exceptions.

3.5 SHUTDOWN AND HALT

The HLT instruction stops program execution and prevents the processor from using the local bus until restarted. Either NMI, INTR with interrupts enabled (IF = 1), or RESET will force the 386 DX out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

Shutdown will occur when a severe error is detected that prevents further processing. In Real Mode, shutdown can occur under two conditions:

An interrupt or an exception occur (Exceptions 8 or 13) and the interrupt vector is larger than the Interrupt Descriptor Table (i.e. There is not an interrupt handler for the interrupt).

A CALL, INT or PUSH instruction attempts to wrap around the stack segment when SP is not even. (e.g. pushing a value on the stack when SP = 0001 resulting in a stack segment greater than FFFFH)

An NMI input can bring the processor out of shutdown if the Interrupt Descriptor Table limit is large enough to contain the NMI interrupt vector (at least 0017H) and the stack has enough room to contain the vector and flag information (i.e. SP is greater than 0005H). Otherwise shutdown can only be exited via the RESET input.

4. PROTECTED MODE ARCHITECTURE

4.1 INTRODUCTION

The complete capabilities of the 386 DX are unlocked when the processor operates in Protected Virtual Address Mode (Protected Mode). Protected Mode vastly increases the linear address space to four gigabytes (2^{32} bytes) and allows the running of virtual memory programs of almost unlimited size (64 terabytes or 2^{46} bytes). In addition Protected Mode allows the 386 DX to run all of the existing 8086 and 80286 software, while providing a sophisticated memory management and a hardware-assisted protection mechanism. Protected Mode allows the use of additional instructions especially optimized for supporting multitasking operating systems. The base architecture of the 386 DX remains the same, the registers, instructions, and addressing modes described in the previous sections are retained. The main difference between Protected Mode, and Real Mode from a programmer's view is the increased address space, and a different addressing mechanism.

5

Table 3-1

Function	Interrupt Number	Related Instructions	Return Address Location
Interrupt table limit too small	8	INT Vector is not within table limit	Before Instruction
CS, DS, ES, FS, GS Segment overrun exception	13	Word memory reference beyond offset = FFFFH. An attempt to execute past the end of CS segment.	Before Instruction
SS Segment overrun exception	12	Stack Reference beyond offset = FFFFH	Before Instruction

4.2 ADDRESSING MECHANISM

Like Real Mode, Protected Mode uses two components to form the logical address, a 16-bit selector is used to determine the linear base address of a segment, the base address is added to a 32-bit effective address to form a 32-bit linear address. The linear address is then either used as the 32-bit physical address, or if paging is enabled the paging mechanism maps the 32-bit linear address into a 32-bit physical address.

The difference between the two modes lies in calculating the base address. In Protected Mode the selector is used to specify an index into an operating system defined table (see Figure 4-1). The table contains the 32-bit base address of a given segment. The physical address is formed by adding the base address obtained from the table to the offset.

Paging provides an additional memory management mechanism which operates only in Protected Mode. Paging provides a means of managing the very large segments of the 386 DX. As such, paging operates beneath segmentation. The paging mechanism translates the protected linear address which comes from the segmentation unit into a physical address. Figure 4-2 shows the complete 386 DX addressing mechanism with paging enabled.

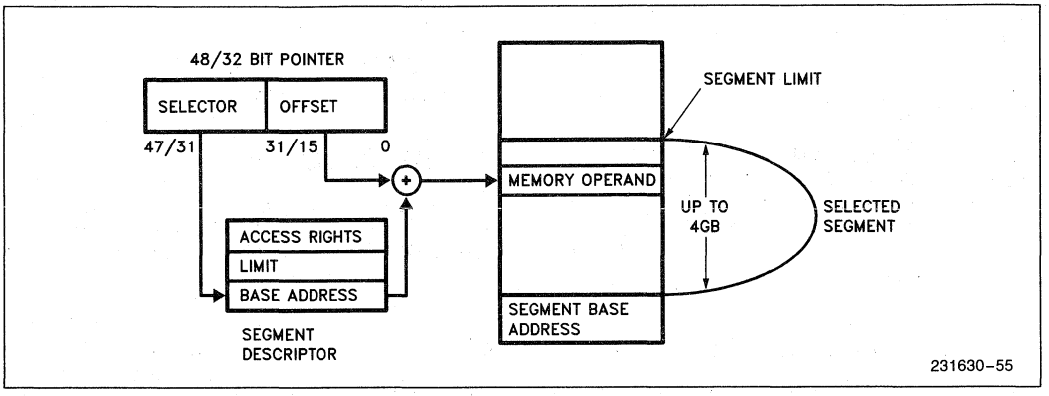


Figure 4-1. Protected Mode Addressing

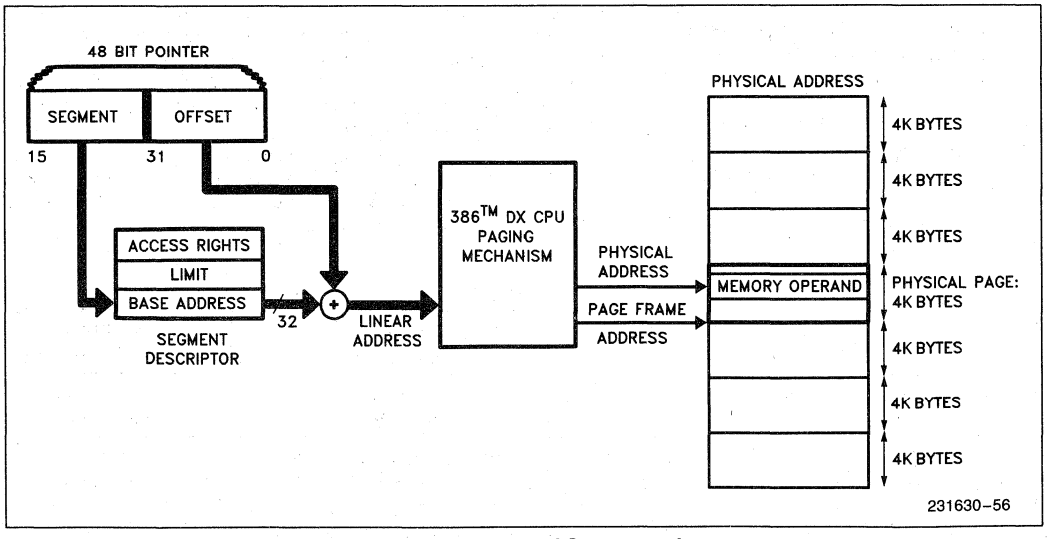


Figure 4-2. Paging and Segmentation

4.3 SEGMENTATION

4.3.1 Segmentation Introduction

Segmentation is one method of memory management. Segmentation provides the basis for protection. Segments are used to encapsulate regions of memory which have common attributes. For example, all of the code of a given program could be contained in a segment, or an operating system table may reside in a segment. All information about a segment is stored in an 8 byte data structure called a descriptor. All of the descriptors in a system are contained in tables recognized by hardware.

4.3.2 Terminology

The following terms are used throughout the discussion of descriptors, privilege levels and protection:

PL: Privilege Level—One of the four hierarchical privilege levels. Level 0 is the most privileged level and level 3 is the least privileged. More privileged levels are numerically smaller than less privileged levels.

RPL: Requestor Privilege Level—The privilege level of the original supplier of the selector. RPL is determined by the **least two** significant bits of a selector.

DPL: Descriptor Privilege Level—This is the least privileged level at which a task may access that descriptor (and the segment associated with that descriptor). Descriptor Privilege Level is determined by bits 6:5 in the Access Right Byte of a descriptor.

CPL: Current Privilege Level—The privilege level at which a task is currently executing, which equals the privilege level of the code segment being executed. CPL can also be determined by examining the lowest 2 bits of the CS register, except for conforming code segments.

EPL: Effective Privilege Level—The effective privilege level is the least privileged of the RPL and DPL. Since smaller privilege level **values** indicate greater privilege, EPL is the numerical maximum of RPL and DPL.

Task: One instance of the execution of a program. Tasks are also referred to as processes.

4.3.3 Descriptor Tables

4.3.3.1 DESCRIPTOR TABLES INTRODUCTION

The descriptor tables define all of the segments which are used in an 386 DX system. There are three types of tables on the 386 DX which hold descriptors: the Global Descriptor Table, Local Descriptor Table, and the Interrupt Descriptor Table. All of the tables are variable length memory arrays. They can range in size between 8 bytes and 64K bytes. Each table can hold up to 8192 8 byte descriptors. The upper 13 bits of a selector are used as an index into the descriptor table. The tables have registers associated with them which hold the 32-bit linear base address, and the 16-bit limit of each table.

Each of the tables has a register associated with it the GDTR, LDTR, and the IDTR (see Figure 4-3). The LGDT, LLDT, and LIDT instructions, load the base and limit of the Global, Local, and Interrupt Descriptor Tables, respectively, into the appropriate register. The SGDT, SLDT, and SIDT instructions store the base and limit values. These tables are manipulated by the operating system. Therefore, the load descriptor table instructions are privileged instructions.

4.3.3.2 GLOBAL DESCRIPTOR TABLE

The Global Descriptor Table (GDT) contains descriptors which are possibly available to all of the tasks in a system. The GDT can contain any type of segment descriptor except for descriptors which are used for servicing interrupts (i.e. interrupt and trap descriptors). Every 386 DX system contains a

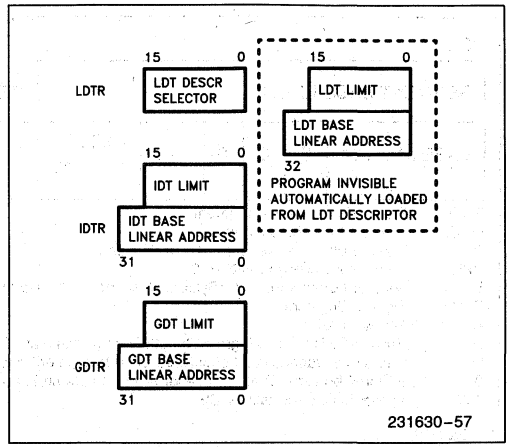


Figure 4-3. Descriptor Table Registers

GDT. Generally the GDT contains code and data segments used by the operating systems and task state segments, and descriptors for the LDTs in a system.

The first slot of the Global Descriptor Table corresponds to the null selector and is not used. The null selector defines a null pointer value.

4.3.3.3 LOCAL DESCRIPTOR TABLE

LDTs contain descriptors which are associated with a given task. Generally, operating systems are designed so that each task has a separate LDT. The LDT may contain only code, data, stack, task gate, and call gate descriptors. LDTs provide a mechanism for isolating a given task's code and data segments from the rest of the operating system, while the GDT contains descriptors for segments which are common to all tasks. A segment cannot be accessed by a task if its segment descriptor does not exist in either the current LDT or the GDT. This provides both isolation and protection for a task's segments, while still allowing global data to be shared among tasks.

Unlike the 6 byte GDT or IDT registers which contain a base address and limit, the visible portion of the LDT register contains only a 16-bit selector. This selector refers to a Local Descriptor Table descriptor in the GDT.

4.3.3.4 INTERRUPT DESCRIPTOR TABLE

The third table needed for 386 DX systems is the Interrupt Descriptor Table. (See Figure 4-4.) The IDT contains the descriptors which point to the location of up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates, and trap gates. The IDT should be at least 256 bytes in size in order to hold the descriptors for the 32 Intel Reserved Interrupts. Every interrupt used by a system must have an entry in the IDT. The IDT entries are referenced via INT instructions, external interrupt vectors, and exceptions. (See 2.9 **Interrupts).**

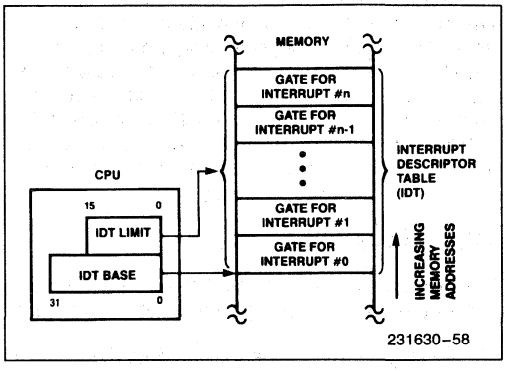
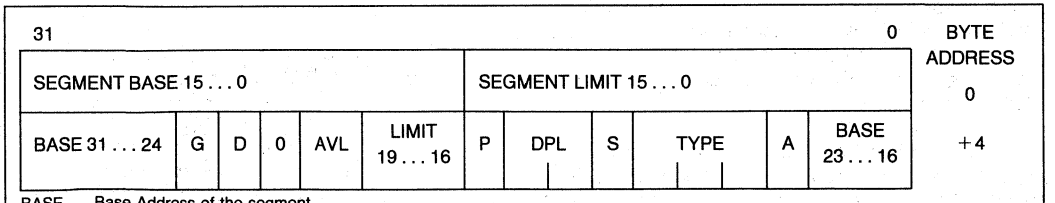


Figure 4-4. Interrupt Descriptor Table Register Use

4.3.4 Descriptors

4.3.4.1 DESCRIPTOR ATTRIBUTE BITS

The object to which the segment selector points to is called a descriptor. Descriptors are eight byte quantities which contain attributes about a given region of linear address space (i.e. a segment). These attributes include the 32-bit base linear address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, the default size of the operands (16-bit or



BASE	Base Address of the segment
LIMIT	The length of the segment
P	Present Bit 1 = Present 0 = Not Present
DPL	Descriptor Privilege Level 0–3
S	Segment Descriptor 0 = System Descriptor 1 = Code or Data Segment Descriptor
TYPE	Type of Segment
A	Accessed Bit
G	Granularity Bit 1 = Segment length is page granular 0 = Segment length is byte granular
D	Default Operation Size (recognized in code segment descriptors only) 1 = 32-bit segment 0 = 16-bit segment
0	Bit must be zero (0) for compatibility with future processors
AVL	Available field for user or OS

NOTE:
In a maximum-size segment (ie. a segment with G = 1 and segment limit 19...0 = FFFFFH), the lowest 12 bits of the segment base should be zero (ie. segment base 11...000 = 000H).

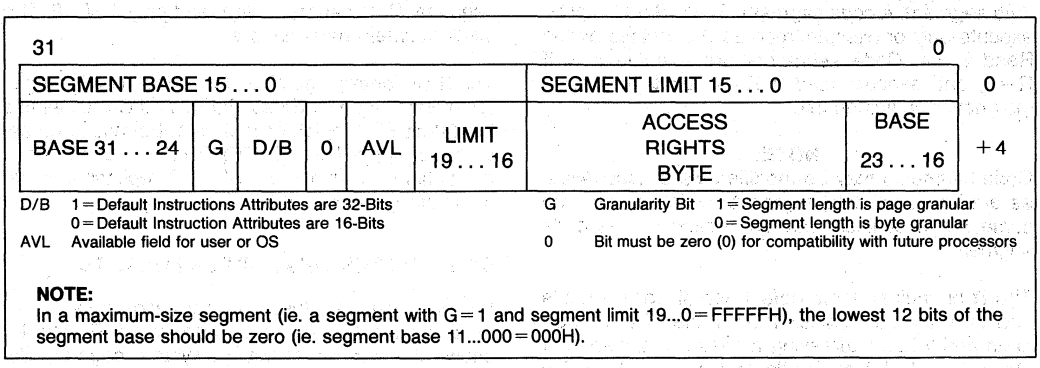
Figure 4-5. Segment Descriptors

32-bit), and the type of segment. All of the attribute information about a segment is contained in 12 bits in the segment descriptor. Figure 4-5 shows the general format of a descriptor. All segments on the 386 DX have three attribute fields in common: the **P** bit, the **DPL** bit, and the **S** bit. The Present **P** bit is 1 if the segment is loaded in physical memory, if P=0 then any attempt to access this segment causes a not present exception (exception 11). The Descriptor Privilege Level **DPL** is a two-bit field which specifies the protection level 0–3 associated with a segment.

The 386 DX has two main categories of segments system segments and non-system segments (for code and data). The segment **S** bit in the segment descriptor determines if a given segment is a system segment or a code or data segment. If the S bit is 1 then the segment is either a code or data segment, if it is 0 then the segment is a system segment.

4.3.4.2 386™ DX CODE, DATA DESCRIPTORS (S=1)

Figure 4-6 shows the general format of a code and data descriptor and Table 4-1 illustrates how the bits in the Access Rights Byte are interpreted.



NOTE:
In a maximum-size segment (ie. a segment with G=1 and segment limit 19...0=FFFFFH), the lowest 12 bits of the segment base should be zero (ie. segment base 11...000=000H).

Figure 4-6. Segment Descriptors

Table 4-1. Access Rights Byte Definition for Code and Data Descriptions

	Bit Position	Name	Function	
Type Field Definition	7	Present (P)	P = 1 Segment is mapped into physical memory. P = 0 No mapping to physical memory exits, base and limit are not used.	
	6–5	Descriptor Privilege Level (DPL)	Segment privilege attribute used in privilege tests.	
	4	Segment Descriptor (S)	S = 1 Code or Data (includes stacks) segment descriptor S = 0 System Segment Descriptor or Gate Descriptor	
	3	Executable (E)	E = 0 Descriptor type is data segment:	If Data Segment (S = 1, E = 0)
	2	Expansion Direction (ED)	ED = 0 Expand up segment, offsets must be ≤ limit. ED = 1 Expand down segment, offsets must be > limit.	
	1	Writeable (W)	W = 0 Data segment may not be written into. W = 1 Data segment may be written into.	
	3	Executable (E)	E = 1 Descriptor type is code segment:	If Code Segment (S = 1, E = 1)
	2	Conforming (C)	C = 1 Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.	
	1	Readable (R)	R = 0 Code segment may not be read. R = 1 Code segment may be read.	
	0	Accessed (A)	A = 0 Segment has not been accessed. A = 1 Segment selector has been loaded into segment register or used by selector test instructions.	

Code and data segments have several descriptor fields in common. The accessed **A** bit is set whenever the processor accesses a descriptor. The **A** bit is used by operating systems to keep usage statistics on a given segment. The **G** bit, or granularity bit, specifies if a segment length is byte-granular or page-granular. 386 DX segments can be one megabyte long with byte granularity (G=0) or four gigabytes with page granularity (G=1), (i.e., 2^{20} pages each page is 4K bytes in length). The granularity is totally unrelated to paging. A 386 DX system can consist of segments with byte granularity, and page granularity, whether or not paging is enabled.

The executable **E** bit tells if a segment is a code or data segment. A code segment (E=1, S=1) may be execute-only or execute/read as determined by the Read **R** bit. Code segments are execute only if R=0, and execute/read if R=1. Code segments may never be written into.

NOTE:
Code segments may be modified via aliases. Aliases are writeable data segments which occupy the same range of linear address space as the code segment.

The **D** bit indicates the default length for operands and effective addresses. If D=1 then 32-bit operands and 32-bit addressing modes are assumed. If D=0 then 16-bit operands and 16-bit addressing modes are assumed. Therefore all existing 80286 code segments will execute on the 386 DX assuming the D bit is set 0.

Another attribute of code segments is determined by the conforming **C** bit. Conforming segments, C=1, can be executed and shared by programs at different privilege levels. (See section 4.4 **Protection.**)

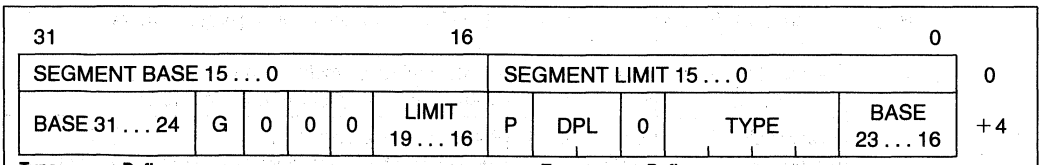
Segments identified as data segments (E=0, S=1) are used for two types of 386 DX segments: stack and data segments. The expansion direction **(ED)** bit specifies if a segment expands downward (stack) or upward (data). If a segment is a stack segment all offsets must be greater than the segment limit. On a data segment all offsets must be less than or equal to the limit. In other words, stack segments start at the base linear address plus the maximum segment limit and grow down to the base linear address plus the limit. On the other hand, data segments start at the base linear address and expand to the base linear address plus limit.

The write **W** bit controls the ability to write into a segment. Data segments are read-only if W=0. The stack segment must have W=1.

The **B** bit controls the size of the stack pointer register. If B=1, then PUSHes, POPs, and CALLs all use the 32-bit ESP register for stack references and assume an upper limit of FFFFFFFFH. If B=0, stack instructions all use the 16-bit SP register and assume an upper limit of FFFFH.

4.3.4.3 SYSTEM DESCRIPTOR FORMATS

System segments describe information about operating system tables, tasks, and gates. Figure 4-7 shows the general format of system segment descriptors, and the various types of system segments. 386 DX system descriptors contain a 32-bit base linear address and a 20-bit segment limit. 80286 system descriptors have a 24-bit base address and a 16-bit segment limit. 80286 system descriptors are identified by the upper 16 bits being all zero.



Type	Defines	Type	Defines
0	Invalid	8	Invalid
1	Available 80286 TSS	9	Available 386™ DX TSS
2	LDT	A	Undefined (Intel Reserved)
3	Busy 80286 TSS	B	Busy 386™ DX TSS
4	80286 Call Gate	C	386™ DX Call Gate
5	Task Gate (for 80286 or 386™ DX Task)	D	Undefined (Intel Reserved)
6	80286 Interrupt Gate	E	386™ DX Interrupt Gate
7	80286 Trap Gate	F	386™ DX Trap Gate

NOTE:
In a maximum-size segment (ie. a segment with G=1 and segment limit 19...0=FFFFFH), the lowest 12 bits of the segment base should be zero (ie. segment base 11...000=000H).

Figure 4-7. System Segments Descriptors

4.3.4.4 LDT DESCRIPTORS (S=0, TYPE=2)

LDT descriptors (S=0 TYPE=2) contain information about Local Descriptor Tables. LDTs contain a table of segment descriptors, unique to a particular task. Since the instruction to load the LDTR is only available at privilege level 0, the DPL field is ignored. LDT descriptors are only allowed in the Global Descriptor Table (GDT).

4.3.4.5 TSS DESCRIPTORS (S=0, TYPE=1, 3, 9, B)

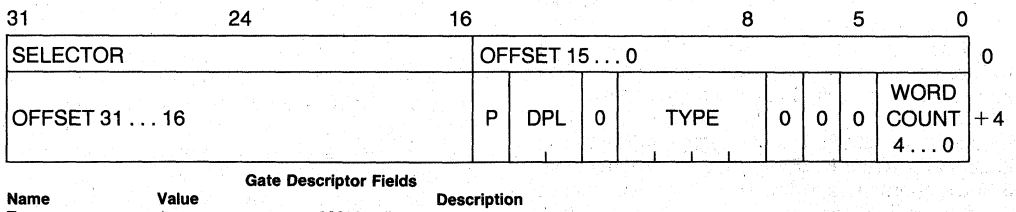
A Task State Segment (TSS) descriptor contains information about the location, size, and privilege level of a Task State Segment (TSS). A TSS in turn is a special fixed format segment which contains all the state information for a task and a linkage field to permit nesting tasks. The TYPE field is used to indicate whether the task is currently BUSY (i.e. on a chain of active tasks) or the TSS is available. The TYPE field also indicates if the segment contains a 80286 or a 386 DX TSS. The Task Register (TR) contains the selector which points to the current Task State Segment.

4.3.4.6 GATE DESCRIPTORS (S=0, TYPE=4-7, C, F)

Gates are used to control access to entry points within the target code segment. The various types of gate descriptors are **call** gates, **task** gates, **interrupt** gates, and **trap** gates. Gates provide a level of indirection between the source and destination of the control transfer. This indirection allows the processor to automatically perform protection checks. It also allows system designers to control entry points to the operating system. Call gates are used to change privilege levels (see section 4.4 **Protection**), task gates are used to perform a task switch, and interrupt and trap gates are used to specify interrupt service routines.

Figure 4-8 shows the format of the four types of gate descriptors. Call gates are primarily used to transfer program control to a more privileged level. The call gate descriptor consists of three fields: the access byte, a long pointer (selector and offset) which points to the start of a routine and a word count which specifies how many parameters are to be copied from the caller's stack to the stack of the called routine. The word count field is only used by call gates when there is a change in the privilege level, other types of gates ignore the word count field.

Interrupt and trap gates use the destination selector and destination offset fields of the gate descriptor as a pointer to the start of the interrupt or trap handler routines. The difference between interrupt gates and trap gates is that the interrupt gate disables interrupts (resets the IF bit) while the trap gate does not.



Gate Descriptor Fields

Name	Value	Description
Type	4	80286 call gate
	5	Task gate (for 80286 or 386™ DX task)
	6	80286 interrupt gate
	7	80286 trap gate
	C	386™ DX call gate
	E	386™ DX interrupt gate
	F	386™ DX trap gate
P	0	Descriptor contents are not valid
	1	Descriptor contents are valid

DPL—least privileged level at which a task may access the gate. WORD COUNT 0-31—the number of parameters to copy from caller's stack to the called procedure's stack. The parameters are 32-bit quantities for 386™ DX gates, and 16-bit quantities for 80286 gates.

DESTINATION SELECTOR	16-bit selector	Selector to the target code segment or Selector to the target task state segment for task gate
DESTINATION OFFSET	offset 16-bit 80286 32-bit 386™ DX	Entry point within the target code segment

Figure 4-8. Gate Descriptor Formats

Task gates are used to switch tasks. Task gates may only refer to a task state segment (see section 4.4.6 **Task Switching**) therefore only the destination selector portion of a task gate descriptor is used, and the destination offset is ignored.

Exception 13 is generated when a destination selector does not refer to a correct descriptor type, i.e., a code segment for an interrupt, trap or call gate, a TSS for a task gate.

The access byte format is the same for all gate descriptors. P=1 indicates that the gate contents are valid. P=0 indicates the contents are not valid and causes exception 11 if referenced. DPL is the descriptor privilege level and specifies when this descriptor may be used by a task (see section 4.4 **Protection**). The S field, bit 4 of the access rights byte, must be 0 to indicate a system control descriptor. The type field specifies the descriptor type as indicated in Figure 4-8.

4.3.4.7 DIFFERENCES BETWEEN 386™ DX AND 80286 DESCRIPTORS

In order to provide operating system compatibility between the 80286 and 386 DX, the 386 DX supports all of the 80286 segment descriptors. Figure 4-9 shows the general format of an 80286 system segment descriptor. The only differences between 80286 and 386 DX descriptor formats are that the values of the type fields, and the limit and base address fields have been expanded for the 386 DX. The 80286 system segment descriptors contained a 24-bit base address and 16-bit limit, while the 386 DX system segment descriptors have a 32-bit base address, a 20-bit limit field, and a granularity bit.

By supporting 80286 system segments the 386 DX is able to execute 80286 application programs on a 386 DX operating system. This is possible because the processor automatically understands which descriptors are 80286-style descriptors and which descriptors are 386 DX-style descriptors. In particular, if the upper word of a descriptor is zero, then that descriptor is a 80286-style descriptor.

The only other differences between 80286-style descriptors and 386 DX descriptors is the interpretation of the word count field of call gates and the B bit. The word count field specifies the number of 16-bit quantities to copy for 80286 call gates and 32-bit quantities for 386 DX call gates. The B bit controls the size of PUSHes when using a call gate; if B=0 PUSHes are 16 bits, if B=1 PUSHes are 32 bits.

4.3.4.8 SELECTOR FIELDS

A selector in Protected Mode has three fields: Local or Global Descriptor Table Indicator (TI), Descriptor Entry Index (Index), and Requestor (the selector's) Privilege Level (RPL) as shown in Figure 4-10. The TI bits select one of two memory-based tables of descriptors (the Global Descriptor Table or the Local Descriptor Table). The Index selects one of 8K descriptors in the appropriate descriptor table. The RPL bits allow high speed testing of the selector's privilege attributes.

4.3.4.9 SEGMENT DESCRIPTOR CACHE

In addition to the selector value, every segment register has a segment descriptor cache register associated with it. Whenever a segment register's contents are changed, the 8-byte descriptor associated with that selector is automatically loaded (cached) on the chip. Once loaded, all references to that segment use the cached descriptor information instead of reaccessing the descriptor. The contents of the descriptor cache are not visible to the programmer. Since descriptor caches only change when a segment register is changed, programs which modify the descriptor tables must reload the appropriate segment registers after changing a descriptor's value.

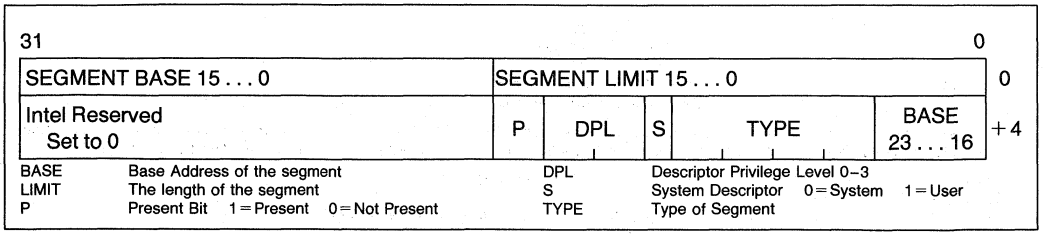


Figure 4-9. 80286 Code and Data Segment Descriptors

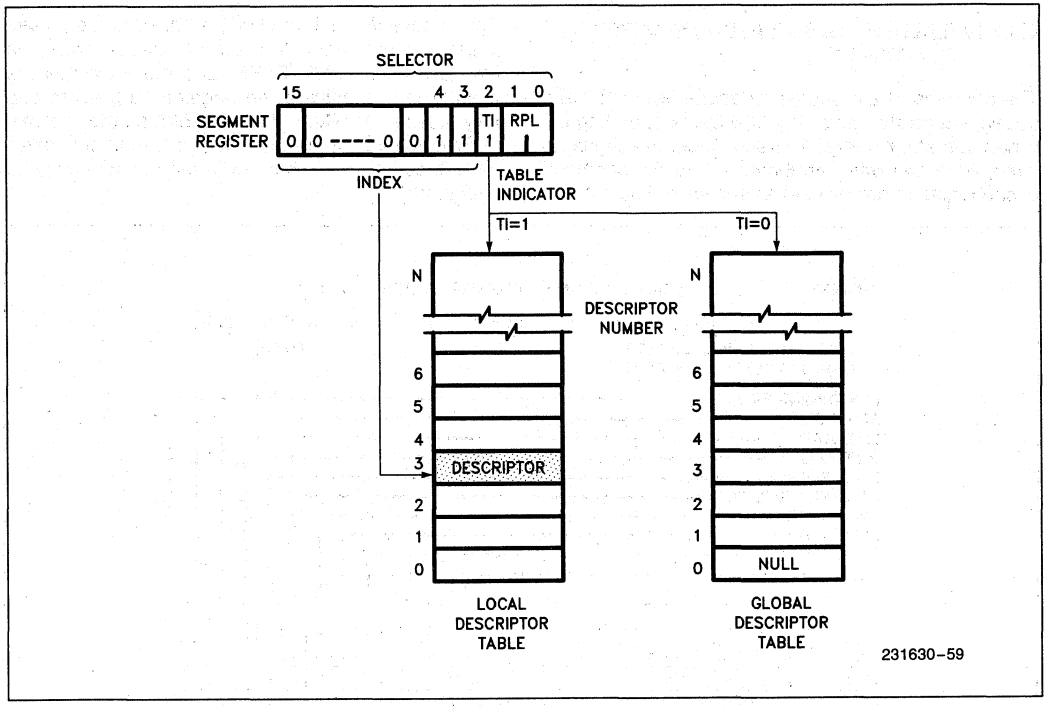


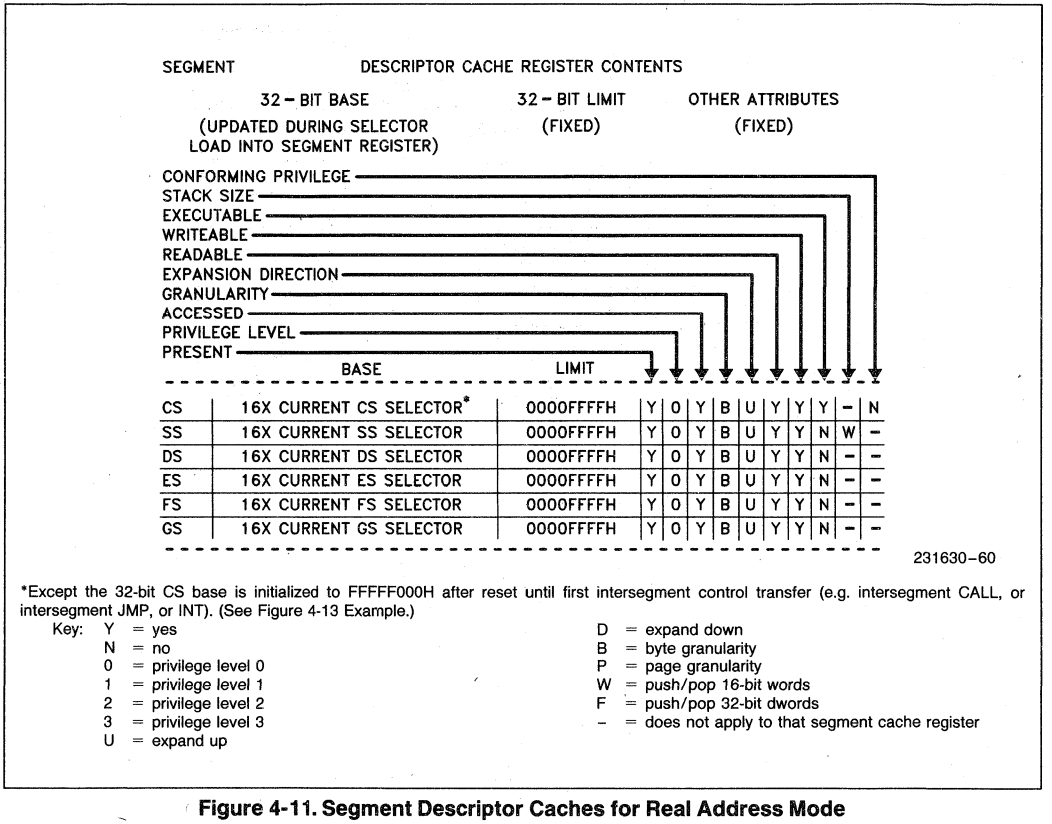
Figure 4-10. Example Descriptor Selection

5

4.3.4.10 SEGMENT DESCRIPTOR REGISTER SETTINGS

The contents of the segment descriptor cache vary depending on the mode the 386 DX is operating in. When operating in Real Address Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4-11.

For compatiblity with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. In Real Address Mode, the internal "privilege level" is always fixed to the highest level, level 0, so I/O and other privileged opcodes may be executed.

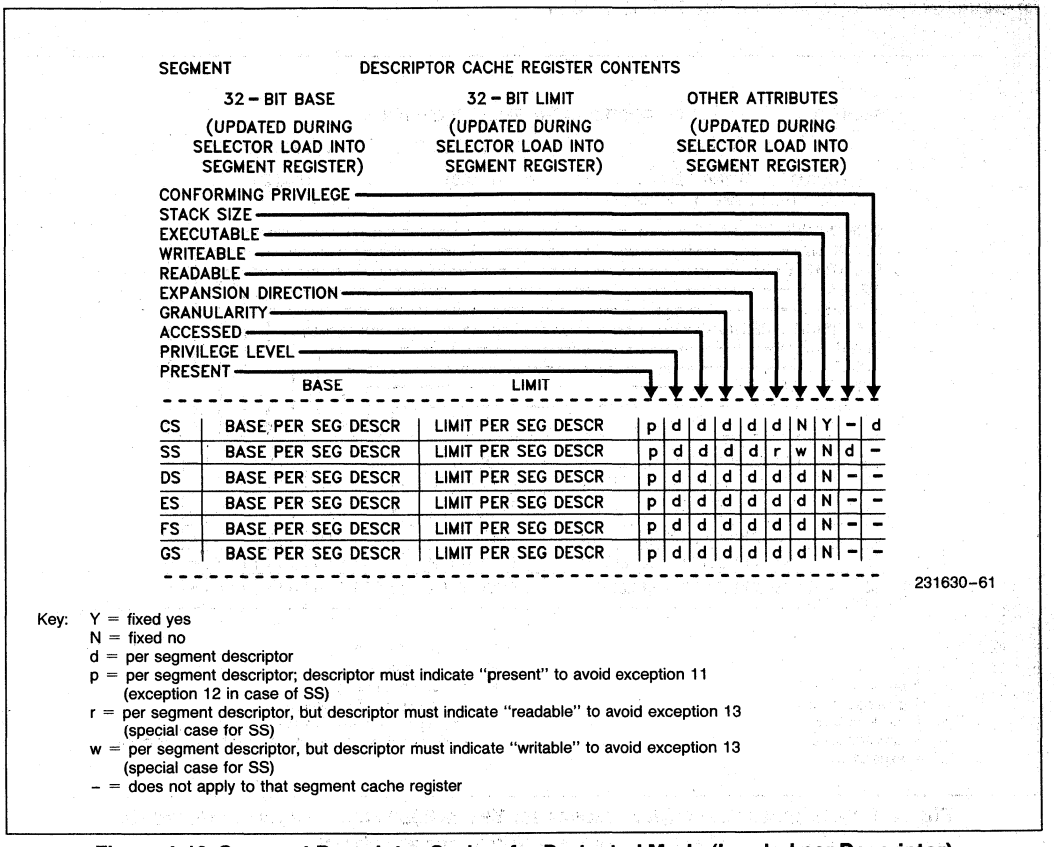


*Except the 32-bit CS base is initialized to FFFFF000H after reset until first intersegment control transfer (e.g. intersegment CALL, or intersegment JMP, or INT). (See Figure 4-13 Example.)

Key:
- Y = yes
- N = no
- 0 = privilege level 0
- 1 = privilege level 1
- 2 = privilege level 2
- 3 = privilege level 3
- U = expand up
- D = expand down
- B = byte granularity
- P = page granularity
- W = push/pop 16-bit words
- F = push/pop 32-bit dwords
- – = does not apply to that segment cache register

Figure 4-11. Segment Descriptor Caches for Real Address Mode (Segment Limit and Attributes are Fixed)

When operating in Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4-12. In Protected Mode, each of these fields are defined according to the contents of the segment descriptor indexed by the selector value loaded into the segment register.



Key: Y = fixed yes
N = fixed no
d = per segment descriptor
p = per segment descriptor; descriptor must indicate "present" to avoid exception 11 (exception 12 in case of SS)
r = per segment descriptor, but descriptor must indicate "readable" to avoid exception 13 (special case for SS)
w = per segment descriptor, but descriptor must indicate "writable" to avoid exception 13 (special case for SS)
– = does not apply to that segment cache register

Figure 4-12. Segment Descriptor Caches for Protected Mode (Loaded per Descriptor)

When operating in a Virtual 8086 Mode within the Protected Mode, the segment base, limit, and other attributes within the segment cache registers are defined as shown in Figure 4-13. For compatibility with the 8086 architecture, the base is set to sixteen times the current selector value, the limit is fixed at 0000FFFFH, and the attributes are fixed so as to indicate the segment is present and fully usable. The virtual program executes at lowest privilege level, level 3, to allow trapping of all IOPL-sensitive instructions and level-0-only instructions.

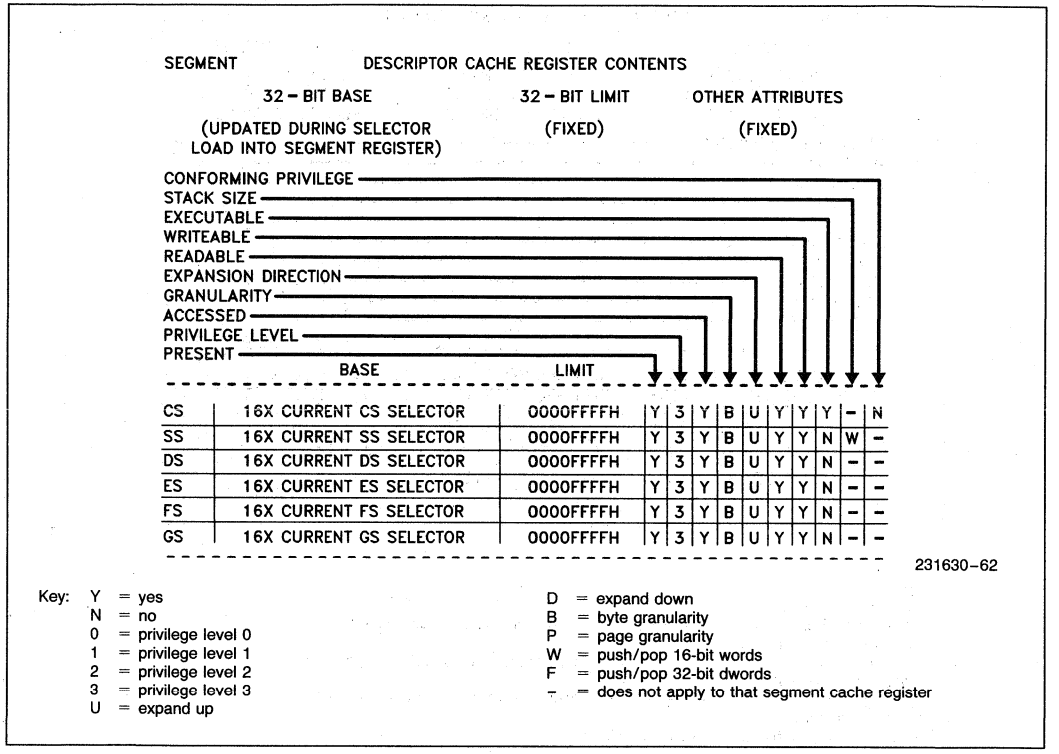


Figure 4-13. Segment Descriptor Caches for Virtual 8086 Mode within Protected Mode (Segment Limit and Attributes are Fixed)

4.4 PROTECTION

4.4.1 Protection Concepts

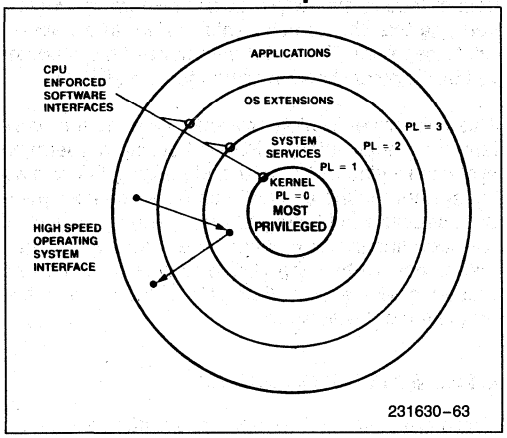


Figure 4-14. Four-Level Hierachical Protection

The 386 DX has four levels of protection which are optimized to support the needs of a multi-tasking operating system to isolate and protect user programs from each other and the operating system. The privilege levels control the use of privileged instructions, I/O instructions, and access to segments and segment descriptors. Unlike traditional microprocessor-based systems where this protection is achieved only through the use of complex external hardware and software the 386 DX provides the protection as part of its integrated Memory Management Unit. The 386 DX offers an additional type of protection on a page basis, when paging is enabled (See section 4.5.3 **Page Level Protection**).

The four-level hierarchical privilege system is illustrated in Figure 4-14. It is an extension of the user/supervisor privilege mode commonly used by minicomputers and, in fact, the user/supervisor mode is fully supported by the 386 DX paging mechanism. The privilege levels (PL) are numbered 0 through 3. Level 0 is the most privileged or trusted level.

4.4.2 Rules of Privilege

The 386 DX controls access to both data and procedures between levels of a task, according to the following rules.

- Data stored in a segment with privilege level **p** can be accessed only by code executing at a privilege level at least as privileged as **p**.
- A code segment/procedure with privilege level **p** can only be called by a task executing at the same or a lesser privilege level than **p**.

4.4.3 Privilege Levels

4.4.3.1 TASK PRIVILEGE

At any point in time, a task on the 386 DX always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies the task's privilege level. A task's CPL may only be changed by control transfers through gate descriptors to a code segment with a different privilege level. (See section 4.4.4 **Privilege Level Transfers**) Thus, an application program running at PL = 3 may call an operating system routine at PL = 1 (via a gate) which would cause the task's CPL to be set to 1 until the operating system routine was finished.

4.4.3.2 SELECTOR PRIVILEGE (RPL)

The privilege level of a selector is specified by the RPL field. The RPL is the two least significant bits of the selector. The selector's RPL is only used to establish a less trusted privilege level than the current privilege level for the use of a segment. This level is called the task's effective privilege level (EPL). The EPL is defined as being the least privileged (i.e. numerically larger) level of a task's CPL and a selector's RPL. Thus, if selector's RPL = 0 then the CPL always specifies the privilege level for making an access using the selector. On the other hand if RPL = 3 then a selector can only access segments at level 3 regardless of the task's CPL. The RPL is most commonly used to verify that pointers passed to an operating system procedure do not access data that is of higher privilege than the procedure that originated the pointer. Since the originator of a selector can specify any RPL value, the Adjust RPL (ARPL) instruction is provided to force the RPL bits to the originator's CPL.

4.4.3.3 I/O PRIVILEGE AND I/O PERMISSION BITMAP

The I/O privilege level (IOPL, a 2-bit field in the EFLAG register) defines the least privileged level at which I/O instructions can be unconditionally performed. I/O instructions can be unconditionally performed when CPL ≤ IOPL. (The I/O instructions are IN, OUT, INS, OUTS, REP INS, and REP OUTS.) When CPL > IOPL, and the current task is associated with a 286 TSS, attempted I/O instructions cause an exception 13 fault. When CPL > IOPL, and the current task is associated with a 386 DX TSS, the I/O Permission Bitmap (part of a 386 DX TSS) is consulted on whether I/O to the port is allowed, or an exception 13 fault is to be generated instead. For

diagrams of the I/O Permission Bitmap, refer to Figures 4-15a and 4-15b. For further information on how the I/O Permission Bitmap is used in Protected Mode or in Virtual 8086 Mode, refer to section 4.6.4 Protection and I/O Permission Bitmap.

The I/O privilege level (IOPL) also affects whether several other instructions can be executed or cause an exception 13 fault instead. These instructions are called "IOPL-sensitive" instructions and they are CLI and STI. (Note that the LOCK prefix is *not* IOPL-sensitive on the 386 DX.)

The IOPL also affects whether the IF (interrupts enable flag) bit can be changed by loading a value into the EFLAGS register. When CPL ≤ IOPL, then the IF bit can be changed by loading a new value into the EFLAGS register. When CPL > IOPL, the IF bit cannot be changed by a new value POP'ed into (or otherwise loaded into) the EFLAGS register; the IF bit merely remains unchanged and no exception is generated.

Table 4-2. Pointer Test Instructions

Instruction	Operands	Function
ARPL	Selector, Register	Adjust Requested Privilege Level: adjusts the RPL of the selector to the numeric maximum of current selector RPL value and the RPL value in the register. Set zero flag if selector RPL was changed.
VERR	Selector	VERify for Read: sets the zero flag if the segment referred to by the selector can be read.
VERW	Selector	VERify for Write: sets the zero flag if the segment referred to by the selector can be written.
LSL	Register, Selector	Load Segment Limit: reads the segment limit into the register if privilege rules and descriptor type allow. Set zero flag if successful.
LAR	Register, Selector	Load Access Rights: reads the descriptor access rights byte into the register if privilege rules allow. Set zero flag if successful.

4.4.3.4 PRIVILEGE VALIDATION

The 386 DX provides several instructions to speed pointer testing and help maintain system integrity by verifying that the selector value refers to an appropriate segment. Table 4-2 summarizes the selector validation procedures available for the 386 DX.

This pointer verification prevents the common problem of an application at PL = 3 calling a operating systems routine at PL = 0 and passing the operating system routine a "bad" pointer which corrupts a data structure belonging to the operating system. If the operating system routine uses the ARPL instruction to ensure that the RPL of the selector has no greater privilege than that of the caller, then this problem can be avoided.

4.4.3.5 DESCRIPTOR ACCESS

There are basically two types of segment accesses: those involving code segments such as control transfers, and those involving data accesses. Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL as described above.

Any time an instruction loads data segment registers (DS, ES, FS, GS) the 386 DX makes protection validation checks. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segments or readable code segments. The data access rules are specified in section 4.2.2 **Rules of Privilege**. The only exception to those rules is readable conforming code segments which can be accessed at any privilege level.

Finally the privilege validation checks are performed. The CPL is compared to the EPL and if the EPL is more privileged than the CPL an exception 13 (general protection fault) is generated.

The rules regarding the stack segment are slightly different than those involving data segments. Instructions that load selectors into SS must refer to data segment descriptors for writeable data segments. The DPL and RPL must equal the CPL. All other descriptor types or a privilege level violation will cause exception 13. A stack not present fault causes exception 12. Note that an exception 11 is used for a not-present code or data segment.

4.4.4 Privilege Level Transfers

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call

Table 4-3. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL	CALL	Call Gate	GDT/LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
	CALL, JMP	Task State Segment	GDT
Task Switch	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag register) = 0
**NT (Nested Task bit of flag register) = 1

or a jump to another routine. There are five types of control transfers which are summarized in Table 4-3. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only via control transfers, by using gates, task switches, and interrupt or trap gates.

Control transfers can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules will cause an exception 13 (e.g. JMP through a call gate, or IRET from a normal subroutine call).

In order to provide further system security, all control transfers are also subject to the privilege rules.

The privilege rules require that:

— Privilege level transitions can only occur via gates.

— JMPs can be made to a non-conforming code segment with the same privilege or to a conforming code segment with greater or equal privilege.

— CALLs can be made to a non-conforming code segment with the same privilege or via a gate to a more privileged level.

— Interrupts handled within the task obey the same privilege rules as CALLs.

— Conforming Code segments are accessible by privilege levels which are the same or less privileged than the conforming-code segment's DPL.

— Both the requested privilege level (RPL) in the selector pointing to the gate and the task's CPL must be of equal or greater privilege than the gate's DPL.

— The code segment selected in the gate must be the same or more privileged than the task's CPL.

— Return instructions that do not switch tasks can only return control to a code segment with same or less privilege.

— Task switches can be performed by a CALL, JMP, or INT which references either a task gate or task state segment who's DPL is less privileged or the same privilege as the old task's CPL.

Any control transfer that changes CPL within a task causes a change of stacks as a result of the privilege level change. The initial values of SS:ESP for privilege levels 0, 1, and 2 are retained in the task state segment (see section 4.4.6 **Task Switching**). During a JMP or CALL control transfer, the new stack pointer is loaded into the SS and ESP registers and the previous stack pointer is pushed onto the new stack.

When RETurning to the original privilege level, use of the lower-privileged stack is restored as part of the RET or IRET instruction operation. For subroutine calls that pass parameters on the stack and cross privilege levels, a fixed number of words (as specified in the gate's word count field) are copied from the previous stack to the current stack. The inter-segment RET instruction with a stack adjustment value will correctly restore the previous stack pointer upon return.

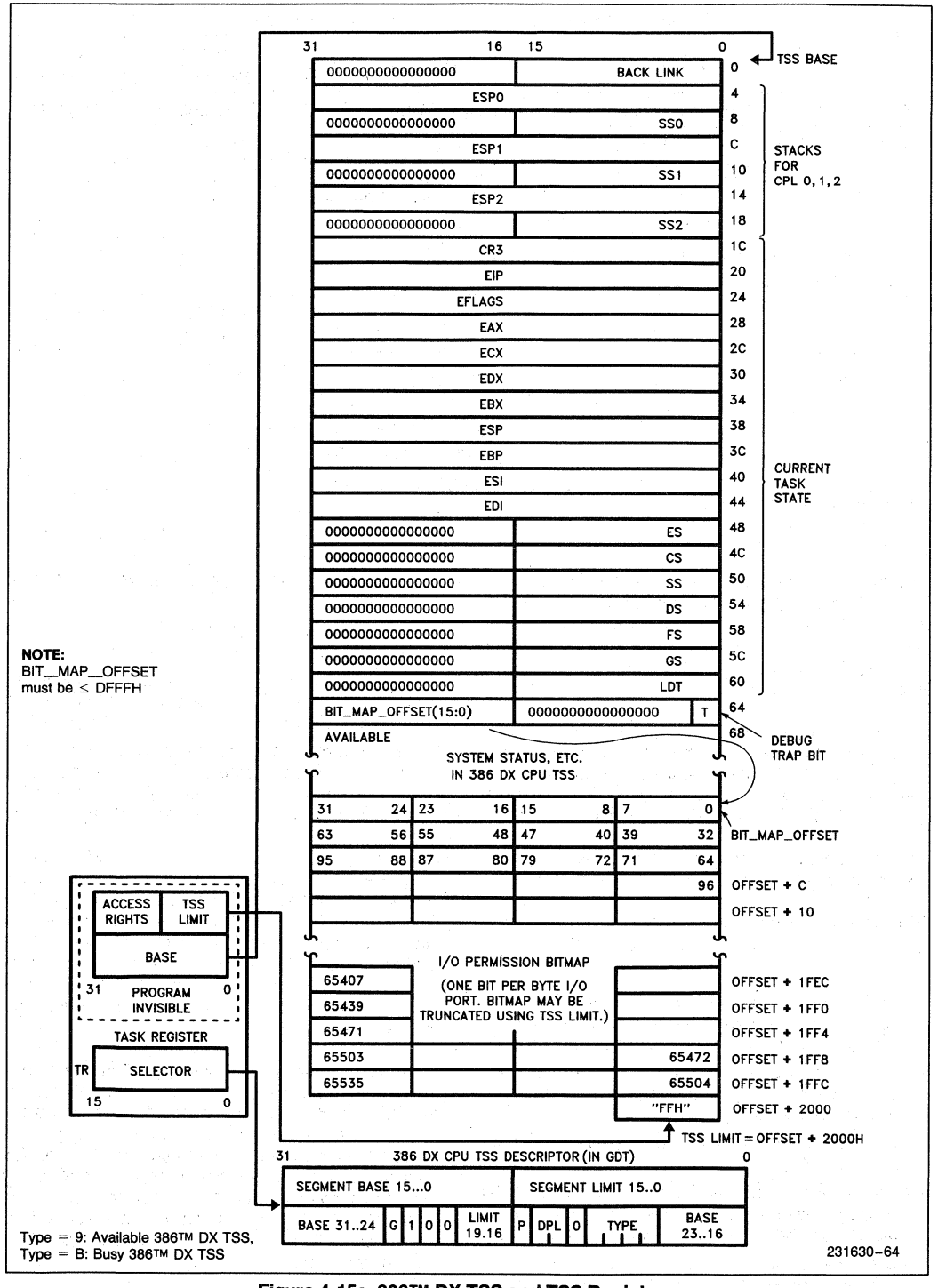


Figure 4-15a. 386™ DX TSS and TSS Registers

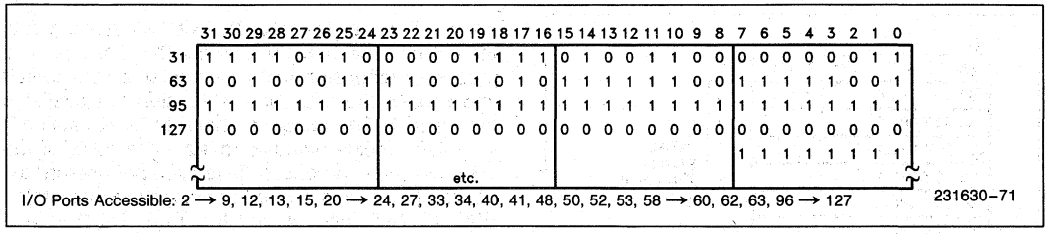


Figure 4-15b. Sample I/O Permission Bit Map

4.4.5 Call Gates

Gates provide protected, indirect CALLs. One of the major uses of gates is to provide a secure method of privilege transfers within a task. Since the operating system defines all of the gates in a system, it can ensure that all gates only allow entry into a few trusted procedures (such as those which allocate memory, or perform I/O).

Gate descriptors follow the data access rules of privilege; that is, gates can be accessed by a task if the EPL, is equal to or more privileged than the gate descriptor's DPL. Gates follow the control transfer rules of privilege and therefore may only transfer control to a more privileged level.

Call Gates are accessed via a CALL instruction and are syntactically identical to calling a normal subroutine. When an inter-level 386 DX call gate is activated, the following actions occur.

1. Load CS:EIP from gate check for validity
2. SS is pushed zero-extended to 32 bits
3. ESP is pushed
4. Copy Word Count 32-bit parameters from the old stack to the new stack
5. Push Return address on stack

The procedure is identical for 80286 Call gates, except that 16-bit parameters are copied and 16-bit registers are pushed.

Interrupt Gates and Trap gates work in a similar fashion as the call gates, except there is no copying of parameters. The only difference between Trap and Interrupt gates is that control transfers through an Interrupt gate disable further interrupts (i.e. the IF bit is set to 0), and Trap gates leave the interrupt status unchanged.

4.4.6 Task Switching

A very important attribute of any multi-tasking/multi-user operating systems is its ability to rapidly switch between tasks or processes. The 386 DX directly supports this operation by providing a task switch instruction in hardware. The 386 DX task switch operation saves the entire state of the machine (all of the registers, address space, and a link to the previous task), loads a new execution state, performs protection checks, and commences execution in the new task, in about 17 microseconds. Like transfer of control via gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap, or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot.

The TSS descriptor points to a segment (see Figure 4-15) containing the entire 386 DX execution state while a task gate descriptor contains a TSS selector. The 386 DX supports both 80286 and 386 DX style TSSs. Figure 4-16 shows a 80286 TSS. The limit of a 386 DX TSS must be greater than 0064H (002BH for a 80286 TSS), and can be as large as 4 Gigabytes. In the additional TSS space, the operating system is free to store additional information such as the reason the task is inactive, time the task has spent running, and open files belong to the task.

Each task must have a TSS associated with it. The current TSS is identified by a special register in the 386 DX called the Task State Segment Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR are loaded whenever TR is loaded with a new selector. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to the task which was interrupted. The current executing task's state is saved in the TSS and the old task state is restored from its TSS.

Several bits in the flag register and machine status word (CR0) give information about the state of a task which are useful to the operating system. The Nested Task (NT) (bit 14 in EFLAGS) controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular return; when NT = 1, IRET performs a task switch operation back to the previous task. The NT bit is set or reset in the following fashion:

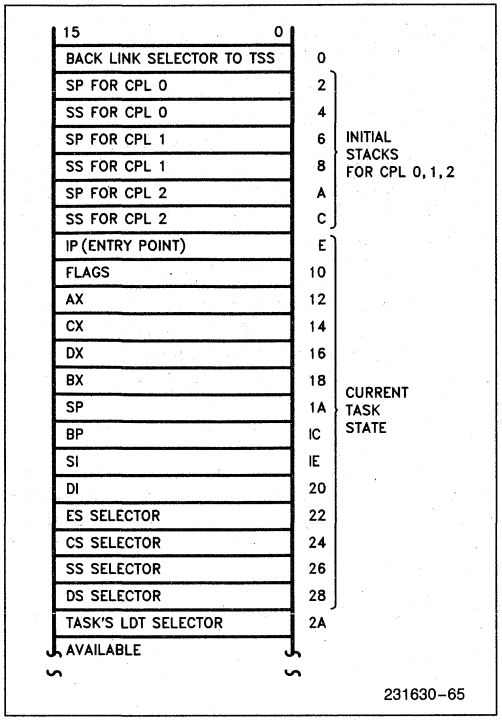


Figure 4-16. 80286 TSS

When a CALL or INT instruction initiates a task switch, the new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT. (The NT bit will be restored after execution of the interrupt handler) NT may also be set or cleared by POPF or IRET instructions.

The 386 DX task state segment is marked busy by changing the descriptor type field from TYPE 9H to TYPE BH. An 80286 TSS is marked busy by changing the descriptor type field from TYPE 1 to TYPE 3. Use of a selector that references a busy task state segment causes an exception 13.

The Virtual Mode (VM) bit 17 is used to indicate if a task, is a virtual 8086 task. If VM = 1, then the tasks will use the Real Mode addressing mechanism. The virtual 8086 environment is only entered and exited via a task switch (see section 4.6 **Virtual Mode**).

The coprocessor's state is not automatically saved when a task switch occurs, because the incoming task may not use the coprocessor. The Task Switched (TS) Bit (bit 3 in the CR0) helps deal with the coprocessor's state in a multi-tasking environment. Whenever the 386 DX switches tasks, it sets the TS bit. The 386 DX detects the first use of a processor extension instruction after a task switch and causes the processor extension not available exception 7. The exception handler for exception 7 may then decide whether to save the state of the coprocessor. A processor extension not present exception (7) will occur when attempting to execute an ESC or WAIT instruction if the Task Switched and Monitor coprocessor extension bits are both set (i.e. TS = 1 and MP = 1).

The **T** bit in the 386 DX TSS indicates that the processor should generate a debug exception when switching to a task. If T = 1 then upon entry to a new task a debug exception 1 will be generated.

4.4.7 Initialization and Transition to Protected Mode

Since the 386 DX begins executing in Real Mode immediately after RESET it is necessary to initialize the system tables and registers with the appropriate values.

The GDT and IDT registers must refer to a valid GDT and IDT. The IDT should be at least 256 bytes long, and GDT must contain descriptors for the initial code, and data segments. Figure 4-17 shows the tables and Figure 4-18 the descriptors needed for a simple Protected Mode 386 DX system. It has a single code and single data/stack segment each four gigabytes long and a single privilege level PL = 0.

The actual method of enabling Protected Mode is to load CR0 with the PE bit set, via the MOV CR0, R/M instruction. This puts the 386 DX in Protected Mode.

After enabling Protected Mode, the next instruction should execute an intersegment JMP to load the CS register and flush the instruction decode queue. The final step is to load all of the data segment registers with the initial selector values.

An alternate approach to entering Protected Mode which is especially appropriate for multi-tasking operating systems, is to use the built in task-switch to load all of the registers. In this case the GDT would contain two TSS descriptors in addition to the code and data descriptors needed for the first task. The first JMP instruction in Protected Mode would jump to the TSS causing a task switch and loading all of the registers with the values stored in the TSS. The Task State Segment Register should be initialized to point to a valid TSS descriptor since a task switch saves the state of the current task in a task state segment.

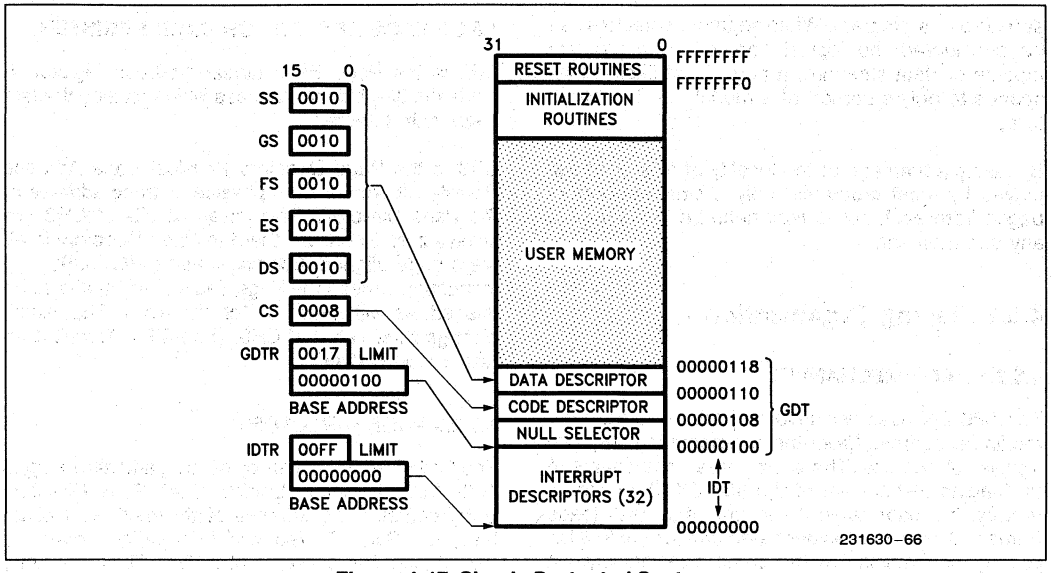


Figure 4-17. Simple Protected System

DATA DESCRIPTOR	SEGMENT BASE 15 . . . 0 0118 (H)								SEGMENT LIMIT 15 . . . 0 FFFF (H)										
	BASE 31 . . . 24 00 (H)		G 1	D 1	0	0	LIMIT 19.16 F (H)		1	0	0	1	0	0	1	0	BASE 23 . . . 16 00 (H)		
CODE DESCRIPTOR	SEGMENT BASE 15 . . . 0 0118 (H)								SEGMENT LIMIT 15 . . . 0 FFFF (H)										
	BASE 31 . . . 24 00 (H)		G 1	D 1	0	0	LIMIT 19.16 F (H)		1	0	0	1	1	0	1	0	BASE 23 . . . 16 00 (H)		
								NULL	DESCRIPTOR										
	31					24		16	15					8					0

Figure 4-18. GDT Descriptors for Simple System

4.4.8 Tools for Building Protected Systems

In order to simplify the design of a protected multitasking system, Intel provides a tool which allows the system designer an easy method of constructing the data structures needed for a Protected Mode 386 DX system. This tool is the builder BLD-386™. BLD-386 lets the operating system writer specify all of the segment descriptors discussed in the previous sections (LDTs, IDTs, GDTs, Gates, and TSSs) in a high-level language.

4.5 PAGING

4.5.1 Paging Concepts

Paging is another type of memory management useful for virtual memory multitasking operating systems. Unlike segmentation which modularizes programs and data into variable length segments, paging divides programs into multiple uniform size pages. Pages bear no direct relation to the logical

structure of a program. While segment selectors can be considered the logical "name" of a program module or data structure, a page most likely corresponds to only a portion of a module or data structure.

By taking advantage of the locality of reference displayed by most programs, only a small number of pages from each active task need be in memory at any one moment.

4.5.2 Paging Organization

4.5.2.1 PAGE MECHANISM

The 386 DX uses two levels of tables to translate the linear address (from the segmentation unit) into a physical address. There are three components to the paging mechanism of the 386 DX: the page directory, the page tables, and the page itself (page frame). All memory-resident elements of the 386 DX paging mechanism are the same size, namely, 4K bytes. A uniform size for all of the elements simplifies memory allocation and reallocation schemes, since there is no problem with memory fragmentation. Figure 4-19 shows how the paging mechanism works.

4.5.2.2 PAGE DESCRIPTOR BASE REGISTER

CR2 is the Page Fault Linear Address register. It holds the 32-bit linear address which caused the last page fault detected.

CR3 is the Page Directory Physical Base Address Register. It contains the physical starting address of the Page Directory. The lower 12 bits of CR3 are always zero to ensure that the Page Directory is always page aligned. Loading it via a MOV CR3, reg instruction causes the Page Table Entry cache to be flushed, as will a task switch through a TSS which **changes** the value of CR0. (See 4.5.4 **Translation Lookaside Buffer**).

4.5.2.3 PAGE DIRECTORY

The Page Directory is 4K bytes long and allows up to 1024 Page Directory Entries. Each Page Directory Entry contains the address of the next level of tables, the Page Tables and information about the page table. The contents of a Page Directory Entry are shown in Figure 4-20. The upper 10 bits of the linear address (A22–A31) are used as an index to select the correct Page Directory Entry.

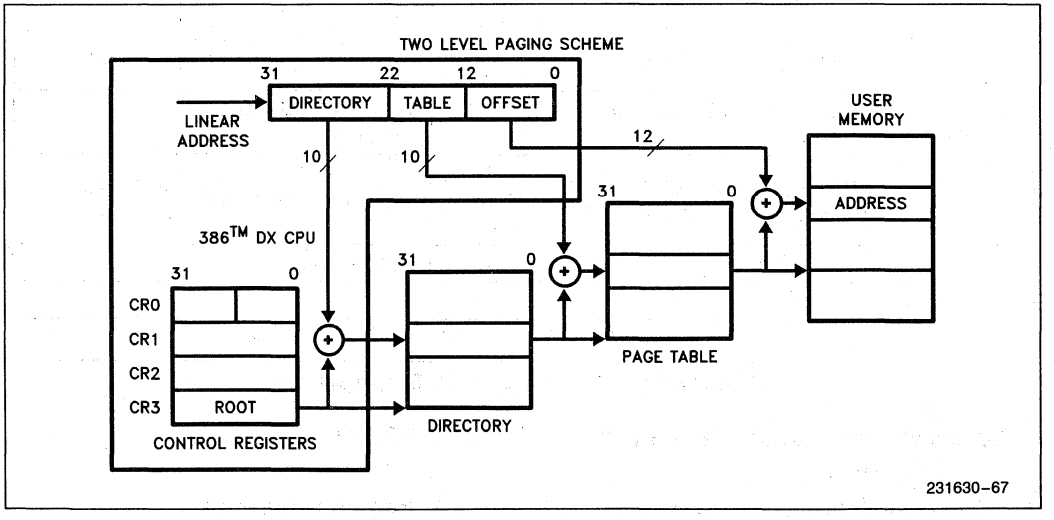


Figure 4-19. Paging Mechanism

31 — 12	11 — 10	9	8	7	6	5	4	3	2	1	0
PAGE TABLE ADDRESS 31..12	OS RESERVED		0	0	D	A	0	0	U/S	R/W	P

Figure 4-20. Page Directory Entry (Points to Page Table)

31 12	11 10	9	8	7	6	5	4	3	2	1	0
PAGE FRAME ADDRESS 31..12	OS RESERVED		0	0	D	A	0	0	U — S	R — W	P

Figure 4-21. Page Table Entry (Points to Page)

4.5.2.4 PAGE TABLES

Each Page Table is 4K bytes and holds up to 1024 Page Table Entries. Page Table Entries contain the starting address of the page frame and statistical information about the page (see Figure 4-21). Address bits A12–A21 are used as an index to select one of the 1024 Page Table Entries. The 20 upper-bit page frame address is concatenated with the lower 12 bits of the linear address to form the physical address. Page tables can be shared between tasks and swapped to disks.

4.5.2.5 PAGE DIRECTORY/TABLE ENTRIES

The lower 12 bits of the Page Table Entries and Page Directory Entries contain statistical information about pages and page tables respectively. The **P** (Present) bit 0 indicates if a Page Directory or Page Table entry can be used in address translation. If P = 1 the entry can be used for address translation; if P = 0 the entry can not be used for translation. Note that the present bit of the page table entry that points to the page where code is currently being executed should always be set. Code that marks its own page not present should not be written. All of the other bits are available for use by the software. For example the remaining 31 bits could be used to indicate where on the disk the page is stored.

The **A** (Accessed) bit 5, is set by the 386 DX for both types of entries before a read or write access occurs to an address covered by the entry. The **D** (Dirty) bit 6 is set to 1 before a write to an address covered by that page table entry occurs. The D bit is undefined for Page Directory Entries. When the P, A and D bits are updated by the 386 DX, the processor generates a Read-Modify-Write cycle which locks the bus and prevents conflicts with other processors or perpherials. Software which modifies these bits should use the LOCK prefix to ensure the integrity of the page tables in multi-master systems.

The 3 bits marked **OS Reserved** in Figure 4-20 and Figure 4-21 (bits 9–11) are software definable. OSs are free to use these bits for whatever purpose they wish. An example use of the **OS Reserved** bits would be to store information about page aging. By keeping track of how long a page has been in memory since being accessed, an operating system can implement a page replacement algorithm like Least Recently Used.

The (User/Supervisor) U/S bit 2 and the (Read/Write) R/W bit 1 are used to provide protection attributes for individual pages.

4.5.3 Page Level Protection (R/W, U/S Bits)

The 386 DX provides a set of protection attributes for paging systems. The paging mechanism distinguishes between two levels of protection: User which corresponds to level 3 of the segmentation based protection, and supervisor which encompasses all of the other protection levels (0, 1, 2). Programs executing at Level 0, 1 or 2 bypass the page protection, although segmentation based protection is still enforced by the hardware.

The U/S and R/W bits are used to provide User/Supervisor and Read/Write protection for individual pages or for all pages covered by a Page Table Directory Entry. The U/S and R/W bits in the first level Page Directory Table apply to all pages described by the page table pointed to by that directory entry. The U/S and R/W bits in the second level Page Table Entry apply only to the page described by that entry. The U/S and R/W bits for a given page are obtained by taking the most restrictive of the U/S and R/W from the Page Directory Table Entries and the Page Table Entries and using these bits to address the page.

Example: If the U/S and R/W bits for the Page Directory entry were 10 and the U/S and R/W bits for the Page Table Entry were 01, the access rights for the page would be 01, the numerically smaller of the two. Table 4-4 shows the affect of the U/S and R/W bits on accessing memory.

Table 4-4. Protection Provided by R/W and U/S

U/S	R/W	Permitted Level 3	Permitted Access Levels 0, 1, or 2
0	0	None	Read/Write
0	1	None	Read/Write
1	0	Read-Only	Read/Write
1	1	Read/Write	Read/Write

However a given segment can be easily made read-only for level 0, 1, or 2 via the use of segmented protection mechanisms. (Section 4.4 **Protection**).

4.5.4 Translation Lookaside Buffer

The 386 DX paging hardware is designed to support demand paged virtual memory systems. However, performance would degrade substantially if the processor was required to access two levels of tables for every memory reference. To solve this problem, the 386 DX keeps a cache of the most recently accessed pages, this cache is called the Translation Lookaside Buffer (TLB). The TLB is a four-way set associative 32-entry page table cache. It automatically keeps the most commonly used Page Table Entries in the processor. The 32-entry TLB coupled with a 4K page size, results in coverage of 128K bytes of memory addresses. For many common multi-tasking systems, the TLB will have a hit rate of about 98%. This means that the processor will only have to access the two-level page structure on 2% of all memory references. Figure 4-22 illustrates how the TLB complements the 386 DX's paging mechanism.

4.5.5 Paging Operation

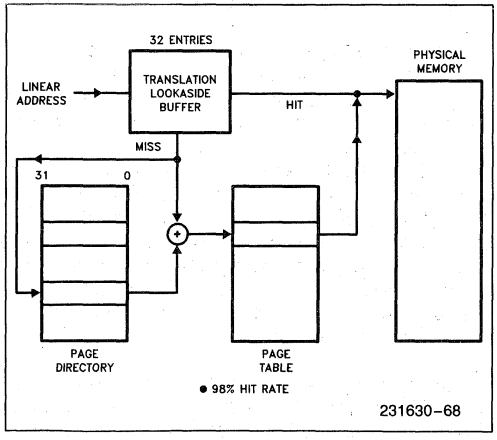


Figure 4-22. Translation Lookaside Buffer

The paging hardware operates in the following fashion. The paging unit hardware receives a 32-bit linear address from the segmentation unit. The upper 20 linear address bits are compared with all 32 entries in the TLB to determine if there is a match. If there is a match (i.e. a TLB hit), then the 32-bit physical address is calculated and will be placed on the address bus.

However, if the page table entry is not in the TLB, the 386 DX will read the appropriate Page Directory Entry. If P = 1 on the Page Directory Entry indicating that the page table is in memory, then the 386 DX will read the appropriate Page Table Entry and set the Access bit. If P = 1 on the Page Table Entry indicating that the page is in memory, the 386 DX will update the Access and Dirty bits as needed and fetch the operand. The upper 20 bits of the linear address, read from the page table, will be stored in the TLB for future accesses. However, if P = 0 for either the Page Directory Entry or the Page Table Entry, then the processor will generate a page fault, an Exception 14.

The processor will also generate an exception 14, page fault, if the memory reference violated the page protection attributes (i.e. U/S or R/W) (e.g. trying to write to a read-only page). CR2 will hold the linear address which caused the page fault. If a second page fault occurs, while the processor is attempting to enter the service routine for the first, then the processor will invoke the page fault (exception 14) handler a second time, rather than the double fault (exception 8) handler. Since Exception 14 is classified as a fault, CS: EIP will point to the instruction causing the page fault. The 16-bit error code pushed as part of the page fault handler will contain status bits which indicate the cause of the page fault.

The 16-bit error code is used by the operating system to determine how to handle the page fault Figure 4-23A shows the format of the page-fault error code and the interpretation of the bits.

NOTE:
Even though the bits in the error code (U/S, W/R, and P) have similar names as the bits in the Page Directory/Table Entries, the interpretation of the error code bits is different. Figure 4-23B indicates what type of access caused the page fault.

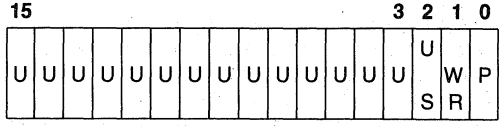


Figure 4-23A. Page Fault Error Code Format

U/S: The U/S bit indicates whether the access causing the fault occurred when the processor was executing in User Mode (U/S = 1) or in Supervisor mode (U/S = 0)

W/R: The W/R bit indicates whether the access causing the fault was a Read (W/R = 0) or a Write (W/R = 1)

P: The P bit indicates whether a page fault was caused by a not-present page (P = 0), or by a page level protection violation (P = 1)

U: UNDEFINED

U/S	W/R	Access Type
0	0	Supervisor* Read
0	1	Supervisor Write
1	0	User Read
1	1	User Write

*Descriptor table access will fault with U/S = 0, even if the program is executing at level 3.

Figure 4-23B. Type of Access Causing Page Fault

4.5.6 Operating System Responsibilities

The 386 DX takes care of the page address translation process, relieving the burden from an operating system in a demand-paged system. The operating system is responsible for setting up the initial page tables, and handling any page faults. The operating system also is required to invalidate (i.e. flush) the TLB when any changes are made to any of the page table entries. The operating system must reload CR3 to cause the TLB to be flushed.

Setting up the tables is simply a matter of loading CR3 with the address of the Page Directory, and allocating space for the Page Directory and the Page Tables. The primary responsibility of the operating system is to implement a swapping policy and handle all of the page faults.

A final concern of the operating system is to ensure that the TLB cache matches the information in the paging tables. In particular, any time the operating system sets the P present bit of page table entry to zero, the TLB must be flushed. Operating systems may want to take advantage of the fact that CR3 is stored as part of a TSS, to give every task or group of tasks its own set of page tables.

4.6 VIRTUAL 8086 ENVIRONMENT

4.6.1 Executing 8086 Programs

The 386 DX allows the execution of 8086 application programs in both Real Mode and in the Virtual 8086 Mode (Virtual Mode). Of the two methods, Virtual 8086 Mode offers the system designer the most flexibility. The Virtual 8086 Mode allows the execution of 8086 applications, while still allowing the system designer to take full advantage of the 386 DX protection mechanism. In particular, the 386 DX allows the simultaneous execution of 8086 operating systems and its applications, and a 386 DX operating system and both 80286 and 386 DX applications. Thus, in a multi-user 386 DX computer, one person could be running an MS-DOS spreadsheet, another person using MS-DOS, and a third person could be running multiple Unix utilities and applications. Each person in this scenario would believe that he had the computer completely to himself. Figure 4-24 illustrates this concept.

4.6.2 Virtual 8086 Mode Addressing Mechanism

One of the major differences between 386 DX Real and Protected modes is how the segment selectors are interpreted. When the processor is executing in Virtual 8086 Mode the segment registers are used in an identical fashion to Real Mode. The contents of the segment register is shifted left 4 bits and added to the offset to form the segment base linear address.

The 386 DX allows the operating system to specify which programs use the 8086 style address mechanism, and which programs use Protected Mode addressing, on a per task basis. Through the use of paging, the one megabyte address space of the Virtual Mode task can be mapped to anywhere in the 4 gigabyte linear address space of the 386 DX. Like Real Mode, Virtual Mode effective addresses (i.e., segment offsets) that exceed 64K byte will cause an exception 13. However, these restrictions should not prove to be important, because most tasks running in Virtual 8086 Mode will simply be existing 8086 application programs.

4.6.3 Paging In Virtual Mode

The paging hardware allows the concurrent running of multiple Virtual Mode tasks, and provides protection and operating system isolation. Although it is not strictly necessary to have the paging hardware enabled to run Virtual Mode tasks, it is needed in order to run multiple Virtual Mode tasks or to relocate the address space of a Virtual Mode task to physical address space greater than one megabyte.

The paging hardware allows the 20-bit linear address produced by a Virtual Mode program to be divided into up to 256 pages. Each one of the pages can be located anywhere within the maximum 4 gigabyte physical address space of the 386 DX. In addition, since CR3 (the Page Directory Base Register) is loaded by a task switch, each Virtual Mode task can use a different mapping scheme to map pages to different physical locations. Finally, the paging hardware allows the sharing of the 8086 operating

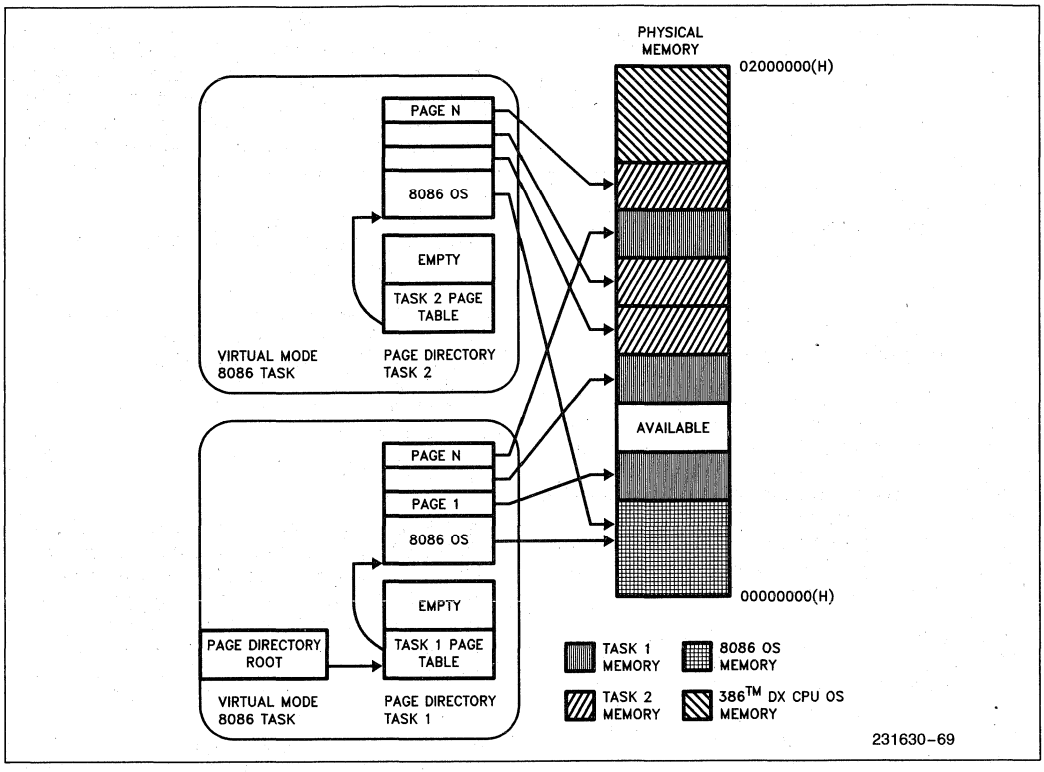


Figure 4-24. Virtual 8086 Environment Memory Management

system code between multiple 8086 applications. Figure 4-24 shows how the 386 DX paging hardware enables multiple 8086 programs to run under a virtual memory demand paged system.

4.6.4 Protection and I/O Permission Bitmap

All Virtual 8086 Mode programs execute at privilege level 3, the level of least privilege. As such, Virtual 8086 Mode programs are subject to all of the protection checks defined in Protected Mode. (This is different from Real Mode which implicitly is executing at privilege level 0, the level of greatest privilege.) Thus, an attempt to execute a privileged instruction when in Virtual 8086 Mode will cause an exception 13 fault.

The following are privileged instructions, which may be executed only at Privilege Level 0. Therefore, attempting to execute these instructions in Virtual 8086 Mode (or anytime CPL > 0) causes an exception 13 fault:

```
LIDT;    MOV DRn,reg;    MOV reg,DRn;
LGDT;    MOV TRn,reg;    MOV reg,TRn;
LMSW;    MOV CRn,reg;    MOV reg,CRn.
CLTS;
HLT;
```

Several instructions, particularly those applying to the multitasking model and protection model, are available only in Protected Mode. Therefore, attempting to execute the following instructions in Real Mode or in Virtual 8086 Mode generates an exception 6 fault:

```
LTR;     STR;
LLDT;    SLDT;
LAR;     VERR;
LSL;     VERW;
ARPL.
```

The instructions which are IOPL-sensitive in Protected Mode are:

```
IN;          STI;
OUT;         CLI
INS;
OUTS;
REP INS;
REP OUTS;
```

In Virtual 8086 Mode, a slightly different set of instructions are made IOPL-sensitive. The following instructions are IOPL-sensitive in Virtual 8086 Mode:

```
INT n;    STI;
PUSHF;    CLI;
POPF;     IRET
```

The PUSHF, POPF, and IRET instructions are IOPL-sensitive in Virtual 8086 Mode only. This provision allows the IF flag (interrupt enable flag) to be virtualized to the Virtual 8086 Mode program. The INT n software interrupt instruction is also IOPL-sensitive in Virtual 8086 Mode. Note, however, that the INT 3 (opcode 0CCH), INTO, and BOUND instructions are not IOPL-sensitive in Virtual 8086 mode (they aren't IOPL sensitive in Protected Mode either).

Note that the I/O instructions (IN, OUT, INS, OUTS, REP INS, and REP OUTS) are **not** IOPL-sensitive in Virtual 8086 mode. Rather, the I/O instructions become automatically sensitive to the **I/O Permission Bitmap** contained in the **386 DX Task State Segment**. The I/O Permission Bitmap, automatically used by the 386 DX in Virtual 8086 Mode, is illustrated by Figures 4.15a and 4-15b.

The I/O Permission Bitmap can be viewed as a 0–64 Kbit bit string, which begins in memory at offset Bit_Map_Offset in the current TSS. Bit_Map_Offset must be ≤ DFFFH so the entire bit map and the byte FFH which follows the bit map are all at offsets ≤ FFFFH from the TSS base. The 16-bit pointer Bit_Map_Offset (15:0) is found in the word beginning at offset 66H (102 decimal) from the TSS base, as shown in Figure 4-15a.

Each bit in the I/O Permission Bitmap corresponds to a single byte-wide I/O port, as illustrated in Figure 4-15a. If a bit is 0, I/O to the corresponding byte-wide port can occur without generating an exception. Otherwise the I/O instruction causes an exception 13 fault. Since every byte-wide I/O port must be protectable, all bits corresponding to a word-wide or dword-wide port must be 0 for the word-wide or dword-wide I/O to be permitted. If all the referenced bits are 0, the I/O will be allowed. If any referenced bits are 1, the attempted I/O will cause an exception 13 fault.

Due to the use of a pointer to the base of the I/O Permission Bitmap, the bitmap may be located anywhere within the TSS, or may be ignored completely by pointing the Bit_Map_Offset (15:0) beyond the limit of the TSS segment. In the same manner, only a small portion of the 64K I/O space need have an associated map bit, by adjusting the TSS limit to truncate the bitmap. This eliminates the commitment of 8K of memory when a complete bitmap is not required, while allowing the fully general case if desired.

EXAMPLE OF BITMAP FOR I/O PORTS 0–255: Setting the TSS limit to {bit_Map_Offset + 31 +1**} [** see note below] will allow a 32-byte bitmap for the I/O ports #0–255, plus a terminator byte of all 1's [** see note below]. This allows the I/O bitmap to control I/O Permission to I/O port 0–255 while causing an exception 13 fault on attempted I/O to any I/O port 80256 through 65,565.

****IMPORTANT IMPLEMENTATION NOTE:** Beyond the last byte of I/O mapping information in the I/O Permission Bitmap **must** be a byte containing all 1's. The byte of all 1's must be within the limit of the 386 DX TSS segment (see Figure 4-15a).

4.6.5 Interrupt Handling

In order to fully support the emulation of an 8086 machine, interrupts in Virtual 8086 Mode are handled in a unique fashion. When running in Virtual Mode all interrupts and exceptions involve a privilege change back to the host 386 DX operating system. The 386 DX operating system determines if the interrupt comes from a Protected Mode application or from a Virtual Mode program by examining the VM bit in the EFLAGS image stored on the stack.

When a Virtual Mode program is interrupted and execution passes to the interrupt routine at level 0, the VM bit is cleared. However, the VM bit is still set in the EFLAG image on the stack.

The 386 DX operating system in turn handles the exception or interrupt and then returns control to the 8086 program. The 386 DX operating system may choose to let the 8086 operating system handle the interrupt or it may emulate the function of the interrupt handler. For example, many 8086 operating system calls are accessed by PUSHing parameters on the stack, and then executing an INT n instruction. If the IOPL is set to 0 then all INT n instructions will be intercepted by the 386 DX Microprocessor operating system. The 386 DX operating system could emulate the 8086 operating system's call. Figure 4-25 shows how the 386 DX operating system could intercept an 8086 operating system's call to "Open a File".

A 386 DX operating system can provide a Virtual 8086 Environment which is totally transparent to the application software via intercepting and then emulating 8086 operating system's calls, and intercepting IN and OUT instructions.

4.6.6 Entering and Leaving Virtual 8086 Mode

Virtual 8086 mode is entered by executing an IRET instruction (at CPL = 0), or Task Switch (at any CPL) to a 386 DX task whose 386 DX TSS has a FLAGS image containing a 1 in the VM bit position while the processor is executing in Protected Mode. That is, one way to enter Virtual 8086 mode is to switch to a task with a 386 DX TSS that has a 1 in the VM bit in the EFLAGS image. The other way is to execute a 32-bit IRET instruction at privilege level 0, where the stack has a 1 in the VM bit in the EFLAGS image. POPF does not affect the VM bit, even if the processor is in Protected Mode or level 0, and so cannot be used to enter Virtual 8086 Mode. PUSHF always pushes a 0 in the VM bit, even if the processor is in Virtual 8086 Mode, so that a program cannot tell if it is executing in REAL mode, or in Virtual 8086 mode.

The VM bit can be set by executing an IRET instruction only at privilege level 0, or by any instruction or Interrupt which causes a task switch in Protected Mode (with VM = 1 in the new FLAGS image), and can be cleared only by an interrupt or exception in Virtual 8086 Mode. IRET and POPF instructions executed in REAL mode or Virtual 8086 mode will not change the value in the VM bit.

The transition out of virtual 8086 mode to 386 DX protected mode occurs only on receipt of an interrupt or exception (such as due to a sensitive instruction). In Virtual 8086 mode, all interrupts and exceptions vector through the protected mode IDT, and enter an interrupt handler in protected 386 DX mode. That is, as part of interrupt processing, the VM bit is cleared.

Because the matching IRET must occur from level 0, if an Interrupt or Trap Gate is used to field an interrupt or exception out of Virtual 8086 mode, the Gate must perform an inter-level interrupt only to level 0. Interrupt or Trap Gates through conforming segments, or through segments with DPL > 0, will raise a GP fault with the CS selector as the error code.

4.6.6.1 TASK SWITCHES TO/FROM VIRTUAL 8086 MODE

Tasks which can execute in virtual 8086 mode must be described by a TSS with the new 386 DX format (TYPE 9 or 11 descriptor).

A task switch out of virtual 8086 mode will operate exactly the same as any other task switch out of a task with a 386 DX TSS. All of the programmer visible state, including the FLAGS register with the VM bit set to 1, is stored in the TSS. The segment registers in the TSS will contain 8086 segment base values rather than selectors.

A task switch into a task described by a 386 DX TSS will have an additional check to determine if the incoming task should be resumed in virtual 8086 mode. Tasks described by 80286 format TSSs cannot be resumed in virtual 8086 mode, so no check is required there (the FLAGS image in 80286 format TSS has only the low order 16 FLAGS bits). Before loading the segment register images from a 386 DX TSS, the FLAGS image is loaded, so that the segment registers are loaded from the TSS image as 8086 segment base values. The task is now ready to resume in virtual 8086 execution mode.

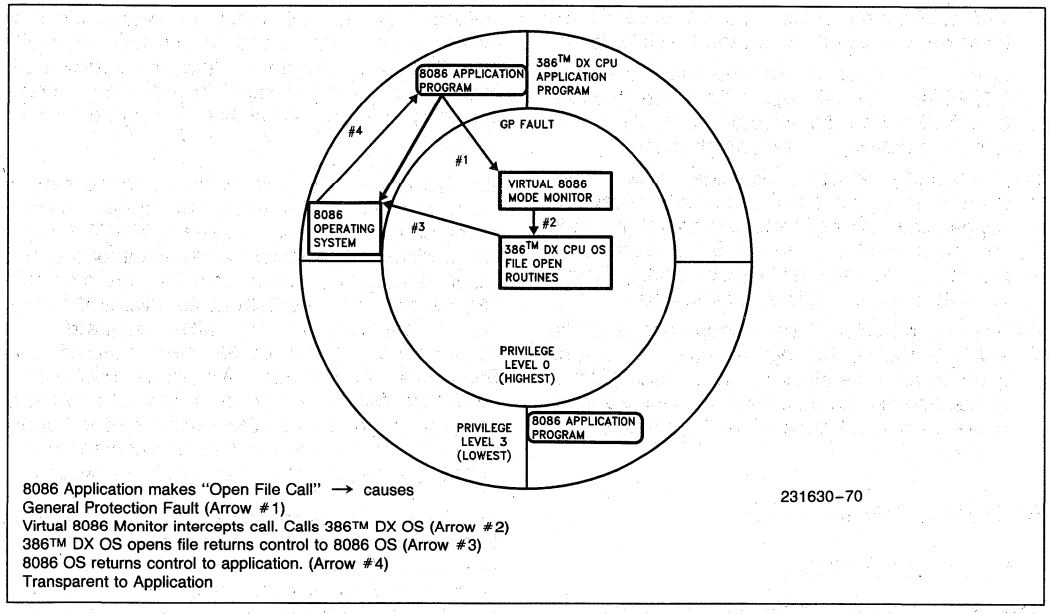
4.6.6.2 TRANSITIONS THROUGH TRAP AND INTERRUPT GATES, AND IRET

A task switch is one way to enter or exit virtual 8086 mode. The other method is to exit through a Trap or Interrupt gate, as part of handling an interrupt, and to enter as part of executing an IRET instruction. The transition out must use a 386 DX Trap Gate (Type 14), or 386 DX Interrupt Gate (Type 15), which must point to a non-conforming level 0 segment (DPL = 0) in order to permit the trap handler to IRET back to the Virtual 8086 program. The Gate must point to a non-conforming level 0 segment to perform a level switch to level 0 so that the matching IRET can change the VM bit. 386 DX gates must be used, since 80286 gates save only the low 16 bits of the FLAGS register, so that the VM bit will not be saved on transitions through the 80286 gates. Also, the 16-bit IRET (presumably) used to terminate the 80286 interrupt handler will pop only the lower 16 bits from FLAGS, and will not affect the VM bit. The action taken for a 386 DX Trap or Interrupt gate if an interrupt occurs while the task is executing in virtual 8086 mode is given by the following sequence.

(1) Save the FLAGS register in a temp to push later. Turn off the VM and TF bits, and if the interrupt is serviced by an Interrupt Gate, turn off IF also.

(2) Interrupt and Trap gates must perform a level switch from 3 (where the VM86 program executes) to level 0 (so IRET can return). This process involves a stack switch to the stack given in the TSS for privilege level 0. Save the Virtual 8086 Mode SS and ESP registers to push in a later step. The segment register load of SS will be done as a Protected Mode segment load, since the VM bit was turned off above.

(3) Push the 8086 segment register values onto the new stack, in the order: GS, FS, DS, ES. These are pushed as 32-bit quantities, with undefined values in the upper 16 bits. Then load these 4 registers with null selectors (0).



8086 Application makes "Open File Call" → causes General Protection Fault (Arrow #1)
Virtual 8086 Monitor intercepts call. Calls 386™ DX OS (Arrow #2)
386™ DX OS opens file returns control to 8086 OS (Arrow #3)
8086 OS returns control to application. (Arrow #4)
Transparent to Application

Figure 4-25. Virtual 8086 Environment Interrupt and Call Handling

(4) Push the old 8086 stack pointer onto the new stack by pushing the SS register (as 32-bits, high bits undefined), then pushing the 32-bit ESP register saved above.

(5) Push the 32-bit FLAGS register saved in step 1.

(6) Push the old 8086 instruction pointer onto the new stack by pushing the CS register (as 32-bits, high bits undefined), then pushing the 32-bit EIP register.

(7) Load up the new CS:EIP value from the interrupt gate, and begin execution of the interrupt routine in protected 386 DX mode.

The transition out of virtual 8086 mode performs a level change and stack switch, in addition to changing back to protected mode. In addition, all of the 8086 segment register images are stored on the stack (behind the SS:ESP image), and then loaded with null (0) selectors before entering the interrupt handler. This will permit the handler to safely save and restore the DS, ES, FS, and GS registers as 80286 selectors. This is needed so that interrupt handlers which don't care about the mode of the interrupted program can use the same prolog and epilog code for state saving (i.e. push all registers in prolog, pop all in epilog) regardless of whether or not a "native" mode or Virtual 8086 mode program was interrupted. Restoring null selectors to these registers before executing the IRET will not cause a trap in the interrupt handler. Interrupt routines which expect values in the segment registers, or return values in segment registers will have to obtain/return values from the 8086 register images pushed onto the new stack. They will need to know the mode of the interrupted program in order to know where to find/return segment registers, and also to know how to interpret segment register values.

The IRET instruction will perform the inverse of the above sequence. Only the extended 386 DXs IRET instruction (operand size = 32) can be used, and must be executed at level 0 to change the VM bit to 1.

(1) If the NT bit in the FLAGs register is on, an inter-task return is performed. The current state is stored in the current TSS, and the link field in the current TSS is used to locate the TSS for the interrupted task which is to be resumed.

Otherwise, continue with the following sequence.

(2) Read the FLAGS image from SS:8[ESP] into the FLAGS register. This will set VM to the value active in the interrupted routine.

(3) Pop off the instruction pointer CS:EIP. EIP is popped first, then a 32-bit word is popped which contains the CS value in the lower 16 bits. If VM = 0, this CS load is done as a protected mode segment load. If VM = 1, this will be done as an 8086 segment load.

(4) Increment the ESP register by 4 to bypass the FLAGS image which was "popped" in step 1.

(5) If VM = 1, load segment registers ES, DS, FS, and GS from memory locations SS:[ESP + 8], SS:[ESP + 12], SS:[ESP + 16], and SS:[ESP + 20], respectively, where the new val-

ue of ESP stored in step 4 is used. Since VM = 1, these are done as 8086 segment register loads.

Else if VM = 0, check that the selectors in ES, DS, FS, and GS are valid in the interrupted routine. Null out invalid selectors to trap if an attempt is made to access through them.

(6) If (RPL(CS) > CPL), pop the stack pointer SS:ESP from the stack. The ESP register is popped first, followed by 32-bits containing SS in the lower 16 bits. If VM = 0, SS is loaded as a protected mode segment register load. If VM = 1, an 8086 segment register load is used.

(7) Resume execution of the interrupted routine. The VM bit in the FLAGS register (restored from the interrupt routine's stack image in step 1) determines whether the processor resumes the interrupted routine in Protected mode of Virtual 8086 mode.

5. FUNCTIONAL DATA

5.1 INTRODUCTION

The 386 DX features a straightforward functional interface to the external hardware. The 386 DX has separate, parallel buses for data and address. The data bus is 32-bits in width, and bidirectional. The address bus outputs 32-bit address values in the most directly usable form for the high-speed local bus: 4 individual byte enable signals, and the 30 upper-order bits as a binary value. The data and address buses are interpreted and controlled with their associated control signals.

A **dynamic data bus sizing** feature allows the processor to handle a mix of 32- and 16-bit external buses on a cycle-by-cycle basis (see **5.3.4 Data Bus Sizing**). If 16-bit bus size is selected, the 386 DX automatically makes any adjustment needed, even performing another 16-bit bus cycle to complete the transfer if that is necessary. 8-bit peripheral devices may be connected to 32-bit or 16-bit buses with no loss of performance. A **new address pipelining option** is provided and applies to 32-bit and 16-bit buses for substantially improved memory utilization, especially for the most heavily used memory resources.

The **address pipelining option**, when selected, typically allows a given memory interface to operate with one less wait state than would otherwise be required (see **5.4.2 Address Pipelining**). The pipelined bus is also well suited to interleaved memory designs. When address pipelining is requested by the external hardware, the 386 DX will output the address and bus cycle definition of the next bus cycle (if it is internally available) even while waiting for the current cycle to be acknowledged.

Non-pipelined address timing, however, is ideal for external cache designs, since the cache memory will typically be fast enough to allow non-pipelined cycles. For maximum design flexibility, the address pipelining option is selectable on a cycle-by-cycle basis.

The processor's bus cycle is the basic mechanism for information transfer, either from system to processor, or from processor to system. 386 DX bus cycles perform data transfer in a minimum of only two clock periods. On a 32-bit data bus, the maximum 386 DX transfer bandwidth at 20 MHz is therefore 40 MBytes/sec, at 25 MHz bandwidth, is 50 Mbytes/sec, and at 33 MHz bandwidth, is 66 Mbytes/sec. Any bus cycle will be extended for more than two clock periods, however, if external hardware withholds acknowledgement of the cycle. At the appropriate time, acknowledgement is signalled by asserting the 386 DX READY# input.

The 386 DX can relinquish control of its local buses to allow mastership by other devices, such as direct memory access channels. When relinquished, HLDA is the only output pin driven by the 386 DX providing near-complete isolation of the processor from its system. The near-complete isolation characteristic is ideal when driving the system from test equipment, and in fault-tolerant applications.

Functional data covered in this chapter describes the processor's hardware interface. First, the set of signals available at the processor pins is described (see **5.2 Signal Description**). Following that are the signal waveforms occurring during bus cycles (see **5.3 Bus Transfer Mechanism**, **5.4 Bus Functional Description** and **5.5 Other Functional Descriptions**).

5.2 SIGNAL DESCRIPTION

5.2.1 Introduction

Ahead is a brief description of the 386 DX input and output signals arranged by functional groups. Note the # symbol at the end of a signal name indicates the active, or asserted, state occurs when the signal is at a low voltage. When no # is present after the signal name, the signal is asserted when at the high voltage level.

Example signal: M/IO# — High voltage indicates Memory selected

— Low voltage indicates I/O selected

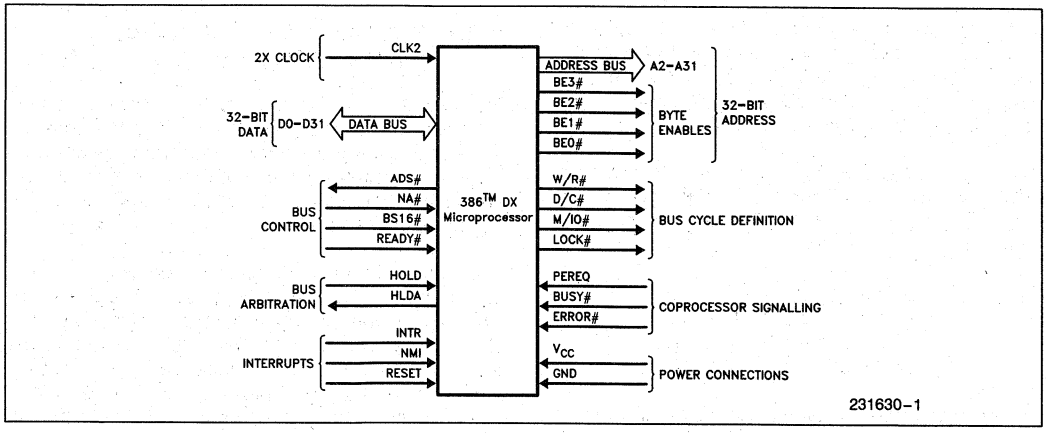


Figure 5-1. Functional Signal Groups

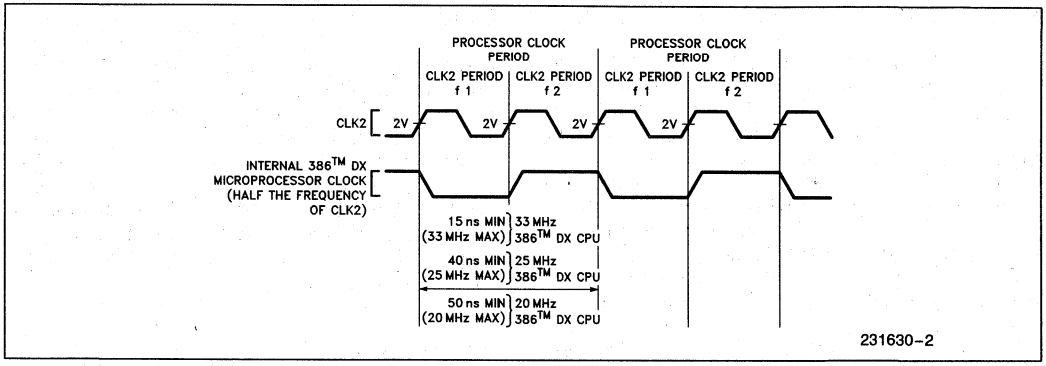


Figure 5-2. CLK2 Signal and Internal Processor Clock

The signal descriptions sometimes refer to AC timing parameters, such as "t_{25} Reset Setup Time" and "t_{26} Reset Hold Time." The values of these parameters can be found in Tables 7-4 and 7-5.

5.2.2 Clock (CLK2)

CLK2 provides the fundamental timing for the 386 DX. It is divided by two internally to generate the internal processor clock used for instruction execution. The internal clock is comprised of two phases, "phase one" and "phase two." Each CLK2 period is a phase of the internal clock. Figure 5-2 illustrates the relationship. If desired, the phase of the internal processor clock can be synchronized to a known phase by ensuring the RESET signal falling edge meets its applicable setup and hold times, t_{25} and t_{26}.

5.2.3 Data Bus (D0 through D31)

These three-state bidirectional signals provide the general purpose data path between the 386 DX and other devices. Data bus inputs and outputs indicate "1" when HIGH. The data bus can transfer data on 32- and 16-bit buses using a data bus sizing feature controlled by the BS16# input. See section **5.2.6 Bus Contol**. Data bus reads require that read data setup and hold times t_{21} and t_{22} be met for correct operation. In addition, the 386 DX requires that all data bus pins be at a valid logic state (high or low) at the end of each read cycle, when READY# is asserted. During any write operation (and during halt cycles and shutdown cycles), the 386 DX always drives all 32 signals of the data bus even if the current bus size is 16-bits.

5.2.4 Address Bus (BE0# through BE3#, A2 through A31)

These three-state outputs provide physical memory addresses or I/O port addresses. The address bus is capable of addressing 4 gigabytes of physical memory space (00000000H through FFFFFFFFH), and 64 kilobytes of I/O address space (00000000H through 0000FFFFH) for programmed I/O. I/O

transfers automatically generated for 386 DX-to-coprocessor communication use I/O addresses 800000F8H through 800000FFH, so A31 HIGH in conjunction with M/IO# LOW allows simple generation of the coprocessor select signal.

The Byte Enable outputs, BE0#–BE3#, directly indicate which bytes of the 32-bit data bus are involved with the current transfer. This is most convenient for external hardware.

BE0# applies to D0–D7
BE1# applies to D8–D15
BE2# applies to D16–D23
BE3# applies to D24–D31

The number of Byte Enables asserted indicates the physical size of the operand being transferred (1, 2, 3, or 4 bytes). Refer to section **5.3.6 Operand Alignment**.

When a memory write cycle or I/O write cycle is in progress, and the operand being transferred occupies **only** the upper 16 bits of the data bus (D16–D31), duplicate data is simultaneously presented on the corresponding lower 16-bits of the data bus (D0–D15). This duplication is performed for optimum write performance on 16-bit buses. The pattern of write data duplication is a function of the Byte Enables asserted during the write cycle. Table 5-1 lists the write data present on D0–D31, as a function of the asserted Byte Enable outputs BE0#–BE3#.

5.2.5 Bus Cycle Definition Signals (W/R#, D/C#, M/IO#, LOCK#)

These three-state outputs define the type of bus cycle being performed. W/R# distinguishes between write and read cycles. D/C# distinguishes between data and control cycles. M/IO# distinguishes between memory and I/O cycles. LOCK# distinguishes between locked and unlocked bus cycles.

The primary bus cycle definition signals are W/R#, D/C# and M/IO#, since these are the signals driven valid as the ADS# (Address Status output) is driven asserted. The LOCK# is driven valid at the same time as the first locked bus cycle begins, which due to address pipelining, could be later than ADS# is driven asserted. See **5.4.3.4 Pipelined Address.** The LOCK# is negated when the READY# input terminates the last bus cycle which was locked.

Exact bus cycle definitions, as a function of W/R#, D/C#, and M/IO#, are given in Table 5-2. Note one combination of W/R#, D/C# and M/IO# is never given when ADS# is asserted (however, that combination, which is listed as "does not occur," may occur during **idle** bus states when ADS# is **not** asserted). If M/IO#, D/C#, and W/R# are qualified by ADS# asserted, then a decoding scheme may be simplified by using this definition of the "does not occur" combination.

Table 5-1. Write Data Duplication as a Function of BE0#–BE3#

386™ DX Byte Enables				386™ DX Write Data				Automatic Duplication?
BE3#	BE2#	BE1#	BE0#	D24–D31	D16–D23	D8–D15	D0–D7	
High	High	High	Low	undef	undef	undef	A	No
High	High	Low	High	undef	undef	B	undef	No
High	Low	High	High	undef	C	undef	C	Yes
Low	High	High	High	D	undef	D	undef	Yes
High	High	Low	Low	undef	undef	B	A	No
High	Low	Low	High	undef	C	B	undef	No
Low	Low	High	High	D	C	D	C	Yes
High	Low	Low	Low	undef	C	B	A	No
Low	Low	Low	High	D	C	B	undef	No
Low	Low	Low	Low	D	C	B	A	No

Key:
D = logical write data d24–d31
C = logical write data d16–d23
B = logical write data d8–d15
A = logical write data d0–d7

Table 5-2. Bus Cycle Definition

M/IO#	D/C#	W/R#	Bus Cycle Type	Locked?
Low	Low	Low	INTERRUPT ACKNOWLEDGE	Yes
Low	Low	High	does not occur	—
Low	High	Low	I/O DATA READ	No
Low	High	High	I/O DATA WRITE	No
High	Low	Low	MEMORY CODE READ	No
High	Low	High	HALT: Address = 2 (BE0# High, BE1# High, BE2# Low, BE3# High, A2–A31 Low) SHUTDOWN: Address = 0 (BE0# Low, BE1# High, BE2# High, BE3# High, A2–A31 Low)	No
High	High	Low	MEMORY DATA READ	Some Cycles
High	High	High	MEMORY DATA WRITE	Some Cycles

5.2.6 Bus Control Signals (ADS#, READY#, NA#, BS16#)

5.2.6.1 INTRODUCTION

The following signals allow the processor to indicate when a bus cycle has begun, and allow other system hardware to control address pipelining, data bus width and bus cycle termination.

5.2.6.2 ADDRESS STATUS (ADS#)

This three-state output indicates that a valid bus cycle definition, and address (W/R#, D/C#, M/IO#, BE0#–BE3#, and A2–A31) is being driven at the 386 DX pins. It is asserted during T1 and T2P bus states (see **5.4.3.2 Non-pipelined Address** and **5.4.3.4 Pipelined Address** for additional information on bus states).

5.2.6.3 TRANSFER ACKNOWLEDGE (READY#)

This input indicates the current bus cycle is complete, and the active bytes indicated by BE0#–BE3# and BS16# are accepted or provided. When READY# is sampled asserted during a read cycle or interrupt acknowledge cycle, the 386 DX latches the input data and terminates the cycle. When READY# is sampled asserted during a write cycle, the processor terminates the bus cycle.

READY# is ignored on the first bus state of all bus cycles, and sampled each bus state thereafter until asserted. READY# must eventually be asserted to acknowledge every bus cycle, including Halt Indication and Shutdown Indication bus cycles. When being sampled, READY must always meet setup and hold times t_{19} and t_{20} for correct operation. See all sections of **5.4 Bus Functional Description**.

5.2.6.4 NEXT ADDRESS REQUEST (NA#)

This is used to request address pipelining. This input indicates the system is prepared to accept new values of BE0#–BE3#, A2–A31, W/R#, D/C# and M/IO# from the 386 DX even if the end of the current cycle is not being acknowledged on READY#. If this input is asserted when sampled, the next address is driven onto the bus, provided the next bus request is already pending internally. See **5.4.2 Address Pipelining** and **5.4.3 Read and Write Cycles**. NA# must always meet setup and hold times, t_{15} and t_{16}, for correct operation.

5.2.6.5 BUS SIZE 16 (BS16#)

The BS16# feature allows the 386 DX to directly connect to 32-bit and 16-bit data buses. Asserting this input constrains the current bus cycle to use only the lower-order half (D0–D15) of the data bus, corresponding to BE0# and BE1#. Asserting BS16# has no additional effect if only BE0# and/or BE1# are asserted in the current cycle. However, during bus cycles asserting BE2# or BE3#, asserting BS16# will automatically cause the 386 DX to make adjustments for correct transfer of the upper bytes(s) using only physical data signals D0–D15.

If the operand spans both halves of the data bus and BS16# is asserted, the 386 DX will automatically perform another 16-bit bus cycle. BS16# must always meet setup and hold times t_{17} and t_{18} for correct operation.

386 DX I/O cycles are automatically generated for coprocessor communication. Since the 386 DX must transfer 32-bit quantities between itself and the 387 DX, BS16# *must not* be asserted during 387 DX communication cycles.

5.2.7 Bus Arbitration Signals (HOLD, HLDA)

5.2.7.1 INTRODUCTION

This section describes the mechanism by which the processor relinquishes control of its local buses when requested by another bus master device. See **5.5.1 Entering and Exiting Hold Acknowledge** for additional information.

5.2.7.2 BUS HOLD REQUEST (HOLD)

This input indicates some device other than the 386 DX requires bus mastership.

HOLD must remain asserted as long as any other device is a local bus master. HOLD is not recognized while RESET is asserted. If RESET is asserted while HOLD is asserted, RESET has priority and places the bus into an idle state, rather than the hold acknowledge (high impedance) state.

HOLD is level-sensitive and is a synchronous input. HOLD signals must always meet setup and hold times t_{23} and t_{24} for correct operation.

5.2.7.3 BUS HOLD ACKNOWLEDGE (HLDA)

Assertion of this output indicates the 386 DX has relinquished control of its local bus in response to HOLD asserted, and is in the bus Hold Acknowledge state.

The Hold Acknowledge state offers near-complete signal isolation. In the Hold Acknowledge state, HLDA is the only signal being driven by the 386 DX. The other output signals or bidirectional signals (D0–D31, BE0#–BE3#, A2–A31, W/R#, D/C#, M/IO#, LOCK# and ADS#) are in a high-impedance state so the requesting bus master may control them. Pullup resistors may be desired on several signals to avoid spurious activity when no bus master is driving them. See **7.2.3 Resistor Recommendations**. Also, one rising edge occuring on the NMI input during Hold Acknowledge is remembered, for processing after the HOLD input is negated.

In addition to the normal usage of Hold Acknowledge with DMA controllers or master peripherals, the near-complete isolation has particular attractiveness during system test when test equipment drives the system, and in hardware-fault-tolerant applications.

5.2.8 Coprocessor Interface Signals (PEREQ, BUSY#, ERROR#)

5.2.8.1 INTRODUCTION

In the following sections are descriptions of signals dedicated to the numeric coprocessor interface. In addition to the data bus, address bus, and bus cycle definition signals, these following signals control communication between the 386 DX and its 387 DX processor extension.

5.2.8.2 COPROCESSOR REQUEST (PEREQ)

When asserted, this input signal indicates a coprocessor request for a data operand to be transferred to/from memory by the 386 DX. In response, the 386 DX transfers information between the coprocessor and memory. Because the 386 DX has internally stored the coprocessor opcode being executed, it performs the requested data transfer with the correct direction and memory address.

PEREQ is level-sensitive and is allowed to be asynchronous to the CLK2 signal.

5.2.8.3 COPROCESSOR BUSY (BUSY#)

When asserted, this input indicates the coprocessor is still executing an instruction, and is not yet able to accept another. When the 386 DX encounters any coprocessor instruction which operates on the numeric stack (e.g. load, pop, or arithmetic operation), or the WAIT instruction, this input is first automatically sampled until it is seen to be negated. This sampling of the BUSY# input prevents overrunning the execution of a previous coprocessor instruction.

The FNINIT and FNCLEX coprocessor instructions are allowed to execute even if BUSY# is asserted, since these instructions are used for coprocessor initialization and exception-clearing.

BUSY# is level-sensitive and is allowed to be asynchronous to the CLK2 signal.

BUSY# serves an additional function. If BUSY# is sampled LOW at the falling edge of RESET, the 386 DX performs an internal self-test (see **5.5.3 Bus Activity During and Following Reset**). If BUSY# is sampled HIGH, no self-test is performed.

5.2.8.4 COPROCESSOR ERROR (ERROR#)

This input signal indicates that the previous coprocessor instruction generated a coprocessor error of a type not masked by the coprocessor's control register. This input is automatically sampled by the 386 DX when a coprocessor instruction is encountered, and if asserted, the 386 DX generates exception 16 to access the error-handling software.

Several coprocessor instructions, generally those which clear the numeric error flags in the coprocessor or save coprocessor state, do execute without the 386 DX generating exception 16 even if ERROR# is asserted. These instructions are FNINIT, FNCLEX, FSTSW, FSTSWAX, FSTCW, FSTENV, FSAVE, FESTENV and FESAVE.

ERROR# is level-sensitive and is allowed to be asynchronous to the CLK2 signal.

5.2.9 Interrupt Signals (INTR, NMI, RESET)

5.2.9.1 INTRODUCTION

The following descriptions cover inputs that can interrupt or suspend execution of the processor's current instruction stream.

5.2.9.2 MASKABLE INTERRUPT REQUEST (INTR)

When asserted, this input indicates a request for interrupt service, which can be masked by the 386 DX Flag Register IF bit. When the 386 DX responds to the INTR input, it performs two interrupt acknowledge bus cycles, and at the end of the second, latches an 8-bit interrupt vector on D0–D7 to identify the source of the interrupt.

INTR is level-sensitive and is allowed to be asynchronous to the CLK2 signal. To assure recognition of an INTR request, INTR should remain asserted until the first interrupt acknowledge bus cycle begins.

5.2.9.3 NON-MASKABLE INTERRUPT REQUEST (NMI)

This input indicates a request for interrupt service, which cannot be masked by software. The non-maskable interrupt request is always processed according to the pointer or gate in slot 2 of the interrupt table. Because of the fixed NMI slot assignment, no interrupt acknowledge cycles are perfomed when processing NMI.

NMI is rising edge-sensitive and is allowed to be asynchronous to the CLK2 signal. To assure recognition of NMI, it must be negated for at least eight CLK2 periods, and then be asserted for at least eight CLK2 periods.

Once NMI processing has begun, no additional NMI's are processed until after the next IRET instruction, which is typically the end of the NMI service routine. If NMI is re-asserted prior to that time, however, one rising edge on NMI will be remembered for processing after executing the next IRET instruction.

5.2.9.4 RESET (RESET)

This input signal suspends any operation in progress and places the 386 DX in a known reset state. The 386 DX is reset by asserting RESET for 15 or more CLK2 periods (80 or more CLK2 periods before requesting self test). When RESET is asserted, all other input pins are ignored, and all other bus pins are driven to an idle bus state as shown in Table 5-3. If RESET and HOLD are both asserted at a point in time, RESET takes priority even if the 386 DX was in a Hold Acknowledge state prior to RESET asserted.

RESET is level-sensitive and must be synchronous to the CLK2 signal. If desired, the phase of the internal processor clock, and the entire 386 DX state can be completely synchronized to external circuitry by ensuring the RESET signal falling edge meets its applicable setup and hold times, t_{25} and t_{26}.

Table 5-3. Pin State (Bus Idle) During Reset

Pin Name	Signal Level During Reset
ADS#	High
D0–D31	High Impedance
BE0#–BE3#	Low
A2–A31	High
W/R#	Low
D/C#	High
M/IO#	Low
LOCK#	High
HLDA	Low

5.2.10 Signal Summary

Table 5-4 summarizes the characteristics of all 386 DX signals.

Table 5-4. 386™ DX Signal Summary

Signal Name	Signal Function	Active State	Input/ Output	Input Synch or Asynch to CLK2	Output High Impedance During HLDA?
CLK2	Clock	—	I	—	—
D0–D31	Data Bus	High	I/O	S	Yes
BE0#–BE3#	Byte Enables	Low	O	—	Yes
A2–A31	Address Bus	High	O	—	Yes
W/R#	Write-Read Indication	High	O	—	Yes
D/C#	Data-Control Indication	High	O	—	Yes
M/IO#	Memory-I/O Indication	High	O	—	Yes
LOCK#	Bus Lock Indication	Low	O	—	Yes
ADS#	Address Status	Low	O	—	Yes
NA#	Next Address Request	Low	I	S	—
BS16#	Bus Size 16	Low	I	S	—
READY#	Transfer Acknowledge	Low	I	S	—
HOLD	Bus Hold Request	High	I	S	—
HLDA	Bus Hold Acknowledge	High	O	—	No
PEREQ	Coprocessor Request	High	I	A	—
BUSY#	Coprocessor Busy	Low	I	A	—
ERROR#	Coprocessor Error	Low	I	A	—
INTR	Maskable Interrupt Request	High	I	A	—
NMI	Non-Maskable Intrpt Request	High	I	A	—
RESET	Reset	High	I	S	—

5.3 BUS TRANSFER MECHANISM

5.3.1 Introduction

All data transfers occur as a result of one or more bus cycles. Logical data operands of byte, word and double-word lengths may be transferred without restrictions on physical address alignment. Any byte boundary may be used, although two or even three physical bus cycles are performed as required for unaligned operand transfers. See **5.3.4 Dynamic Data Bus Sizing** and **5.3.6 Operand Alignment**.

The 386 DX address signals are designed to simplify external system hardware. Higher-order address bits are provided by A2–A31. Lower-order address in the form of BE0#–BE3# directly provides linear selects for the four bytes of the 32-bit data bus. Physical operand size information is thereby implicitly provided each bus cycle in the most usable form.

Byte Enable outputs BE0#–BE3# are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 5-5. During a bus cycle, any possible pattern of contiguous, asserted Byte Enable outputs can occur, but never patterns having a negated Byte Enable separating two or three asserted Enables.

Address bits A0 and A1 of the physical operand's base address can be created when necessary (for instance, for MULTIBUS® I or MULTIBUS® II interface), as a function of the lowest-order asserted Byte Enable. This is shown by Table 5-6. Logic to generate A0 and A1 is given by Figure 5-3.

Table 5-5. Byte Enables and Associated Data and Operand Bytes

Byte Enable Signal	Associated Data Bus Signals
BE0#	D0–D7 (byte 0—least significant)
BE1#	D8–D15 (byte 1)
BE2#	D16–D23 (byte 2)
BE3#	D24–D31 (byte 3—most significant)

Table 5-6. Generating A0–A31 from BE0#–BE3# and A2–A31

386™ DX Address Signals								
A31 ..... A2					BE3#	BE2#	BE1#	BE0#
	Physical Base Address							
A31	.....	A2	A1	A0				
A31	.....	A2	0	0	X	X	X	Low
A31	.....	A2	0	1	X	X	Low	High
A31	.....	A2	1	0	X	Low	High	High
A31	.....	A2	1	1	Low	High	High	High

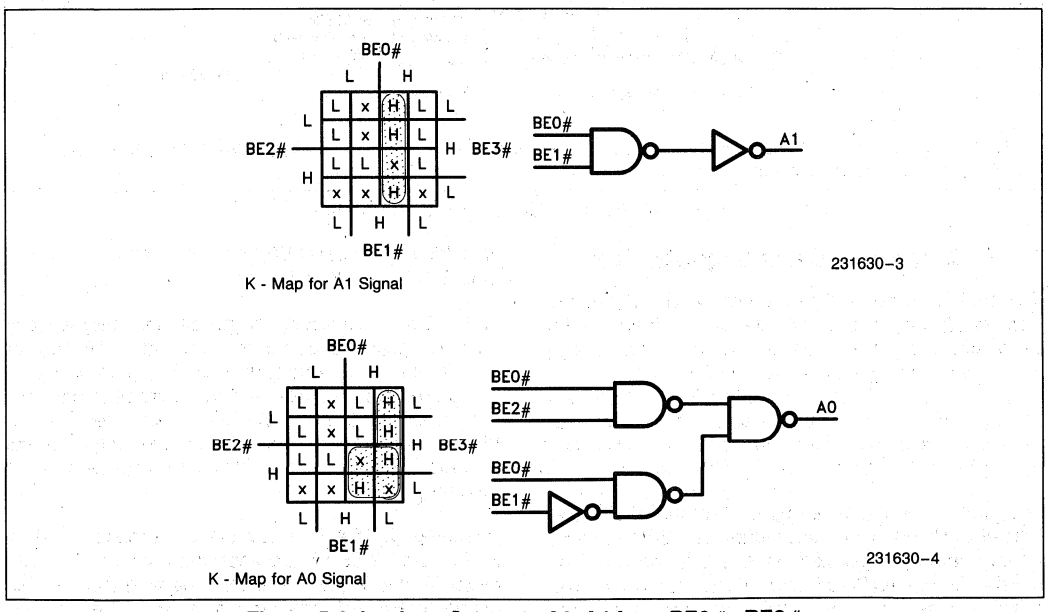


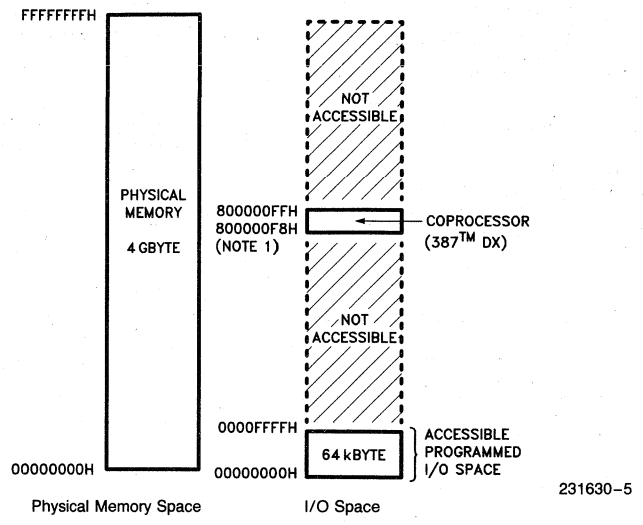
Figure 5-3. Logic to Generate A0, A1 from BE0#–BE3#

Each bus cycle is composed of at least two bus states. Each bus state requires one processor clock period. Additional bus states added to a single bus cycle are called wait states. See **5.4 Bus Functional Description**.

Since a bus cycle requires a minimum of two bus states (equal to two processor clock periods), data can be transferred between external devices and the 386 DX at a maximum rate of one 4-byte Dword every two processor clock periods, for a maximum bus bandwidth of 66 megabytes/second (386 DX operating at 33 MHz processor clock rate).

5.3.2 Memory and I/O Spaces

Bus cycles may access physical memory space or I/O space. Peripheral devices in the system may either be memory-mapped, or I/O-mapped, or both. As shown in Figure 5-4, physical memory addresses range from 00000000H to FFFFFFFFH (4 gigabytes) and I/O addresses from 00000000H to 0000FFFFH (64 kilobytes) for programmed I/O. Note the I/O addresses used by the automatic I/O cycles for coprocessor communication are 800000F8H to 800000FFH, beyond the address range of programmed I/O, to allow easy generation of a coprocessor chip select signal using the A31 and M/IO# signals.



NOTE:
Since A31 is HIGH during automatic communication with coprocessor, A31 HIGH and M/IO# LOW can be used to easily generate a coprocessor select signal.

Figure 5-4. Physical Memory and I/O Spaces

5.3.3 Memory and I/O Organization

The 386 DX datapath to memory and I/O spaces can be 32 bits wide or 16 bits wide. When 32-bits wide, memory and I/O spaces are organized naturally as arrays of physical 32-bit Dwords. Each memory or I/O Dword has four individually addressable bytes at consecutive byte addresses. The lowest-addressed byte is associated with data signals D0–D7; the highest-addressed byte with D24–D31.

The 386 DX includes a bus control input, BS16#, that also allows direct connection to 16-bit memory or I/O spaces organized as a sequence of 16-bit words. Cycles to 32-bit and 16-bit memory or I/O devices may occur in any sequence, since the BS16# control is sampled during each bus cycle. See **5.3.4 Dynamic Data Bus Sizing**. The Byte Enable signals, BE0#–BE3#, allow byte granularity when addressing any memory or I/O structure, whether 32 or 16 bits wide.

5.3.4 Dynamic Data Bus Sizing

Dynamic data bus sizing is a feature allowing direct processor connection to 32-bit or 16-bit data buses for memory or I/O. A single processor may connect to both size buses. Transfers to or from 32- or 16-bit ports are supported by dynamically determining the bus width during each bus cycle. During each bus cycle an address decoding circuit or the slave device itself may assert BS16# for 16-bit ports, or negate BS16# for 32-bit ports.

With BS16# asserted, the processor automatically converts operand transfers larger than 16 bits, or misaligned 16-bit transfers, into two or three transfers as required. All operand transfers physically occur on D0–D15 when BS16# is asserted. Therefore, 16-bit memories or I/O devices only connect on data signals D0–D15. No extra transceivers are required.

Asserting BS16# only affects the processor when BE2# and/or BE3# are asserted during the current cycle. If only D0–D15 are involved with the transfer, asserting BS16# has no affect since the transfer can proceed normally over a 16-bit bus whether BS16# is asserted or not. In other words, asserting BS16# has no effect when only the lower half of the bus is involved with the current cycle.

There are two types of situations where the processor is affected by asserting BS16#, depending on which Byte Enables are asserted during the current bus cycle:

Upper Half Only:
Only BE2# and/or BE3# asserted.

Upper and Lower Half:
At least BE1#, BE2# asserted (and perhaps also BE0# and/or BE3#).

Effect of asserting BS16# during "upper half only" read cycles:

Asserting BS16# during "upper half only" reads causes the 386 DX to read data on the lower 16 bits of the data bus and ignore data on the upper 16 bits of the data bus. Data that would have been read from D16–D31 (as indicated by BE2# and BE3#) will instead be read from D0–D15 respectively.

Effect of asserting BS16# during "upper half only" write cycles:

Asserting BS16# during "upper half only" writes does not affect the 386 DX. When only BE2# and/or BE3# are asserted during a write cycle the 386 DX always duplicates data signals D16–D31 onto D0–D15 (see Table 5-1). Therefore, no further 386 DX action is required to perform these writes on 32-bit or 16-bit buses.

Effect of asserting BS16# during "upper and lower half" read cycles:

Asserting BS16# during "upper and lower half" reads causes the processor to perform two 16-bit read cycles for complete physical operand transfer. Bytes 0 and 1 (as indicated by BE0# and BE1#) are read on the first cycle using D0–D15. Bytes 2 and 3 (as indicated by BE2# and BE3#) are read during the second cycle, again using D0–D15. D16–D31 are ignored during both 16-bit cycles. BE0# and BE1# are always negated during the second 16-bit cycle (See **Figure 5-14, cycles 2 and 2a**).

Effect of asserting BS16# during "upper and lower half" write cycles:

Asserting BS16# during "upper and lower half" writes causes the 386 DX to perform two 16-bit write cycles for complete physical operand transfer. All bytes are available the first write cycle allowing external hardware to receive Bytes 0 and 1 (as indicated by BE0# and BE1#) using D0–D15. On the second cycle the 386 DX duplicates Bytes 2 and 3 on D0–D15 and Bytes 2 and 3 (as indicated by BE2# and BE3#) are written using D0–D15. BE0# and BE1# are always negated during the second 16-bit cycle. BS16# must be asserted during the second 16-bit cycle. See **Figure 5-14, cycles 1 and 1a**.

5.3.5 Interfacing with 32- and 16-Bit Memories

In 32-bit-wide physical memories such as Figure 5-5, each physical Dword begins at a byte address that is a multiple of 4. A2–A31 are directly used as a Dword select and BE0#–BE3# as byte selects. BS16# is negated for all bus cycles involving the 32-bit array.

When 16-bit-wide physical arrays are included in the system, as in Figure 5-6, each 16-bit physical word begins at a address that is a multiple of 2. Note the address is decoded, to assert BS16# only during bus cycles involving the 16-bit array. (If desiring to

5

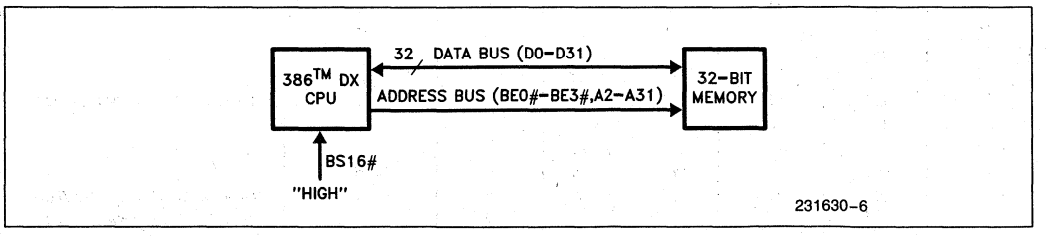


Figure 5-5. 386™ DX with 32-Bit Memory

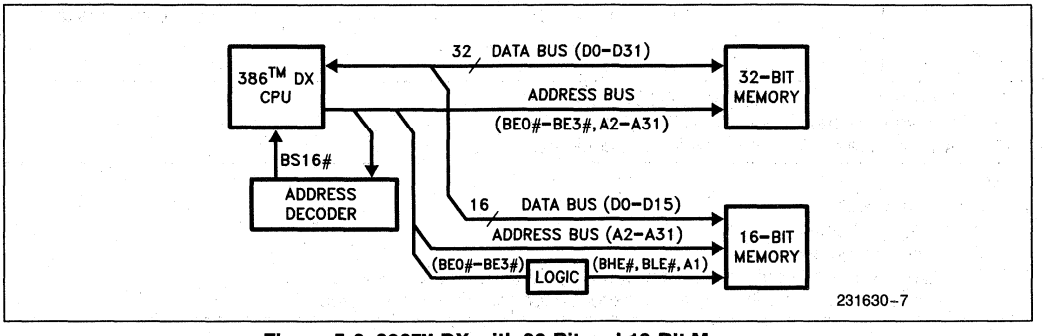


Figure 5-6. 386™ DX with 32-Bit and 16-Bit Memory

use pipelined address with 16-bit memories then BE0#–BE3# and W/R# are also decoded to determine when BS16# should be asserted. See **5.4.3.6 Pipelined Address with Dynamic Data Bus Sizing.**)

A2–A31 are directly usable for addressing 32-bit and 16-bit devices. To address 16-bit devices, A1 and two byte enable signals are also needed.

To generate an A1 signal and two Byte Enable signals for 16-bit access, BE0#–BE3# should be decoded as in Table 5-7. Note certain combinations of BE0#–BE3# are never generated by the 386 DX, leading to "don't care" conditions in the decoder. Any BE0#–BE3# decoder, such as Figure 5-7, may use the non-occurring BE0#–BE3# combinations to its best advantage.

5.3.6 Operand Alignment

With the flexibility of memory addressing on the 386 DX, it is possible to transfer a logical operand that spans more than one physical Dword or word of memory or I/O. Examples are 32-bit Dwordoperands beginning at addresses not evenly divisible by 4, or a 16-bit word operand split between two physical Dwords of the memory array.

Operand alignment and data bus size dictate when multiple bus cycles are required. Table 5-8 describes the transfer cycles generated for all combinations of logical operand lengths, alignment, and data bus sizing. When multiple bus cycles are required to transfer a multi-byte logical operand, the highest-order bytes are transferred first (but if BS16# asserted requires two 16-bit cycles be performed, that part of the transfer is low-order first).

5.4 BUS FUNCTIONAL DESCRIPTION

5.4.1 Introduction

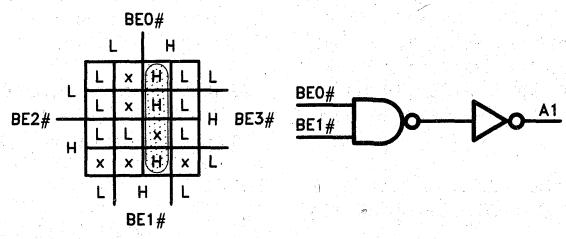
The 386 DX has separate, parallel buses for data and address. The data bus is 32-bits in width, and bidirectional. The address bus provides a 32-bit value using 30 signals for the 30 upper-order address bits and 4 Byte Enable signals to directly indicate the active bytes. These buses are interpreted and controlled via several associated definition or control signals.

Table 5-7. Generating A1, BHE# and BLE# for Addressing 16-Bit Devices

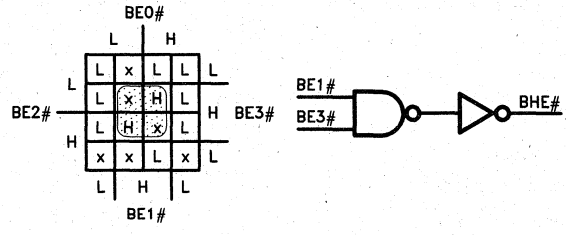
386™ DX Signals				16-Bit Bus Signals			Comments
BE3#	BE2#	BE1#	BE0#	A1	BHE#	BLE# (A0)	
H*	H*	H*	H*	x	x	x	x—no active bytes
H	H	H	L	L	H	L	
H	H	L	H	L	L	H	
H	H	L	L	L	L	L	
H	L	H	H	H	H	L	
H*	L*	H*	L*	x	x	x	x—not contiguous bytes
H	L	L	H	L	L	H	
H	L	L	L	L	L	L	
L	H	H	H	H	L	H	
L*	H*	H*	L*	x	x	x	x—not contiguous bytes
L*	H*	L*	H*	x	x	x	x—not contiguous bytes
L*	H*	L*	L*	x	x	x	x—not contiguous bytes
L	L	H	H	H	L	L	
L*	L*	H*	L*	x	x	x	x—not continguous bytes
L	L	L	H	L	L	H	
L	L	L	L	L	L	L	

BLE# asserted when D0–D7 of 16-bit bus is active.
BHE# asserted when D8–D15 of 16-bit bus is active.
A1 low for all even words; A1 high for all odd words.

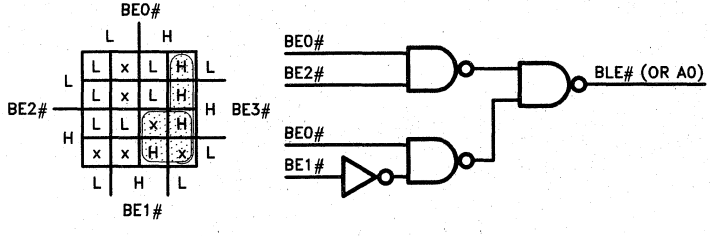
Key:
x = don't care
H = high voltage level
L = low voltage level
* = a non-occurring pattern of Byte Enables; either none are asserted, or the pattern has Byte Enables asserted for non-contiguous bytes



K-map for A1 signal (same as Figure 5-3)



K-map for 16-bit BHE# signal



K-map for 16-bit BLE# signal (same as A0 signal in Figure 5-3)

Figure 5-7. Logic to Generate A1, BHE# and BLE# for 16-Bit Buses

Table 5-8. Transfer Bus Cycles for Bytes, Words and Dwords

	Byte-Length of Logical Operand								
	1	2				4			
Physical Byte Address in Memory (low-order bits)	xx	00	01	10	11	00	01	10	11
Transfer Cycles over 32-Bit Data Bus	b	w	w	w	hb,* lb	d	hb l3	hw, lw	h3, lb
Transfer Cycles over 16-Bit Data Bus	b	w	lb, hb	w	hb, lb	lw, hw	hb, lb, mw	hw, lw	mw, hb, lb

Key: b = byte transfer
w = word transfer
l = low-order portion
m = mid-order portion
x = don't care
3 = 3-byte transfer
d = Dword transfer
h = high-order portion
= BS16# asserted causes second bus cycle

*For this case, 8086, 8088, 80186, 80188, 80286 transfer lb first, then hb.

The definition of each bus cycle is given by three definition signals: M/IO#, W/R# and D/C#. At the same time, a valid address is present on the byte enable signals BE0#–BE3# and other address signals A2–A31. A status signal, ADS#, indicates when the 386 DX issues a new bus cycle definition and address.

Collectively, the address bus, data bus and all associated control signals are referred to simply as "the bus".

When active, the bus performs one of the bus cycles below:

1) read from memory space
2) locked read from memory space
3) write to memory space
4) locked write to memory space
5) read from I/O space (or coprocessor)
6) write to I/O space (or coprocessor)
7) interrupt acknowledge
8) indicate halt, or indicate shutdown

Table 5-2 shows the encoding of the bus cycle definition signals for each bus cycle. See section **5.2.5 Bus Cycle Definition**.

The data bus has a dynamic sizing feature supporting 32- and 16-bit bus size. Data bus size is indicated to the 386 DX using its Bus Size 16 (BS16#) input. All bus functions can be performed with either data bus size.

When the 386 DX bus is not performing one of the activities listed above, it is either Idle or in the Hold Acknowledge state, which may be detected by external circuitry. The idle state can be identified by the 386 DX giving no further assertions on its address strobe output (ADS#) since the beginning of its most recent bus cycle, and the most recent bus cycle has been terminated. The hold acknowledge state is identified by the 386 DX asserting its hold acknowledge (HLDA) output.

The shortest time unit of bus activity is a bus state. A bus state is one processor clock period (two CLK2 periods) in duration. A complete data transfer occurs during a bus cycle, composed of two or more bus states.

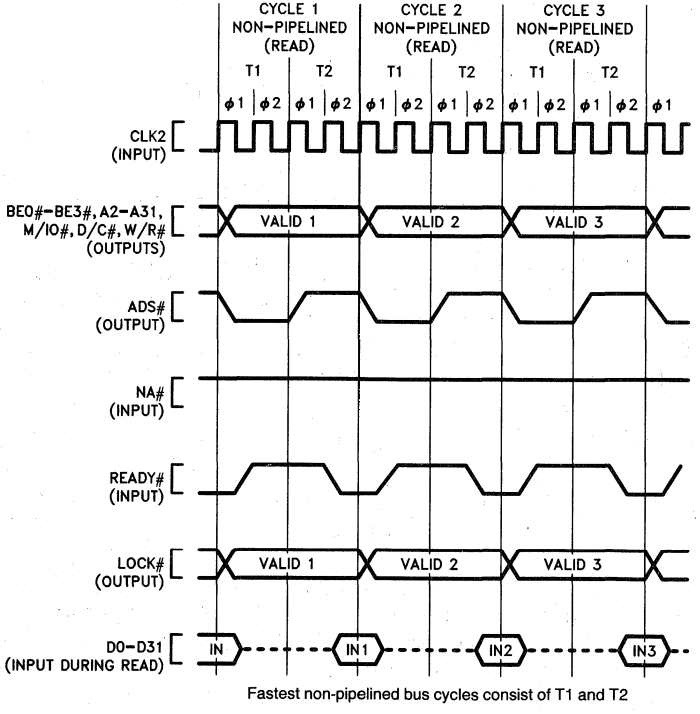


Figure 5-8. Fastest Read Cycles with Non-Pipelined Address Timing

The fastest 386 DX bus cycle requires only two bus states. For example, three consecutive bus read cycles, each consisting of two bus states, are shown by Figure 5-8. The bus states in each cycle are named **T1** and **T2**. Any memory or I/O address may be accessed by such a two-state bus cycle, if the external hardware is fast enough. The high-bandwidth, two-clock bus cycle realizes the full potential of fast main memory, or cache memory.

Every bus cycle continues until it is acknowledged by the external system hardware, using the 386 DX READY# input. Acknowledging the bus cycle at the end of the first T2 results in the shortest bus cycle, requiring only T1 and T2. If READY# is not immediately asserted, however, T2 states are repeated indefinitely until the READY# input is sampled asserted.

5.4.2 Address Pipelining

The address pipelining option provides a choice of bus cycle timings. Pipelined or non-pipelined address timing is selectable on a cycle-by-cycle basis with the Next Address (NA#) input.

When address pipelining is not selected, the current address and bus cycle definition remain stable throughout the bus cycle.

When address pipelining is selected, the address (BE0#–BE3#, A2–A31) and definition (W/R#, D/C# and M/IO#) of the next cycle are available before the end of the current cycle. To signal their availability, the 386 DX address status output (ADS#) is also asserted. Figure 5-9 illustrates the fastest read cycles with pipelined address timing.

Note from Figure 5-9 the fastest bus cycles using pipelined address require only two bus states, named **T1P** and **T2P**. Therefore cycles with pipelined address timing allow the same data bandwidth as non-pipelined cycles, but address-to-data access time is increased compared to that of a non-pipelined cycle.

By increasing the address-to-data access time, pipelined address timing reduces wait state requirements. For example, if one wait state is required with non-pipelined address timing, no wait states would be required with pipelined address.

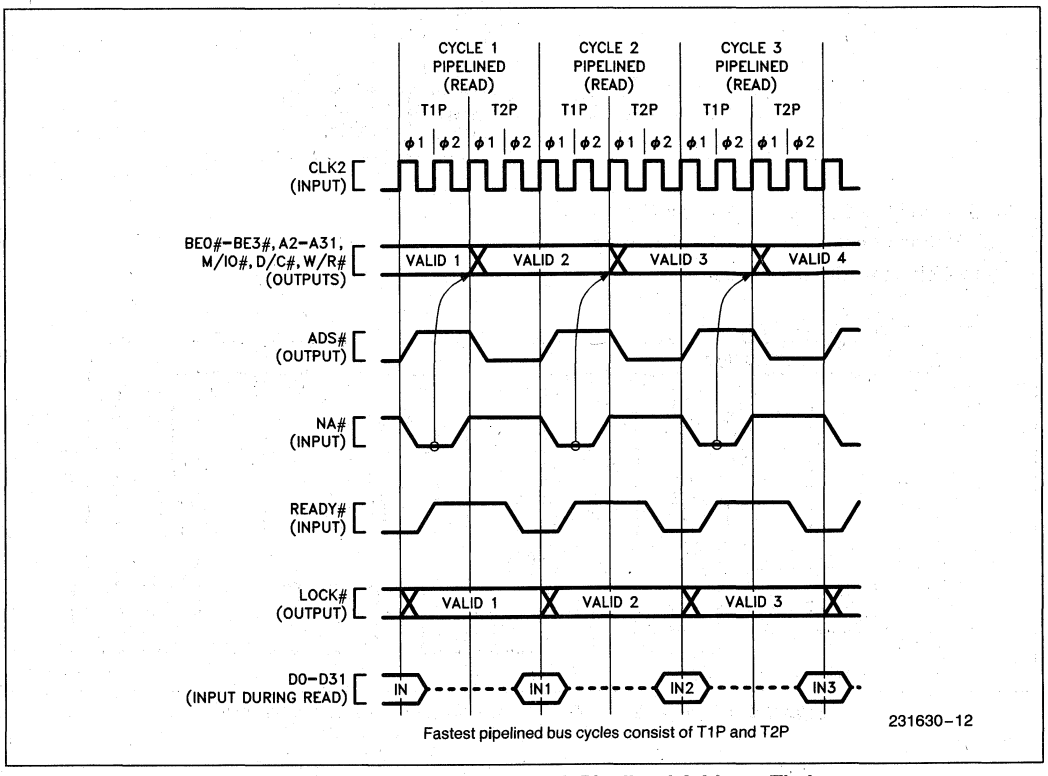


Figure 5-9. Fastest Read Cycles with Pipelined Address Timing

Pipelined address timing is useful in typical systems having address latches. In those systems, once an address has been latched, pipelined availability of the next address allows decoding circuitry to generate chip selects (and other necessary select signals) in advance, so selected devices are accessed immediately when the next cycle begins. In other words, the decode time for the next cycle can be overlapped with the end of the current cycle.

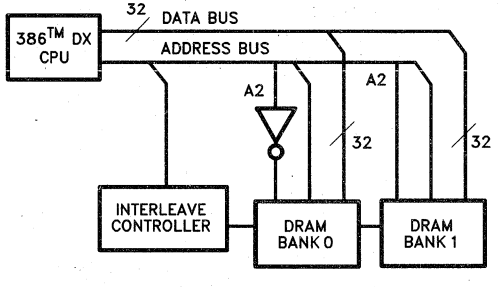
If a system contains a memory structure of two or more interleaved memory banks, pipelined address timing potentially allows even more overlap of activity. This is true when the interleaved memory controller is designed to allow the next memory operation to begin in one memory bank while the current bus cycle is still activating another memory bank. Figure 5-10 shows the general structure of the 386 DX with 2-bank and 4-bank interleaved memory. Note each memory bank of the interleaved memory has full data bus width (32-bit data width typically, unless 16-bit bus size is selected).

Further details of pipelined address timing are given in **5.4.3.4 Pipelined Address, 5.4.3.5 Initiating and Maintaining Pipelined Address, 5.4.3.6 Pipelined Address with Dynamic Bus Sizing,** and **5.4.3.7 Maximum Pipelined Address Usage with 16-Bit Bus Size**.

TWO-BANK INTERLEAVED MEMORY

a) Address signal A2 selects bank

b) 32-bit datapath to each bank

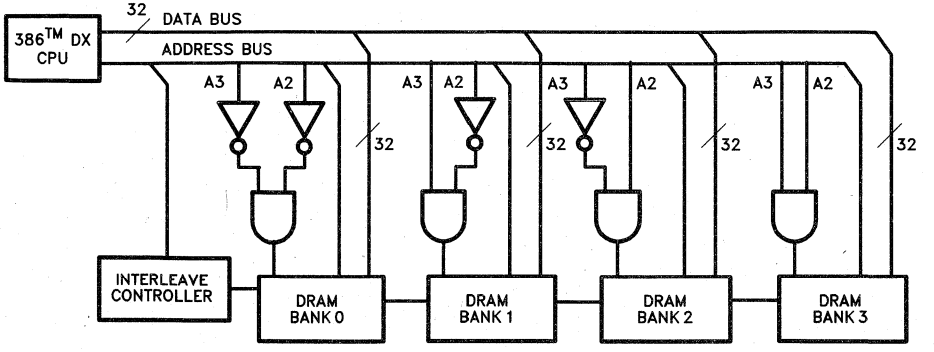


231630-13

FOUR-BANK INTERLEAVED MEMORY

a) Address signals A3 and A2 select bank

b) 32-bit datapath to each bank



231630-14

Figure 5-10. 2-Bank and 4-Bank Interleaved Memory Structure

5.4.3 Read and Write Cycles

5.4.3.1 INTRODUCTION

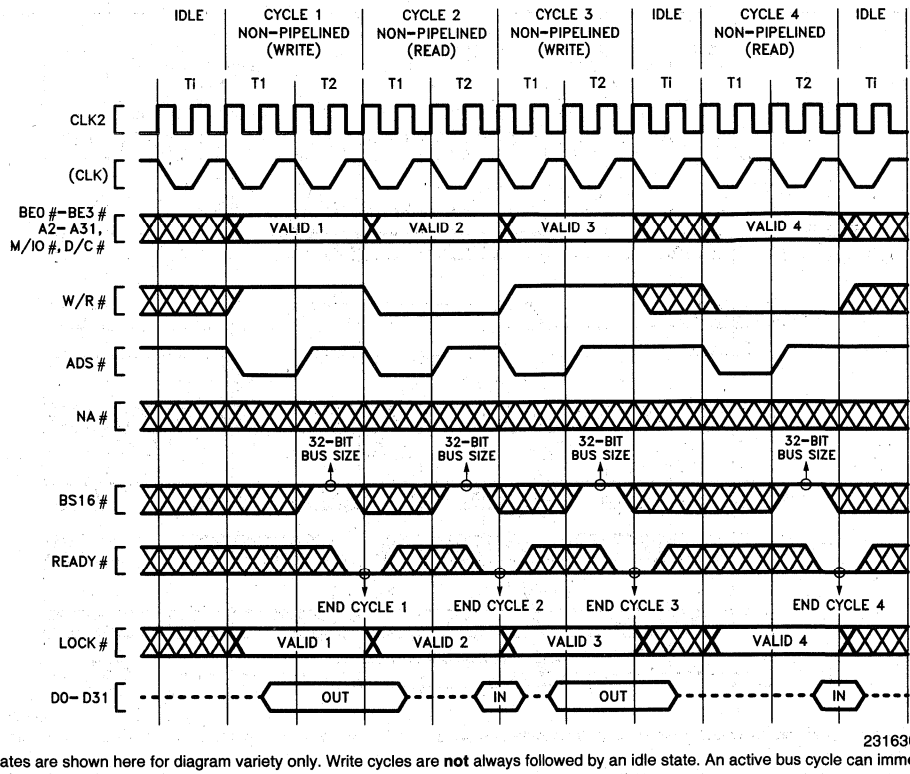
Data transfers occur as a result of bus cycles, classified as read or write cycles. During read cycles, data is transferred from an external device to the processor. During write cycles data is transferred in the other direction, from the processor to an external device.

Two choices of address timing are dynamically selectable: non-pipelined, or pipelined. After a bus idle state, the processor always uses non-pipelined address timing. However, the NA# (Next Address) input may be asserted to select pipelined address timing for the next bus cycle. When pipelining is selected and the 386 DX has a bus request pending internally, the address and definition of the next cycle is made available even before the current bus cycle is acknowledged by READY#. Generally, the NA# input is sampled each bus cycle to select the desired address timing for the next bus cycle.

Two choices of physical data bus width are dynamically selectable: 32 bits, or 16 bits. Generally, the BS16# (Bus Size 16) input is sampled near the end of the bus cycle to confirm the physical data bus size applicable to the current cycle. Negation of BS16# indicates a 32-bit size, and assertion indicates a 16-bit bus size.

If 16-bit bus size is indicated, the 386 DX automatically responds as required to complete the transfer on a 16-bit data bus. Depending on the size and alignment of the operand, another 16-bit bus cycle may be required. Table 5-7 provides all details. When necessary, the 386 DX performs an additional 16-bit bus cycle, using D0–D15 in place of D16–D31.

Terminating a read cycle or write cycle, like any bus cycle, requires acknowledging the cycle by asserting the READY# input. Until acknowledged, the processor inserts wait states into the bus cycle, to allow adjustment for the speed of any external device. External hardware, which has decoded the address and bus cycle type asserts the READY# input at the appropriate time.



231630-15

Idle states are shown here for diagram variety only. Write cycles are **not** always followed by an idle state. An active bus cycle can immediately follow the write cycle.

Figure 5-11. Various Bus Cycles and Idle States with Non-Pipelined Address (zero wait states)

At the end of the second bus state within the bus cycle, READY# is sampled. At that time, if external hardware acknowledges the bus cycle by asserting READY#, the bus cycle terminates as shown in Figure 5-11. If READY# is negated as in Figure 5-12, the cycle continues another bus state (a wait state) and READY# is sampled again at the end of that state. This continues indefinitely until the cycle is acknowledged by READY# asserted.

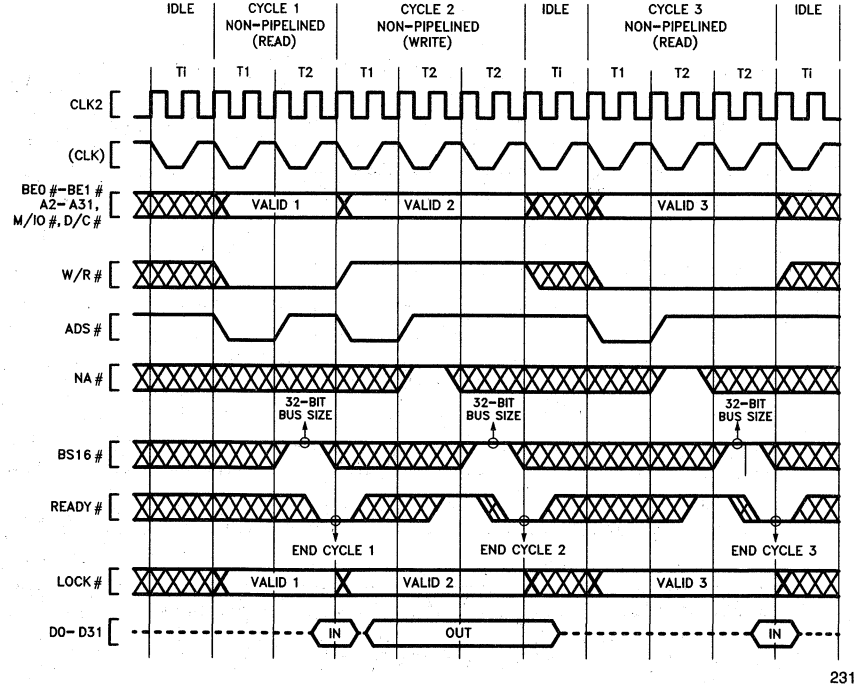
When the current cycle is acknowledged, the 386 DX terminates it. When a read cycle is acknowledged, the 386 DX latches the information present at its data pins. When a write cycle is acknowledged, the 386 DX write data remains valid throughout phase one of the next bus state, to provide write data hold time.

5.4.3.2 NON-PIPELINED ADDRESS

Any bus cycle may be performed with non-pipelined address timing. For example, Figure 5-11 shows a mixture of read and write cycles with non-pipelined address timing. Figure 5-11 shows the fastest possible cycles with non-pipelined address have two bus states per bus cycle. The states are named T1 and T2. In phase one of the T1, the address signals and bus cycle definition signals are driven valid, and to signal their availability, address status (ADS#) is simultaneously asserted.

During read or write cycles, the data bus behaves as follows. If the cycle is a read, the 386 DX floats its data signals to allow driving by the external device being addressed. **The 386 DX requires that all data bus pins be at a valid logic state (high or low) at the end of each read cycle, when READY# is asserted, even if all byte enables are not asserted. The system MUST be designed to meet this requirement.** If the cycle is a write, data signals are driven by the 386 DX beginning in phase two of T1 until phase one of the bus state following cycle acknowledgment.

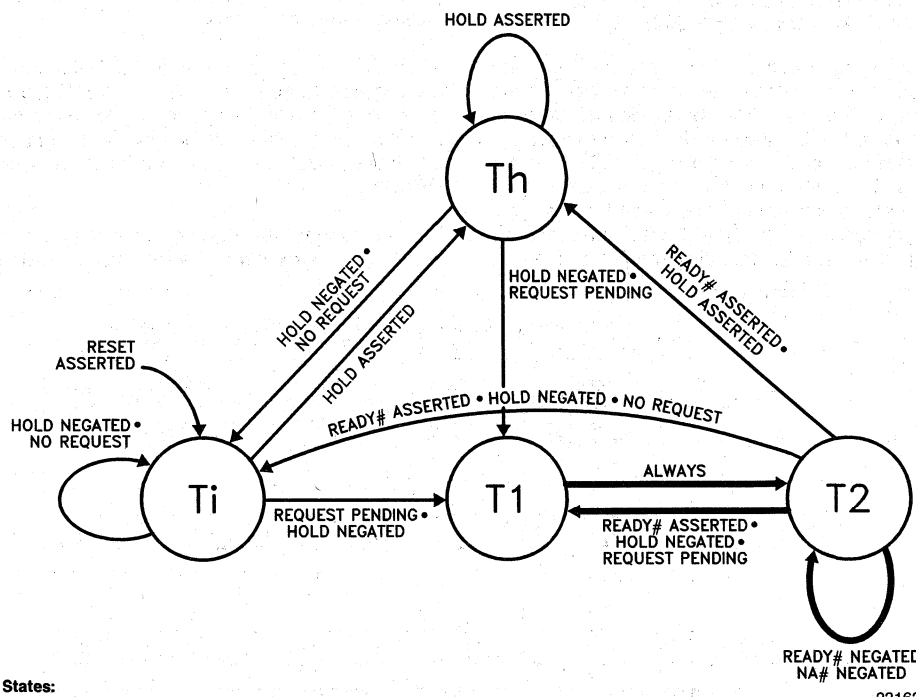
Figure 5-12 illustrates non-pipelined bus cycles with one wait added to cycles 2 and 3. READY# is sampled negated at the end of the first T2 in cycles 2 and 3. Therefore cycles 2 and 3 have T2 repeated. At the end of the second T2, READY# is sampled asserted.



231630-16

Idle states are shown here for diagram variety only. Write cycles are **not** always followed by an idle state. An active bus cycle can immediately follow the write cycle.

Figure 5-12. Various Bus Cycles and Idle States with Non-Pipelined Address (various number of wait states)



Bus States:
T1—first clock of a non-pipelined bus cycle (386™ DX drives new address and asserts ADS#)
T2—subsequent clocks of a bus cycle when NA# has not been sampled asserted in the current bus cycle
Ti— idle state
Th—hold acknowledge state (386™ DX asserts HLDA)
The fastest bus cycle consists of two states: T1 and T2.
Four basic bus states describe bus operation when not using pipelined address. These states do include BS16# usage for 32-bit and 16-bit bus size. If asserting BS16# requires a second 16-bit bus cycle to be performed, it is performed before HOLD asserted is acknowledged.

Figure 5-13. 386™ DX Bus States (not using pipelined address)

When address pipelining is not used, the address and bus cycle definition remain valid during all wait states. When wait states are added and you desire to maintain non-pipelined address timing, it is necessary to negate NA# during each T2 state except the last one, as shown in Figure 5-12 cycles 2 and 3. If NA# is sampled asserted during a T2 other than the last one, the next state would be T2I (for pipelined address) or T2P (for pipelined address) instead of another T2 (for non-pipelined address).

When address pipelining is not used, the bus states and transitions are completely illustrated by Figure 5-13. The bus transitions between four possible states: T1, T2, Ti, and Th. Bus cycles consist of T1 and T2, with T2 being repeated for wait states. Otherwise, the bus may be idle, in the Ti state, or in hold acknowledge, the Th state.

When address pipelining is not used, the bus state diagram is as shown in Figure 5-13. When the bus is idle it is in state Ti. Bus cycles always begin with T1. T1 always leads to T2. If a bus cycle is not acknowledged during T2 and NA# is negated, T2 is repeated. When a cycle is acknowledged during T2, the following state will be T1 of the next bus cycle if a bus request is pending internally, or Ti if there is no bus request pending, or Th if the HOLD input is being asserted.

The bus state diagram in Figure 5-13 also applies to the use of BS16#. If the 386 DX makes internal adjustments for 16-bit bus size, the adjustments do not affect the external bus states. If an additional 16-bit bus cycle is required to complete a transfer on a 16-bit bus, it also follows the state transitions shown in Figure 5-13.

Use of pipelined address allows the 386 DX to enter three additional bus states not shown in Figure 5-13. Figure 5-20 in **5.4.3.4 Pipelined Address** is the complete bus state diagram, including pipelined address cycles.

5.4.3.3 NON-PIPELINED ADDRESS WITH DYNAMIC DATA BUS SIZING

The physical data bus width for any non-pipelined bus cycle can be either 32-bits or 16-bits. At the beginning of the bus cycle, the processor behaves as if the data bus is 32-bits wide. When the bus cycle is acknowledged, by asserting READY# at the end of a T2 state, the most recent sampling of BS16# determines the data bus size for the cycle being acknowledged. If BS16# was most recently negated, the physical data bus size is defined as 32 bits. If BS16# was most recently asserted, the size is defined as 16 bits.

When BS16# is asserted and two 16-bit bus cycles are required to complete the transfer, BS16# must be asserted during the second cycle; 16-bit bus size is not assumed. Like any bus cycle, the second 16-bit cycle must be acknowledged by asserting READY#.

When a second 16-bit bus cycle is required to complete the transfer over a 16-bit bus, the addresses

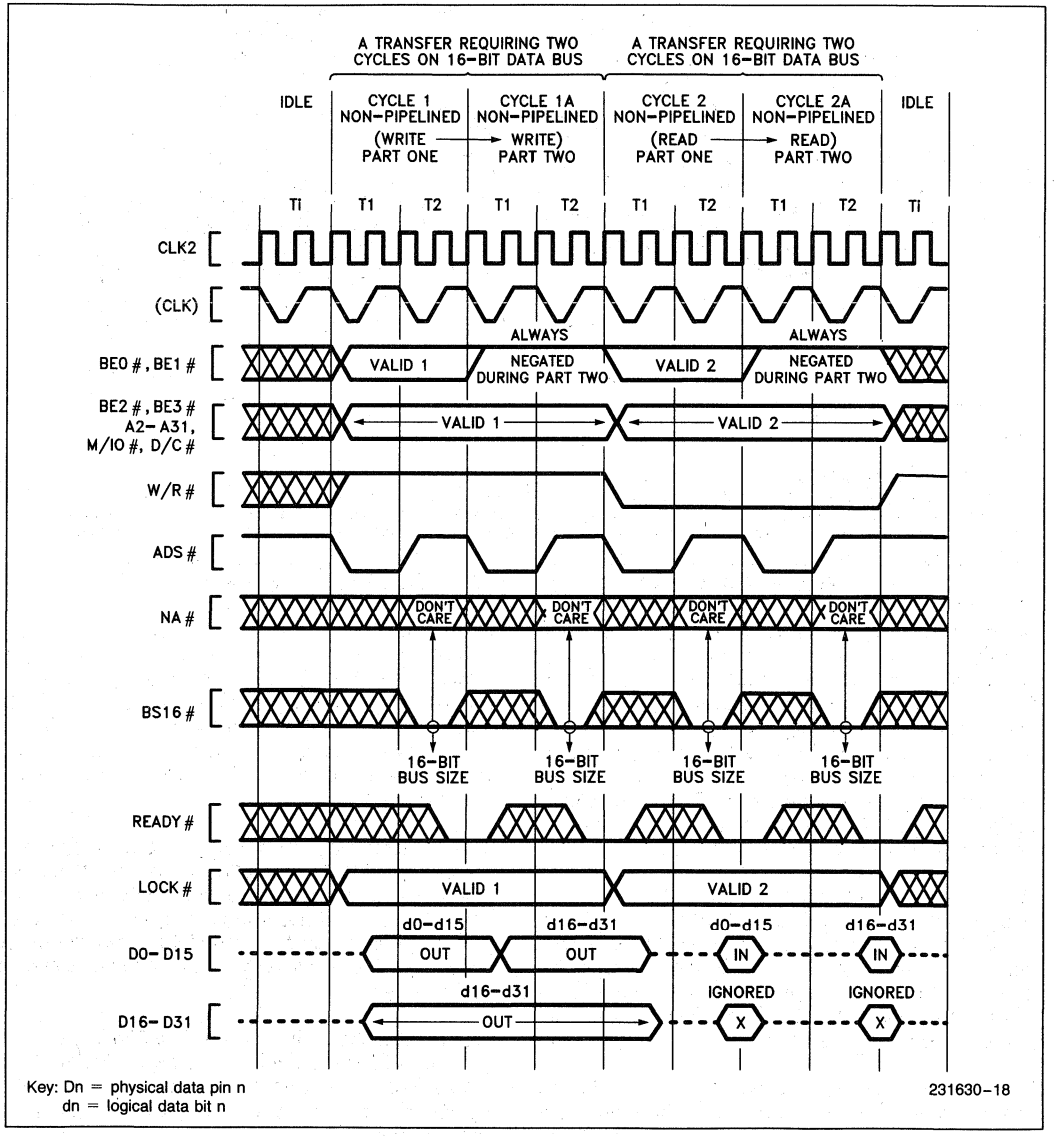


Figure 5-14. Asserting BS16# (zero wait states, non-pipelined address)

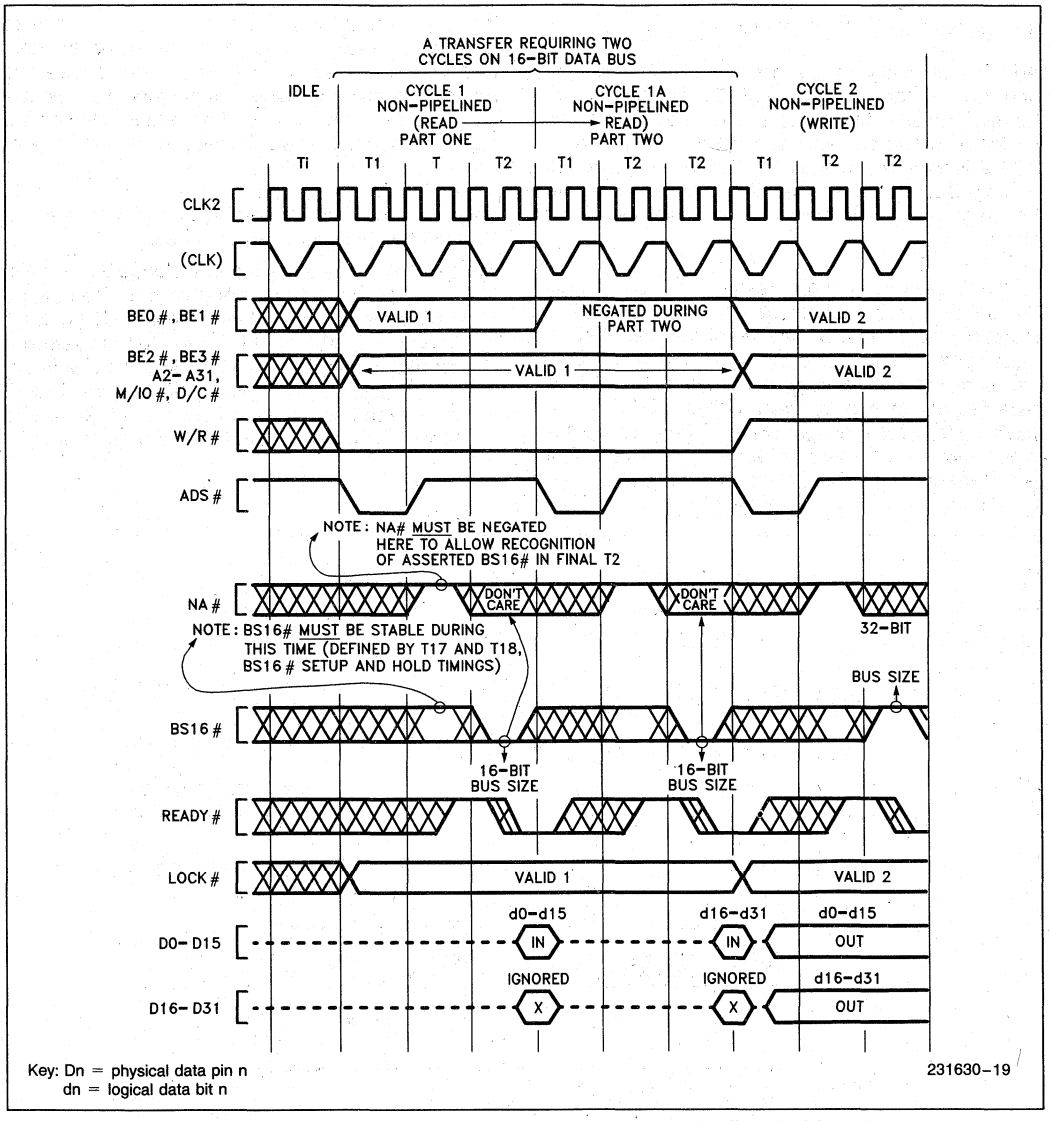


Figure 5-15. Asserting BS16# (one wait state, non-pipelined address)

generated for the two 16-bit bus cycles are closely related to each other. The addresses are the same except BE0# and BE1# are always negated for the second cycle. This is because data on D0–D15 was already transferred during the first 16-bit cycle.

Figures 5-14 and 5-15 show cases where assertion of BS16# requires a second 16-bit cycle for complete operand transfer. Figure 5-14 illustrates cycles without wait states. Figure 5-15 illustrates cycles with one wait state. In Figure 5-15 cycle 1, the bus cycle during which BS16# is asserted, note that NA# must be negated in the T2 state(s) prior to the last T2 state. This is to allow the recognition of BS16# asserted in the final T2 state. Also note that during this state BS16# must be stable (defined by t17 and t18, BS16# setup and hold timings), in order to prevent potential data corruption during split cycle reads. The logic state of BS16# during this time is not important. The relation of NA# and BS16# is given fully in **5.4.3.4 Pipelined Address**, but Figure 5-15 illustrates these precautions you need to know when using BS16# with non-pipelined address.

5.4.3.4 PIPELINED ADDRESS

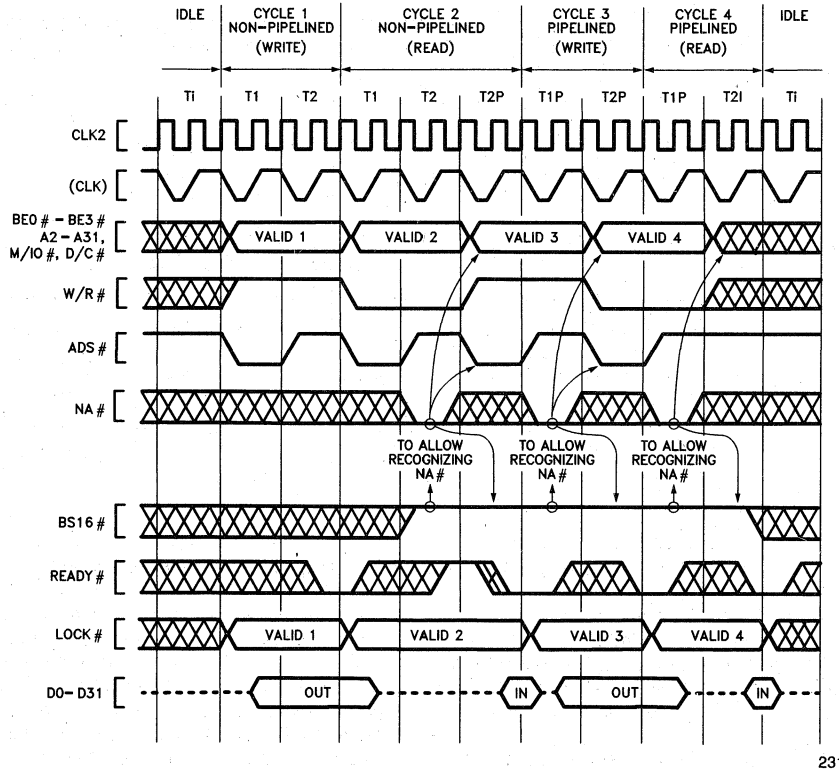
Address pipelining is the option of requesting the address and the bus cycle definition of the next, internally pending bus cycle before the current bus cycle is acknowledged with READY# asserted. ADS# is asserted by the 386 DX when the next address is issued. The address pipelining option is controlled on a cycle-by-cycle basis with the NA# input signal.

Once a bus cycle is in progress and the current address has been valid for at least one entire bus state, the NA# input is sampled at the end of every phase one until the bus cycle is acknowledged. During non-pipelined bus cycles, therefore, NA# is sampled at the end of phase one in every T2. An example is Cycle 2 in Figure 5-16, during which NA# is sampled at the end of phase one of every T2 (it was asserted once during the first T2 and has no further effect during that bus cycle).

If NA# is sampled asserted, the 386 DX is free to drive the address and bus cycle definition of the next bus cycle, and assert ADS#, as soon as it has a bus request internally pending. It may drive the next address as early as the next bus state, whether the current bus cycle is acknowledged at that time or not.

Regarding the details of address pipelining, the 386 DX has the following characteristics:

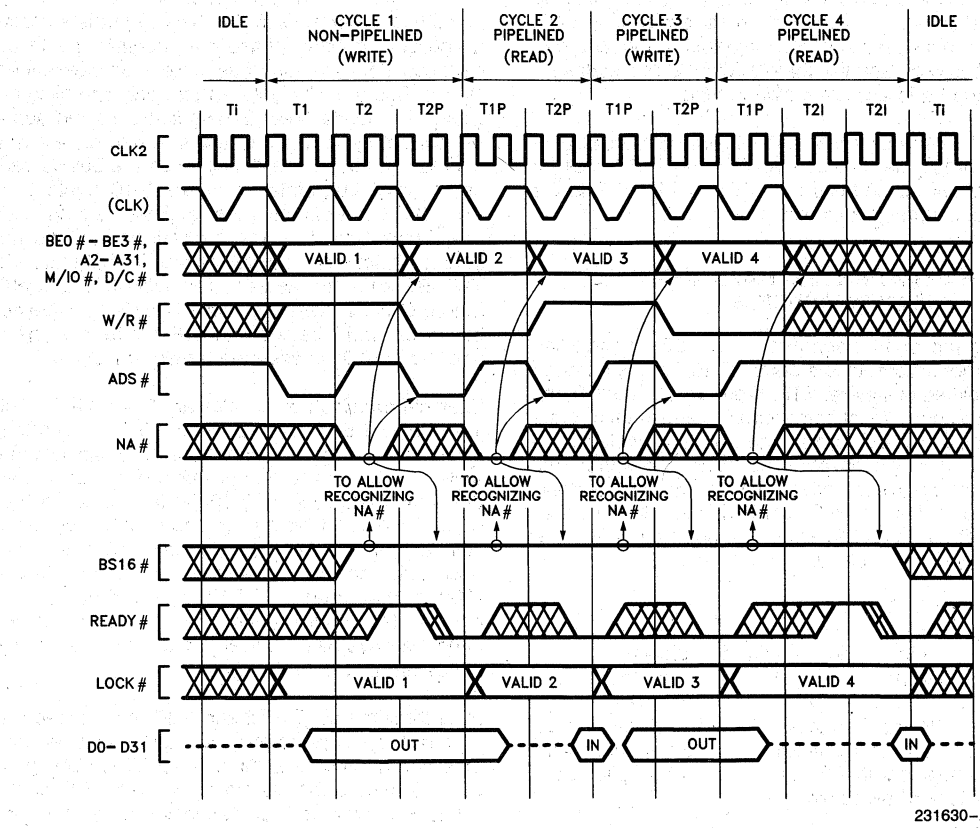
1) For NA# to be sampled asserted, BS16# must be negated at that sampling window (see Figure 5-16 Cycles 2 through 4, and Figure 5-17 Cycles 1 through 4). If NA# and BS16# are both sampled asserted during the last T2 period of a bus cycle, BS16# asserted has priority. Therefore, if both are asserted, the current bus size is taken to be 16 bits and the next address is not pipelined.



231630-20

Following any idle bus state (Ti), addresses are non-pipelined. Within non-pipelined bus cycles, NA# is only sampled during wait states. Therefore, to begin address pipelining during a group of non-pipelined bus cycles requires a non-pipelined cycle with at least one wait state (Cycle 2 above).

Figure 5-16. Transitioning to Pipelined Address During Burst of Bus Cycles



Following any idle bus state (Ti) the address is always non-pipelined and NA# is only sampled during wait states. To start address pipelining after an idle state requires a non-pipelined cycle with at least one wait state (cycle 1 above).
The pipelined cycles (2, 3, 4 above) are shown with various numbers of wait states.

Figure 5-17. Fastest Transition to Pipelined Address Following Idle Bus State

2) The next address may appear as early as the bus state after NA# was sampled asserted (see Figures 5-16 or 5-17). In that case, state T2P is entered immediately. However, when there is not an internal bus request already pending, the next address will not be available immediately after NA# is asserted and T2I is entered instead of T2P (see Figure 5-19 Cycle 3). Provided the current bus cycle isn't yet acknowledged by READY# asserted, T2P will be entered as soon as the 386 DX does drive the next address. External hardware should therefore observe the ADS# output as confirmation the next address is actually being driven on the bus.

3) Once NA# is sampled asserted, the 386 DX commits itself to the highest priority bus request that is pending internally. It can no longer perform another 16-bit transfer to the same address should BS16# be asserted externally, so thereafter must assume the current bus size is 32 bits. Therefore if NA# is sampled asserted within a bus cycle, BS16# must be negated thereafter in that bus cycle (see Figures 5-16, 5-17, 5-19). Consequently, do not assert NA# during bus cycles which must have BS16# driven asserted. See **5.4.3.6 Dynamic Bus Sizing with Pipelined Address.**

4) Any address which is validated by a pulse on the 386 DX ADS# output will remain stable on the address pins for at least two processor clock periods. The 386 DX cannot produce a new address more frequently than every two processor clock periods (see Figures 5-16, 5-17, 5-19).

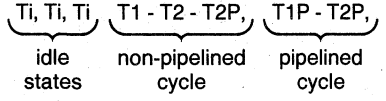
5) Only the address and bus cycle definition of the very next bus cycle is available. The pipelining capability cannot look further than one bus cycle ahead (see Figure 5-19 Cycle 1).

The complete bus state transition diagram, including operation with pipelined address is given by 5-20. Note it is a superset of the diagram for non-pipelined address only, and the three additional bus states for pipelined address are drawn in bold.

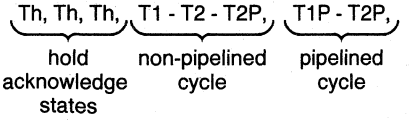
The fastest bus cycle with pipelined address consists of just two bus states, T1P and T2P (recall for non-pipelined address it is T1 and T2). T1P is the first bus state of a pipelined cycle.

5.4.3.5 INITIATING AND MAINTAINING PIPELINED ADDRESS

Using the state diagram Figure 5-20, observe the transitions from an idle state, Ti, to the beginning of a pipelined bus cycle, T1P. From an idle state Ti, the first bus cycle must begin with T1, and is therefore a non-pipelined bus cycle. The next bus cycle will be pipelined, however, provided NA# is asserted and the first bus cycle ends in a T2P state (the address for the next bus cycle is driven during T2P). The fastest path from an idle state to a bus cycle with pipelined address is shown in bold below:



T1-T2-T2P are the states of the bus cycle that establishes address pipelining for the next bus cycle, which begins with T1P. The same is true after a bus hold state, shown below:



The transition to pipelined address is shown functionally by Figure 5-17 Cycle 1. Note that Cycle 1 is used to transition into pipelined address timing for the subsequent Cycles 2, 3 and 4, which are pipelined. The NA# input is asserted at the appropriate time to select address pipelining for Cycles 2, 3 and 4.

Once a bus cycle is in progress and the current address has become valid, the NA# input is sampled at the end of every phase one, beginning with the next bus state, until the bus cycle is acknowledged. During Figure 5-17 Cycle 1 therefore, sampling begins in T2. Once NA# is sampled asserted during the current cycle, the 386 DX is free to drive a new address and bus cycle definition on the bus as early as the next bus state. In Figure 5-16 Cycle 1 for example, the next address is driven during state T2P. Thus Cycle 1 makes the transition to pipelined address timing, since it begins with T1 but ends with T2P. Because the address for Cycle 2 is available before Cycle 2 begins, Cycle 2 is called a pipelined bus cycle, and it begins with T1P. Cycle 2 begins as soon as READY# asserted terminates Cycle 1.

Example transition bus cycles are Figure 5-17 Cycle 1 and Figure 5-16 Cycle 2. Figure 5-17 shows transition during the very first cycle after an idle bus state, which is the fastest possible transition into address pipelining. Figure 5-16 Cycle 2 shows a transition cycle occurring during a burst of bus cycles. In any case, a transition cycle is the same whenever it occurs: it consists at least of T1, T2 (you assert NA# at that time), and T2P (provided the 386 DX has an internal bus request already pending, which it almost always has). T2P states are repeated if wait states are added to the cycle.

Note three states (T1, T2 and T2P) are only required in a bus cycle performing a **transition** from non-pipelined address into pipelined address timing, for example Figure 5-17 Cycle 1. Figure 5-17 Cycles 2, 3 and 4 show that address pipelining can be maintained with two-state bus cycles consisting only of T1P and T2P.

Once a pipelined bus cycle is in progress, pipelined timing is maintained for the next cycle by asserting NA# and detecting that the 386 DX enters T2P during the current bus cycle. The current bus cycle must end in state T2P for pipelining to be maintained in the next cycle. T2P is identified by the assertion of ADS#. Figures 5-16 and 5-17 however, each show pipelining ending after Cycle 4 because Cycle 4 ends in T2I. This indicates the 386 DX didn't have an internal bus request prior to the acknowledgement of Cycle 4. If a cycle ends with a T2 or T2I, the next cycle will not be pipelined.

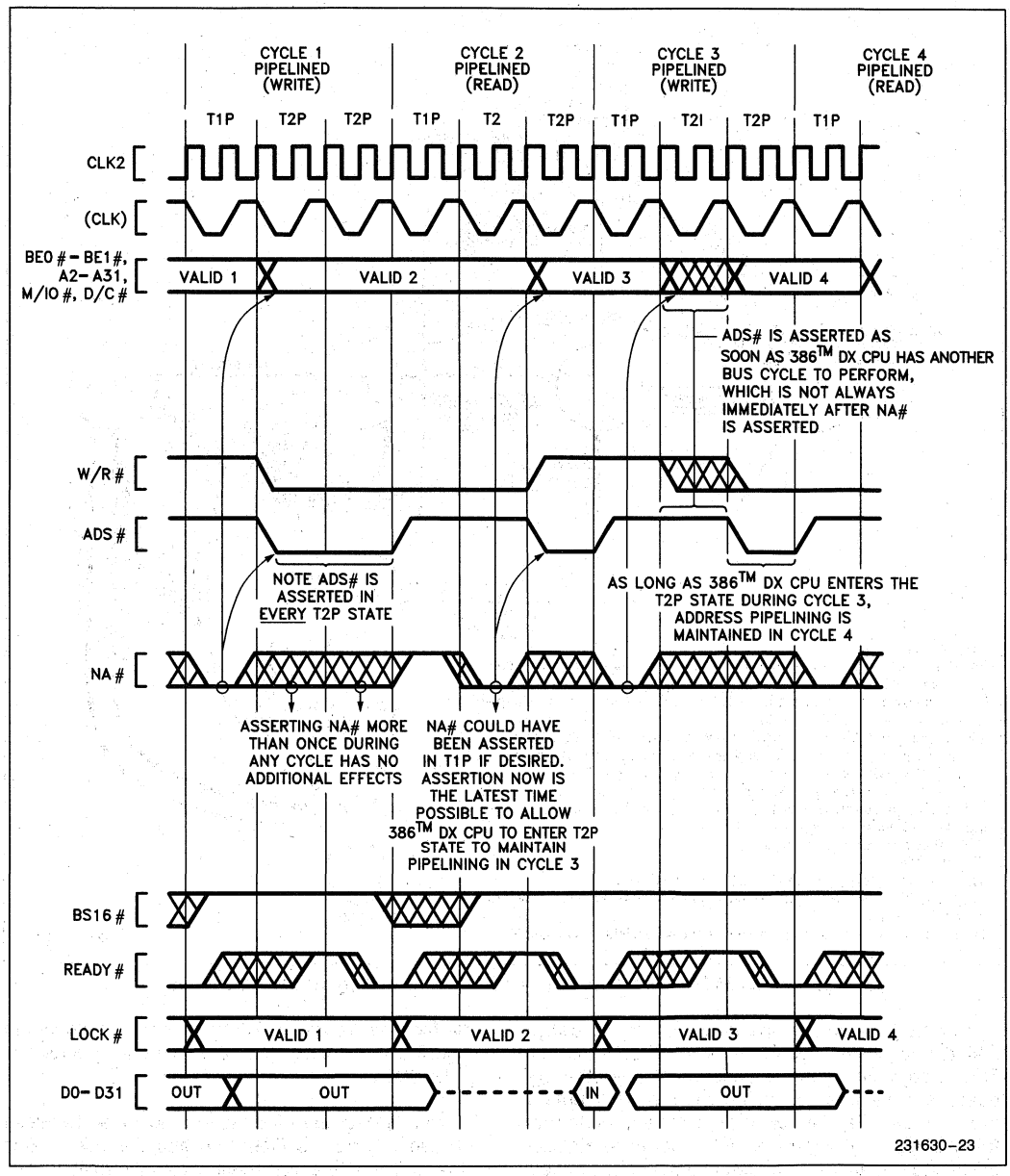


Figure 5-19. Details of Address Pipelining During Cycles with Wait States

5

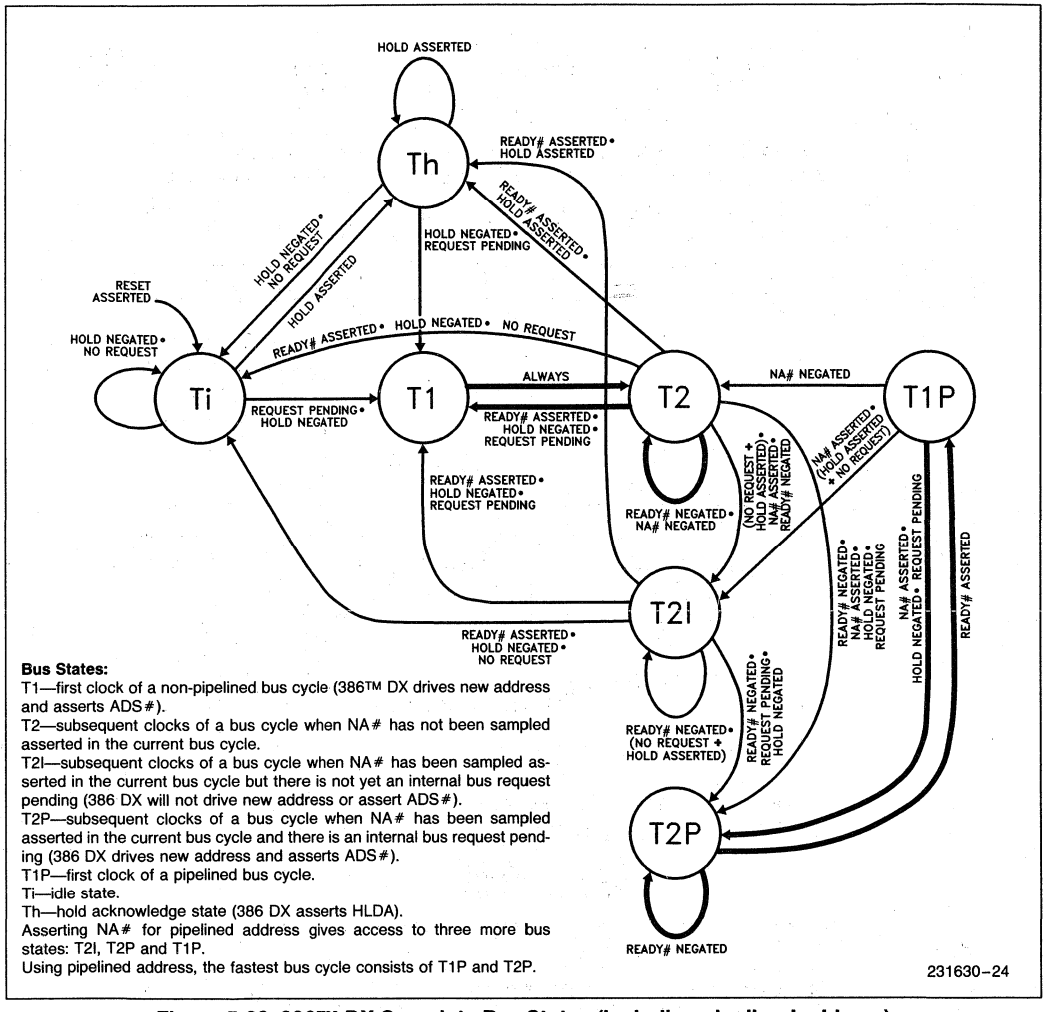


Figure 5-20. 386™ DX Complete Bus States (including pipelined address)

Realistically, address pipelining is almost always maintained as long as NA# is sampled asserted. This is so because in the absence of any other request, a code prefetch request is always internally pending until the instruction decoder and code prefetch queue are completely full. Therefore address pipelining is maintained for long bursts of bus cycles, if the bus is available (i.e., HOLD negated) and NA# is sampled asserted in each of the bus cycles.

5.4.3.6 PIPELINED ADDRESS WITH DYNAMIC DATA BUS SIZING

The BS16# feature allows easy interface to 16-bit data buses. When asserted, the 386 DX bus interface hardware performs appropriate action to make the transfer using a 16-bit data bus connected on D0–D15.

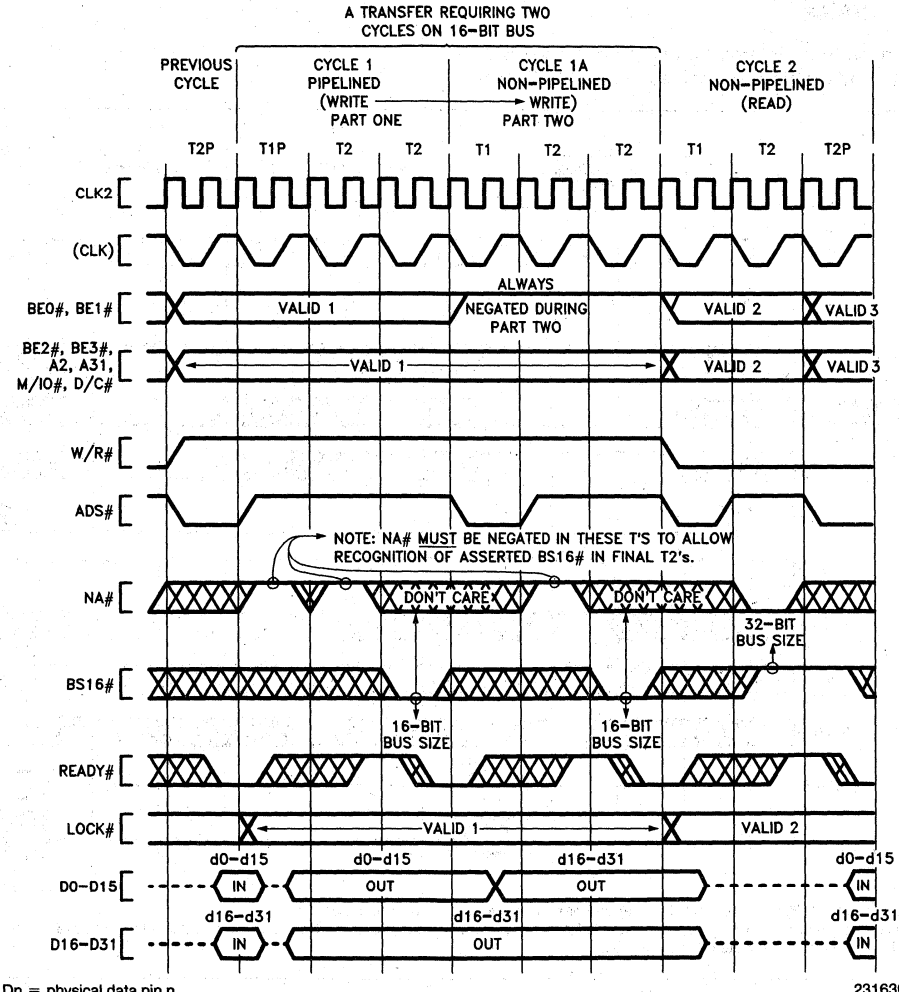
There is a degree of interaction, however, between the use of Address Pipelining and the use of Bus Size 16. The interaction results from the multiple bus cycles required when transferring 32-bit operands over a 16-bit bus. If the operand requires both 16-bit halves of the 32-bit bus, the appropriate 386 DX action is a second bus cycle to complete the operand's transfer. It is this necessity that conflicts with NA# usage.

When NA# is sampled asserted, the 386 DX commits itself to perform the next inter-

nally pending bus request, and is allowed to drive the next internally pending address onto the bus. Asserting NA# therefore makes it impossible for the next bus cycle to again access the current address on A2–A31, such as may be required when BS16# is asserted by the external hardware.

To avoid conflict, the 386 DX is designed with following two provisions:

1) To avoid conflict, BS16# must be negated in the current bus cycle if NA# has already been sampled asserted in the current cycle. If NA# is sampled asserted, the current data bus size is assumed to be 32 bits.
2) To also avoid conflict, if NA# and BS16# are both asserted during the same sampling window, BS16# asserted has priority and the 386 DX acts as if NA# was negated at that time. Internal 386 DX circuitry, shown conceptually in Figure 5-18, assures that BS16# is sampled asserted and NA# is sampled negated if both inputs are externally asserted at the same sampling window.



Key: Dn = physical data pin n
dn = logical data bit n

231630-25

Cycle 1 is pipelined. Cycle 1a cannot be pipelined, but its address can be inferred from that of Cycle 1, to externally simulate address pipelining during Cycle 1a.

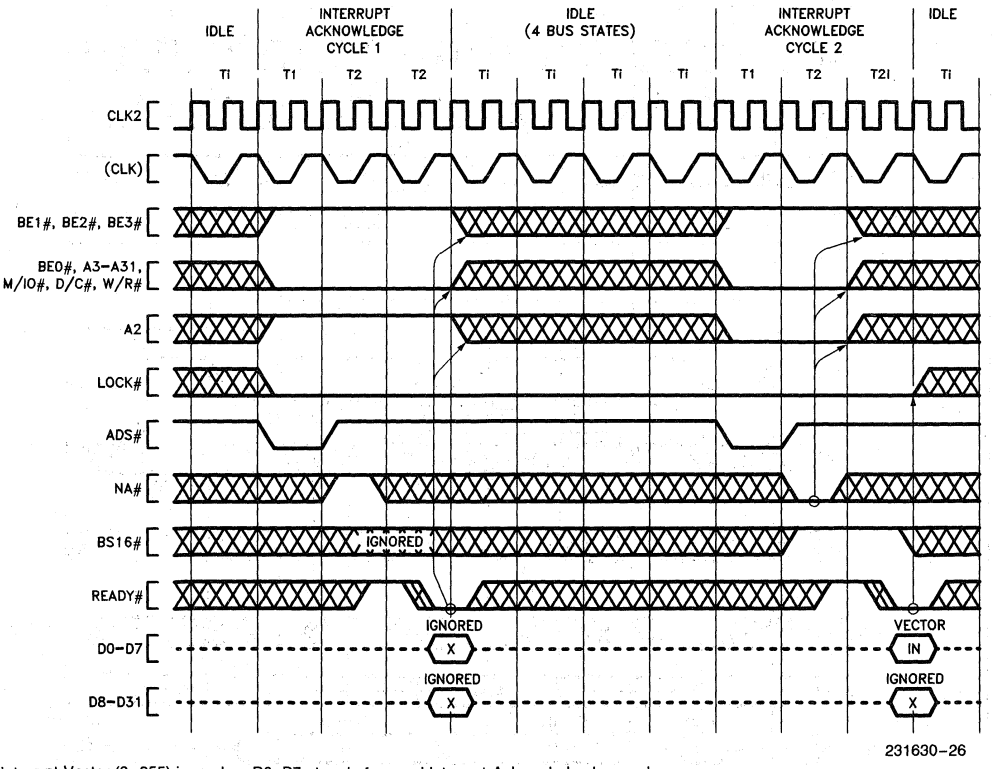
Figure 5-21. Using NA# and BS16#

Certain types of 16-bit or 8-bit operands require no adjustment for correct transfer on a 16-bit bus. Those are read or write operands using only the lower half of the data bus, and write operands using only the upper half of the bus since the 386 DX simultaneously duplicates the write data on the lower half of the data bus. For these patterns of Byte Enables and the R/W# signals, BS16# need not be asserted at the 386 DX allowing NA# to be asserted during the bus cycle if desired.

5.4.4 Interrupt Acknowledge (INTA) Cycles

In response to an interrupt request on the INTR input when interrupts are enabled, the 386 DX performs two interrupt acknowledge cycles. These bus cycles are similar to read cycles in that bus definition signals define the type of bus activity taking place, and each cycle continues until acknowledged by READY# sampled asserted.

The state of A2 distinguishes the first and second interrupt acknowledge cycles. The byte address driven during the first interrupt acknowledge cycle is 4 (A31–A3 low, A2 high, BE3#–BE1# high, and BE0# low). The address driven during the second interrupt acknowledge cycle is 0 (A31–A2 low, BE3#–BE1# high, BE0# low).



Interrupt Vector (0–255) is read on D0–D7 at end of second Interrupt Acknowledge bus cycle.
Because each Interrupt Acknowledge bus cycle is followed by idle bus states, asserting NA# has no practical effect. Choose the approach which is simplest for your system hardware design.

Figure 5-22. Interrupt Acknowledge Cycles

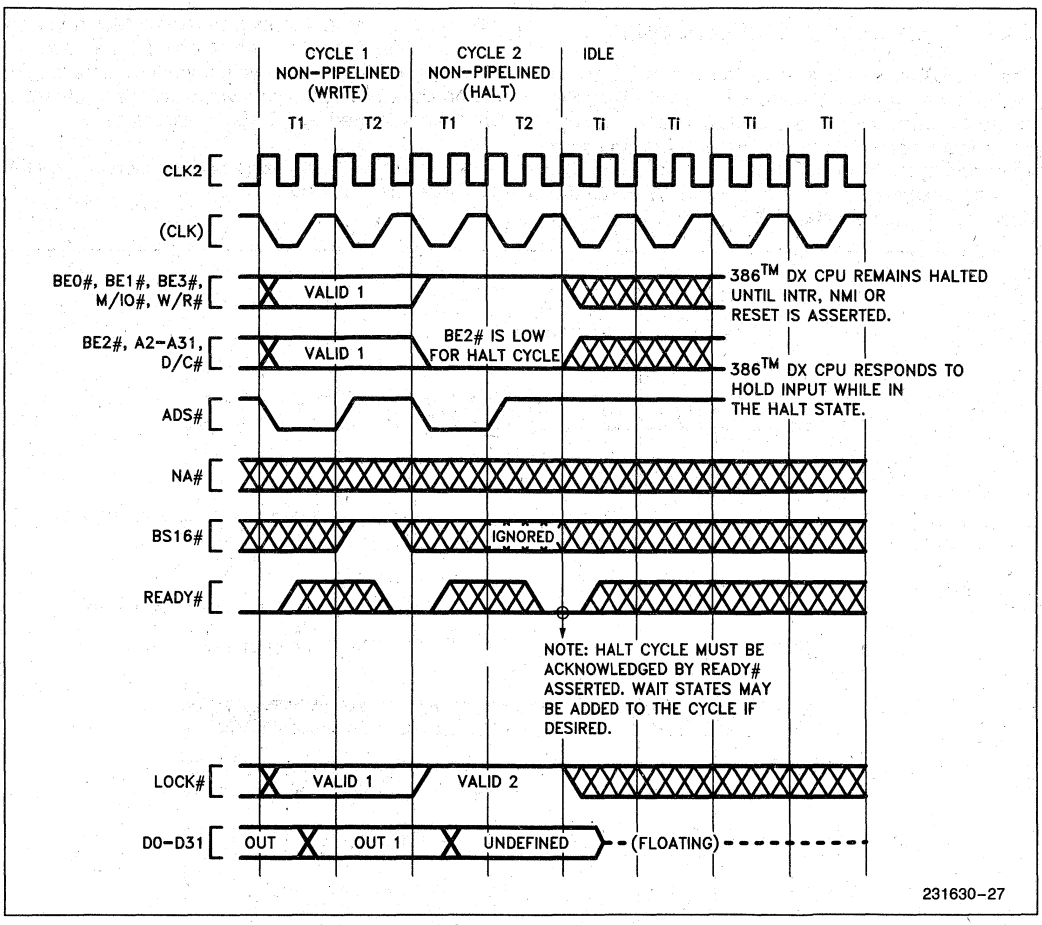


Figure 5-23. Halt Indication Cycle

The LOCK# output is asserted from the beginning of the first interrupt acknowledge cycle until the end of the second interrupt acknowledge cycle. Four idle bus states, Ti, are inserted by the 386 DX between the two interrupt acknowledge cycles, allowing for compatibility with spec TRHRL of the 8259A Interrupt Controller.

During both interrupt acknowledge cycles, D0–D31 float. No data is read at the end of the first interrupt acknowledge cycle. At the end of the second interrupt acknowledge cycle, the 386 DX will read an external interrupt vector from D0–D7 of the data bus. The vector indicates the specific interrupt number (from 0–255) requiring service.

5.4.5 Halt Indication Cycle

The 386 DX halts as a result of executing a HALT instruction. Signaling its entrance into the halt state, a halt indication cycle is performed. The halt indication cycle is identified by the state of the bus definition signals shown in **5.2.5 Bus Cycle Definition** and a byte address of 2. BE0# and BE2# are the only signals distinguishing halt indication from shutdown indication, which drives an address of 0. During the halt cycle undefined data is driven on D0–D31. The halt indication cycle must be acknowledged by READY# asserted.

A halted 386 DX resumes execution when INTR (if interrupts are enabled) or NMI or RESET is asserted.

5.4.6 Shutdown Indication Cycle

The 386 DX shuts down as a result of a protection fault while attempting to process a double fault. Signaling its entrance into the shutdown state, a shutdown indication cycle is performed. The shutdown indication cycle is identified by the state of the bus definition signals shown in **5.2.5 Bus Cycle Definition** and a byte address of 0. BE0# and BE2# are the only signals distinguishing shutdown indication from halt indication, which drives an address of 2. During the shutdown cycle undefined data is driven on D0–D31. The shutdown indication cycle must be acknowledged by READY# asserted.

A shutdown 386 DX resumes execution when NMI or RESET is asserted.

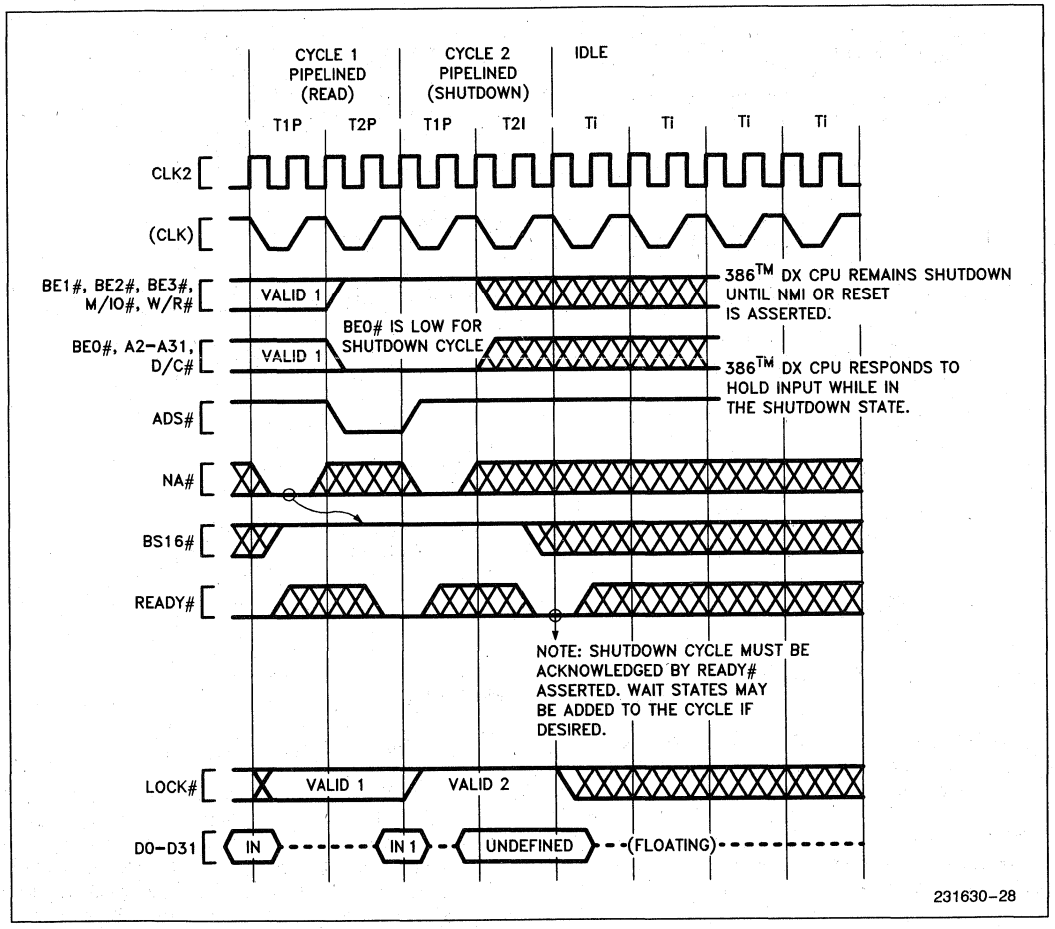
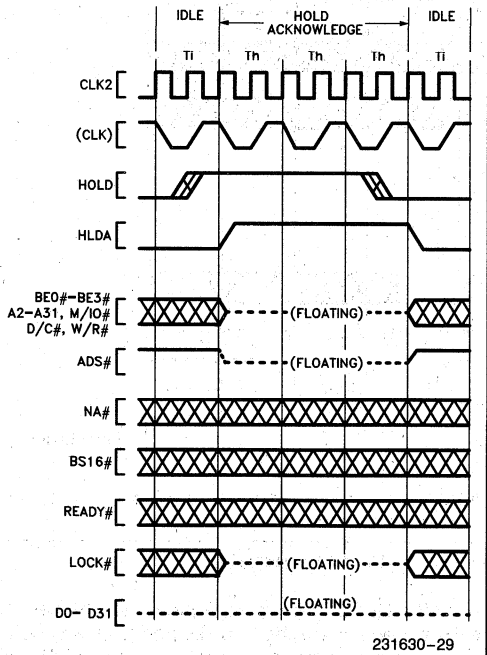


Figure 5-24. Shutdown Indication Cycle

5.5 OTHER FUNCTIONAL DESCRIPTIONS

5.5.1 Entering and Exiting Hold Acknowledge

The bus hold acknowledge state, Th, is entered in response to the HOLD input being asserted. In the bus hold acknowledge state, the 386 DX floats all output or bidirectional signals, except for HLDA. HLDA is asserted as long as the 386 DX remains in the bus hold acknowledge state. In the bus hold acknowledge state, all inputs except HOLD, RESET, BUSY#, ERROR#, and PEREQ are ignored (also up to one rising edge on NMI is remembered for processing when HOLD is no longer asserted).



NOTE:
For maximum design flexibility the 386™ DX has no internal pullup resistors on its outputs. Your design may require an external pullup on ADS# and other 386 DX outputs to keep them negated during float periods.

Figure 5-25. Requesting Hold from Idle Bus

Th may be entered from a bus idle state as in Figure 5-25 or after the acknowledgement of the current physical bus cycle if the LOCK# signal is not asserted, as in Figures 5-26 and 5-27. If HOLD is asserted during a locked bus cycle, the 386 DX may execute one unlocked bus cycle before acknowledging HOLD. If asserting BS16# requires a second 16-bit bus cycle to complete a physical operand transfer, it is performed before HOLD is acknowledged, although the bus state diagrams in Figures 5-13 and 5-20 do not indicate that detail.

Th is exited in response to the HOLD input being negated. The following state will be Ti as in Figure 5-25 if no bus request is pending. The following bus state will be T1 if a bus request is internally pending, as in Figures 5-26 and 5-27.

Th is also exited in response to RESET being asserted.

If a rising edge occurs on the edge-triggered NMI input while in Th, the event is remembered as a nonmaskable interrupt 2 and is serviced when Th is exited, unless of course, the 386 DX is reset before Th is exited.

5.5.2 Reset During Hold Acknowledge

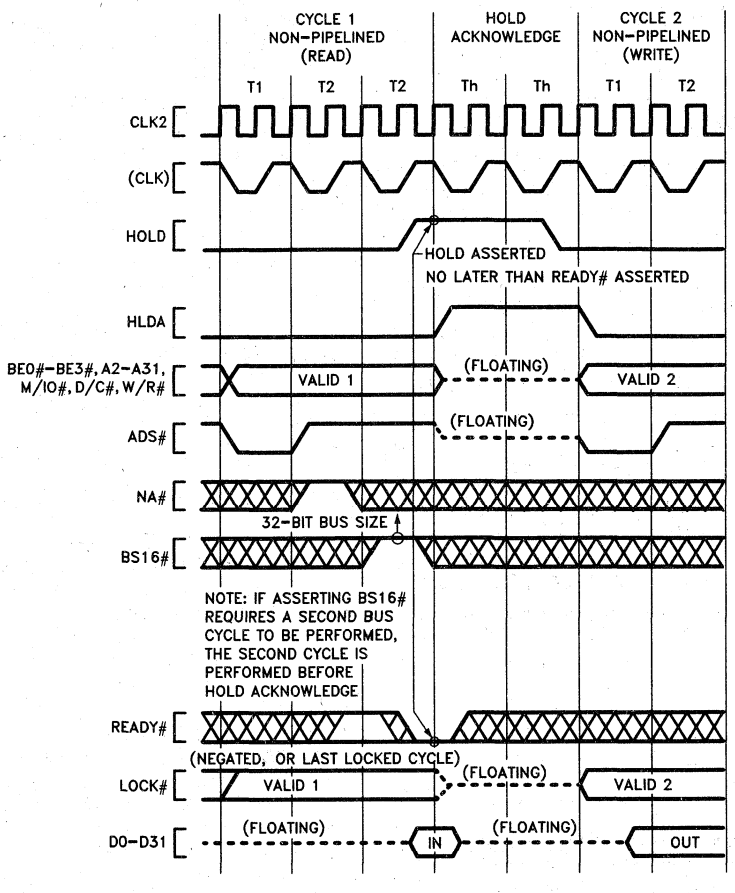
RESET being asserted takes priority over HOLD being asserted. Therefore, Th is exited in reponse to the RESET input being asserted. If RESET is asserted while HOLD remains asserted, the 386 DX drives its pins to defined states during reset, as in **Table 5-3 Pin State During Reset**, and performs internal reset activity as usual.

If HOLD remains asserted when RESET is negated, the 386 DX enters the hold acknowledge state before performing its first bus cycle, provided HOLD is still asserted when the 386 DX would otherwise perform its first bus cycle. If HOLD remains asserted when RESET is negated, the BUSY# input is still sampled as usual to determine whether a self test is being requested, and ERROR# is still sampled as usual to determine whether a 387 DX coprocessor vs. an 80287 (or none) is present.

5.5.3 Bus Activity During and Following Reset

RESET is the highest priority input signal, capable of interrupting any processor activity when it is asserted. A bus cycle in progress can be aborted at any stage, or idle states or bus hold acknowledge states discontinued so that the reset state is established.

RESET should remain asserted for at least 15 CLK2 periods to ensure it is recognized throughout the 386 DX, and at least 80 CLK2 periods if 386 DX self-test is going to be requested at the falling edge. RESET asserted pulses less than 15 CLK2 periods may not be recognized. RESET pulses less than 80 CLK2 periods followed by a self-test may cause the self-test to report a failure when no true failure exists.



NOTE:
HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold (t_{23} and t_{24}) requirements are met. This waveform is useful for determining Hold Acknowledge latency.

Figure 5-26. Requesting Hold from Active Bus (NA# negated)

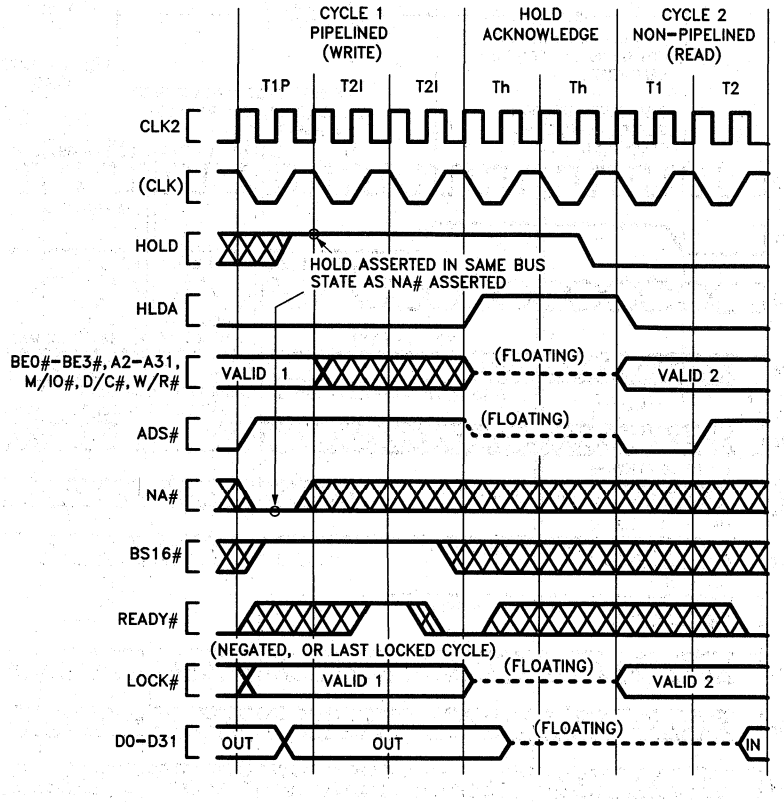
The additional RESET pulse width is required to clear additional state prior to a valid self-test.

Provided the RESET falling edge meets setup and hold times t_{25} and t_{26}, the internal processor clock phase is defined at that time, as illustrated by Figure 5-28 and Figure 7-7.

A 386 DX self-test may be requested at the time RESET is negated by having the BUSY# input at a LOW level, as shown in Figure 5-28. The self-test requires (2^{20}) + approximately 60 CLK2 periods to complete. The self-test duration is not affected by the test results. Even if the self-test indicates a problem, the 386 DX attempts to proceed with the reset sequence afterwards.

After the RESET falling edge (and after the self-test if it was requested) the 386 DX performs an internal initialization sequence for approximately 350 to 450 CLK2 periods.

The 386 DX samples its ERROR# input some time after the falling edge of RESET and before executing the first ESC instruction. During this sampling period BUSY# must be HIGH. If ERROR# was sampled active, the 386 DX employs the 32-bit protocol of the 387 DX. Even though this protocol was selected, it is still necessary to use a software recognition test to determine the presence or identity of the coprocessor and to assure compatibility with future processors. (See Chapter 11 of the 386™ DX Programmer's Reference Manual, Order #230985-002).



NOTE:
HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold (t_{23} and t_{24}) requirements are met. This waveform is useful for determining Hold Acknowledge latency.

Figure 5-27. Requesting Hold from Active Bus (NA# asserted)

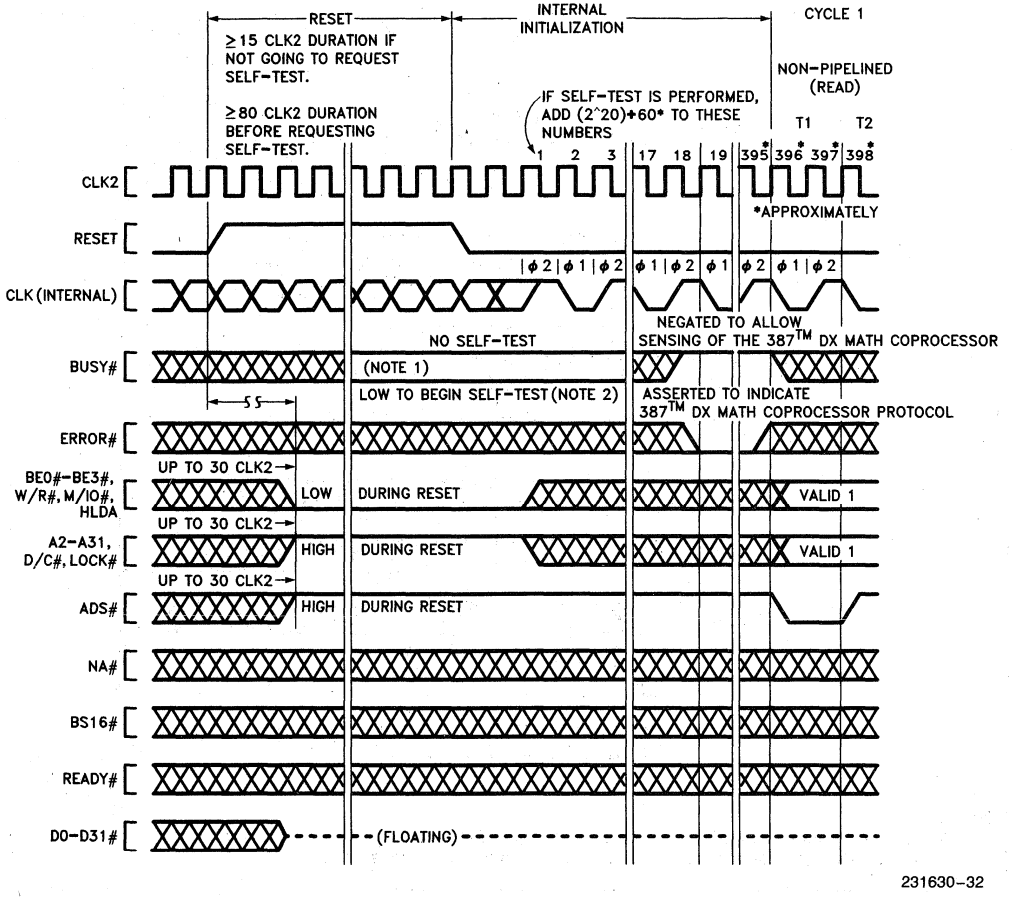
5.6 SELF-TEST SIGNATURE

Upon completion of self-test, (if self-test was requested by holding BUSY# LOW at least eight CLK2 periods before and after the falling edge of RESET), the EAX register will contain a signature of 00000000h indicating the 386 DX passed its self-test of microcode and major PLA contents with no problems detected. The passing signature in EAX, 00000000h, applies to all 386 DX revision levels. Any non-zero signature indicates the 386 DX unit is faulty.

5.7 COMPONENT AND REVISION IDENTIFIERS

To assist 386 DX users, the 386 DX after reset holds a component identifier and a revision identifier in its DX register. The upper 8 bits of DX hold 03h as identification of the 386 DX component. The lower 8 bits of DX hold an 8-bit unsigned binary number related to the component revision level. The revision identifier begins chronologically with a value zero and is subject to change (typically it will be incremented) with component steppings intended to have certain improvements or distinctions from previous steppings.

These features are intended to assist 386 DX users to a practical extent. However, the revision identifier value is not guaranteed to change with every stepping revision, or to follow a completely uniform numerical sequence, depending on the type or intention of revision, or manufacturing materials required to be changed. Intel has sole discretion over these characteristics of the component.



NOTES:
1. BUSY# should be held stable for 8 CLK2 periods before and after the CLK2 period in which RESET falling edge occurs.
2. If self-test is requested, the 386™ DX outputs remain in their reset state as shown here and in Table 5-3.

Figure 5-28. Bus Activity from Reset Until First Code Fetch

Table 5-10. Component and Revision Identifier History

386™ DX Stepping Name	Component Identifier	Revision Identifier	386™ DX Stepping Name	Component Identifier	Revision Identifier
B0	03	03	D0	03	05
B1	03	03	D1	03	08

5.8 COPROCESSOR INTERFACING

The 386 DX provides an automatic interface for the Intel 387 DX numeric floating-point coprocessor. The 387 DX coprocessor uses an I/O-mapped interface driven automatically by the 386 DX and assisted by three dedicated signals: BUSY#, ERROR#, and PEREQ.

As the 386 DX begins supporting a coprocessor instruction, it tests the BUSY# and ERROR# signals to determine if the coprocessor can accept its next instruction. Thus, the BUSY# and ERROR# inputs eliminate the need for any "preamble" bus cycles for communication between processor and coprocessor. The 387 DX can be given its command opcode immediately. The dedicated signals provide instruction synchronization, and eliminate the need of using the 386 DX WAIT opcode (9Bh) for 387 DX coprocessor instruction synchronization (the WAIT opcode was required when 8086 or 8088 was used with the 8087 coprocessor).

Custom coprocessors can be included in 386 DX-based systems, via memory-mapped or I/O-mapped interfaces. Such coprocessor interfaces allow a completely custom protocol, and are not limited to a set of coprocessor protocol "primitives". Instead, memory-mapped or I/O-mapped interfaces may use all applicable 386 DX instructions for high-speed coprocessor communication. The BUSY# and ERROR# inputs of the 386 DX may also be used for the custom coprocessor interface, if such hardware assist is desired. These signals can be tested by the 386 DX WAIT opcode (9Bh). The WAIT instruction will wait until the BUSY# input is negated (interruptable by an NMI or enabled INTR input), but generates an exception 16 fault if the ERROR# pin is in the asserted state when the BUSY# goes (or is) negated. If the custom coprocessor interface is memory-mapped, protection of the addresses used for the interface can be provided with the 386 DX on-chip paging or segmentation mechanisms. If the custom interface is I/O-mapped, protection of the interface can be provided with the 386 DX IOPL (I/O Privilege Level) mechanism.

The 387 DX numeric coprocessor interface is I/O mapped as shown in Table 5-11. Note that the 387 DX coprocessor interface addresses are beyond the 0h-FFFFh range for programmed I/O. When the 386 DX supports the 387 DX coprocessor, the 386 DX automatically generates bus cycles to the coprocessor interface addresses.

Table 5-11. Numeric Coprocessor Port Addresses

Address in 386™ DX I/O Space	387™DX Coprocessor Register
800000F8h	Opcode Register (32-bit port)
800000FCh	Operand Register (32-bit port)

To correctly map the 387 DX coprocessor registers to the appropriate I/O addresses, connect the 387 DX coprocessor CMD0# pin directly to the A2 output of the 386 DX.

5.8.1 Software Testing for Coprocessor Presence

When software is used to test for coprocessor (387 DX) presence, it should use only the following coprocessor opcodes: FINIT, FNINIT, FSTCW mem, FSTSW mem, FSTSW AX. To use other coprocessor opcodes when a coprocessor is known to be not present, first set EM = 1 in 386 DX CR0.

6. INSTRUCTION SET

This section describes the 386 DX instruction set. A table lists all instructions along with instruction encoding diagrams and clock counts. Further details of the instruction encoding are then provided in the following sections, which completely describe the encoding structure and the definition of all fields occurring within 386 DX instructions.

6.1 386™ DX INSTRUCTION ENCODING AND CLOCK COUNT SUMMARY

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in Table 6-1 below, by the processor clock period (e.g. 50 ns for a 20 MHz 386 DX, 40 ns for a 25 MHz 386 DX, and 30 ns for a 33 MHz 386 DX).

For more detailed information on the encodings of instructions refer to section 6.2 Instruction Encodings. Section 6.2 explains the general structure of instruction encodings, and defines exactly the encodings of all fields contained within the instruction.

Instruction Clock Count Assumptions

1. The instruction has been prefetched, decoded, and is ready for execution.
2. Bus cycles do not require wait states.
3. There are no local bus HOLD requests delaying processor access to the bus.
4. No exceptions are detected during instruction execution.
5. If an effective address is calculated, it does not use two general register components. One register, scaling and displacement can be used within the clock counts shown. However, if the effective address calculation uses two general register components, add 1 clock to the clock count shown.

Instruction Clock Count Notation

1. If two clock counts are given, the smaller refers to a register operand and the larger refers to a memory operand.
2. n = number of times repeated.
3. m = number of components in the next instruction executed, where the entire displacement (if any) counts as one component, the entire immediate data (if any) counts as one component, and each of the **other** bytes of the instruction and prefix(es) each count as one component.

Wait States

Add 1 clock per wait state to instruction execution for each data access.

Table 6-1. 386™ DX Instruction Set Clock Count Summary

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
GENERAL DATA TRANSFER					
MOV = Move:					
Register to Register/Memory	1000100w | mod reg r/m	2/2	2/2	b	h
Register/Memory to Register	1000101w | mod reg r/m	2/4	2/4	b	h
Immediate to Register/Memory	1100011w | mod 000 r/m | immediate data	2/2	2/2	b	h
Immediate to Register (short form)	1011 w reg | immediate data	2	2		
Memory to Accumulator (short form)	1010000w | full displacement	4	4	b	h
Accumulator to Memory (short form)	1010001w | full displacement	2	2	b	h
Register Memory to Segment Register	10001110 | mod sreg3 r/m	2/5	18/19	b	h, i, j
Segment Register to Register/Memory	10001100 | mod sreg3 r/m	2/2	2/2	b	h
MOVSX = Move With Sign Extension					
Register From Register/Memory	00001111 | 1011111w | mod reg r/m	3/6	3/6	b	h
MOVZX = Move With Zero Extension					
Register From Register/Memory	00001111 | 1011011w | mod reg r/m	3/6	3/6	b	h
PUSH = Push:					
Register/Memory	11111111 | mod 110 r/m	5	5	b	h
Register (short form)	01010 reg	2	2	b	h
Segment Register (ES, CS, SS or DS)	000 sreg2 110	2	2	b	h
Segment Register (FS or GS)	00001111 | 10 sreg3 000	2	2	b	h
Immediate	011010s0 | immediate data	2	2	b	h
PUSHA = Push All	01100000	18	18	b	h
POP = Pop					
Register/Memory	10001111 | mod 000 r/m	5	5	b	h
Register (short form)	01011 reg	4	4	b	h
Segment Register (ES, SS or DS)	000 sreg 2 111	7	21	b	h, i, j
Segment Register (FS or GS)	00001111 | 10 sreg 3 001	7	21	b	h, i, j
POPA = Pop All	01100001	24	24	b	h
XCHG = Exchange					
Register/Memory With Register	1000011w | mod reg r/m	3/5	3/5	b, f	f, h
Register With Accumulator (short form)	10010 reg	3	3		
IN = Input from:		**Clk Count Virtual 8086 Mode**			
Fixed Port	1110010w | port number	†26	12	6*/26**	m
Variable Port	1110110w	†27	13	7*/27**	m
OUT = Output to:					
Fixed Port	1110011w | port number	†24	10	4*/24**	m
Variable Port	1110111w	†25	11	5*/25**	m
LEA = Load EA to Register	10001101 | mod reg r/m		2	2	

* If CPL ≤ IOPL ** If CPL > IOPL

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
SEGMENT CONTROL					
LDS = Load Pointer to DS	11000101 | mod reg r/m	7	22	b	h, i, j
LES = Load Pointer to ES	11000100 | mod reg r/m	7	22	b	h, i, j
LFS = Load Pointer to FS	00001111 | 10110100 | mod reg r/m	7	25	b	h, i, j
LGS = Load Pointer to GS	00001111 | 10110101 | mod reg r/m	7	25	b	h, i, j
LSS = Load Pointer to SS	00001111 | 10110010 | mod reg r/m	7	22	b	h, i, j
FLAG CONTROL					
CLC = Clear Carry Flag	11111000	2	2		
CLD = Clear Direction Flag	11111100	2	2		
CLI = Clear Interrupt Enable Flag	11111010	8	8		m
CLTS = Clear Task Switched Flag	00001111 | 00000110	6	6	c	l
CMC = Complement Carry Flag	11110101	2	2		
LAHF = Load AH into Flag	10011111	2	2		
POPF = Pop Flags	10011101	5	5	b	h, n
PUSHF = Push Flags	10011100	4	4	b	h
SAHF = Store AH into Flags	10011110	3	3		
STC = Set Carry Flag	11111001	2	2		
STD = Set Direction Flag	11111101	2	2		
STI = Set Interrupt Enable Flag	11111011	8	8		m
ARITHMETIC					
ADD = Add					
Register to Register	000000dw | mod reg r/m	2	2		
Register to Memory	0000000w | mod reg r/m	7	7	b	h
Memory to Register	0000001w | mod reg r/m	6	6	b	h
Immediate to Register/Memory	100000sw | mod 000 r/m | immediate data	2/7	2/7	b	h
Immediate to Accumulator (short form)	0000010w | immediate data	2	2		
ADC = Add With Carry					
Register to Register	000100dw | mod reg r/m	2	2		
Register to Memory	0001000w | mod reg r/m	7	7	b	h
Memory to Register	0001001w | mod reg r/m	6	6	b	h
Immediate to Register/Memory	100000sw | mod 010 r/m | immediate data	2/7	2/7	b	h
Immediate to Accumulator (short form)	0001010w | immediate data	2	2		
INC = Increment					
Register/Memory	1111111w | mod 000 r/m	2/6	2/6	b	h
Register (short form)	01000 reg	2	2		
SUB = Subtract					
Register from Register	001010dw | mod reg r/m	2	2		

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
ARITHMETIC (Continued)					
Register from Memory	0010100w | mod reg r/m	7	7	b	h
Memory from Register	0010101w | mod reg r/m	6	6	b	h
Immediate from Register/Memory	100000sw | mod101 r/m | immediate data	2/7	2/7	b	h
Immediate from Accumulator (short form)	0010110w | immediate data	2	2		
SBB = Subtract with Borrow					
Register from Register	000110dw | mod reg r/m	2	2		
Register from Memory	0001100w | mod reg r/m	7	7	b	h
Memory from Register	0001101w | mod reg r/m	6	6	b	h
Immediate from Register/Memory	100000sw | mod011 r/m | immediate data	2/7	2/7	b	h
Immediate from Accumulator (short form)	0001110w | immediate data	2	2		
DEC = Decrement					
Register/Memory	1111111w | reg001 r/m	2/6	2/6	b	h
Register (short form)	01001 reg	2	2		
CMP = Compare					
Register with Register	001110dw | mod reg r/m	2	2		
Memory with Register	0011100w | mod reg r/m	5	5	b	h
Register with Memory	0011101w | mod reg r/m	6	6	b	h
Immediate with Register/Memory	100000sw | mod111 r/m | immediate data	2/5	2/5	b	h
Immediate with Accumulator (short form)	0011110w | immediate data	2	2		
NEG = Change Sign	1111011w | mod011 r/m	2/6	2/6	b	h
AAA = ASCII Adjust for Add	00110111	4	4		
AAS = ASCII Adjust for Subtract	00111111	4	4		
DAA = Decimal Adjust for Add	00100111	4	4		
DAS = Decimal Adjust for Subtract	00101111	4	4		
MUL = Multiply (unsigned)					
Accumulator with Register/Memory	1111011w | mod100 r/m				
Multiplier-Byte		12–17/15–20	12–17/15–20	b, d	d, h
-Word		12–25/15–28	12–25/15–28	b, d	d, h
-Doubleword		12–41/15–44	12–41/15–44	b, d	d, h
IMUL = Integer Multiply (signed)					
Accumulator with Register/Memory	1111011w | mod101 r/m				
Multiplier-Byte		12–17/15–20	12–17/15–20	b, d	d, h
-Word		12–25/15–28	12–25/15–28	b, d	d, h
-Doubleword		12–41/15–44	12–41/15–44	b, d	d, h
Register with Register/Memory	00001111 | 10101111 | mod reg r/m				
Multiplier-Byte		12–17/15–20	12–17/15–20	b, d	d, h
-Word		12–25/15–28	12–25/15–28	b, d	d, h
-Doubleword		12–41/15–44	12–41/15–44	b, d	d, h
Register/Memory with Immediate to Register	011010s1 | mod reg r/m | immediate data				
-Word		13–26/14–27	13–26/14–27	b, d	d, h
-Doubleword		13–42/14–43	13–42/14–43	b, d	d, h

5

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
ARITHMETIC (Continued)					
DIV = Divide (Unsigned)					
Accumulator by Register/Memory	1111011w | mod110 r/m				
Divisor—Byte		14/17	14/17	b,e	e,h
—Word		22/25	22/25	b,e	e,h
—Doubleword		38/41	38/41	b,e	e,h
IDIV = Integer Divide (Signed)					
Accumulator By Register/Memory	1111011w | mod111 r/m				
Divisor—Byte		19/22	19/22	b,e	e,h
—Word		27/30	27/30	b,e	e,h
—Doubleword		43/46	43/46	b,e	e,h
AAD = ASCII Adjust for Divide	11010101 | 00001010	19	19		
AAM = ASCII Adjust for Multiply	11010100 | 00001010	17	17		
CBW = Convert Byte to Word	10011000	3	3		
CWD = Convert Word to Double Word	10011001	2	2		
LOGIC					
Shift Rotate Instructions					
Not Through Carry **(ROL, ROR, SAL, SAR, SHL,** and **SHR)**					
Register/Memory by 1	1101000w | mod TTT r/m	3/7	3/7	b	h
Register/Memory by CL	1101001w | mod TTT r/m	3/7	3/7	b	h
Register/Memory by Immediate Count	1100000w | mod TTT r/m | immed 8-bit data	3/7	3/7	b	h
Through Carry **(RCL** and **RCR)**					
Register/Memory by 1	1101000w | mod TTT r/m	9/10	9/10	b	h
Register/Memory by CL	1101001w | mod TTT r/m	9/10	9/10	b	h
Register/Memory by Immediate Count	1100000w | mod TTT r/m | immed 8-bit data	9/10	9/10	b	h
	TTT Instruction: 000 ROL; 001 ROR; 010 RCL; 011 RCR; 100 SHL/SAL; 101 SHR; 111 SAR				
SHLD = Shift Left Double					
Register/Memory by Immediate	00001111 | 10100100 | mod reg r/m | immed 8-bit data	3/7	3/7		
Register/Memory by CL	00001111 | 10100101 | mod reg r/m	3/7	3/7		
SHRD = Shift Right Double					
Register/Memory by Immediate	00001111 | 10101100 | mod reg r/m | immed 8-bit data	3/7	3/7		
Register/Memory by CL	00001111 | 10101101 | mod reg r/m	3/7	3/7		
AND = And					
Register to Register	001000dw | mod reg r/m	2	2		

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Clk Count Virtual 8086 Mode	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
LOGIC (Continued)						
Register to Memory	0010000w | mod reg r/m		7	7	b	h
Memory to Register	0010001w | mod reg r/m		6	6	b	h
Immediate to Register/Memory	100000sw | mod 100 r/m | immediate data		2/7	2/7	b	h
Immediate to Accumulator (Short Form)	0010010w | immediate data		2	2		
TEST = And Function to Flags, No Result						
Register/Memory and Register	1000010w | mod reg r/m		2/5	2/5	b	h
Immediate Data and Register/Memory	1111011w | mod 000 r/m | immediate data		2/5	2/5	b	h
Immediate Data and Accumulator (Short Form)	1010100w | immediate data		2	2		
OR = Or						
Register to Register	000010dw | mod reg r/m		2	2		
Register to Memory	0000100w | mod reg r/m		7	7	b	h
Memory to Register	0000101w | mod reg r/m		6	6	b	h
Immediate to Register/Memory	100000sw | mod 001 r/m | immediate data		2/7	2/7	b	h
Immediate to Accumulator (Short Form)	0000110w | immediate data		2	2		
XOR = Exclusive Or						
Register to Register	001100dw | mod reg r/m		2	2		
Register to Memory	0011000w | mod reg r/m		7	7	b	h
Memory to Register	0011001w | mod reg r/m		6	6	b	h
Immediate to Register/Memory	100000sw | mod 110 r/m | immediate data		2/7	2/7	b	h
Immediate to Accumulator (Short Form)	0011010w | immediate data		2	2		
NOT = Invert Register/Memory	1111011w | mod 010 r/m		2/6	2/6	b	h
STRING MANIPULATION						
CMPS = Compare Byte Word	1010011w		10	10	b	h
INS = Input Byte/Word from DX Port	0110110w	†29	15	9*/29**	b	h, m
LODS = Load Byte/Word to AL/AX/EAX	1010110w		5	5	b	h
MOVS = Move Byte Word	1010010w		8	8	b	h
OUTS = Output Byte/Word to DX Port	0110111w	†28	14	8*/28**	b	h, m
SCAS = Scan Byte Word	1010111w		8	8	b	h
STOS = Store Byte/Word from AL/AX/EX	1010101w		5	5	b	h
XLAT = Translate String	11010111		5	5		h
REPEATED STRING MANIPULATION Repeated by Count in CX or ECX						
REPE CMPS = Compare String (Find Non-Match)	11110011 | 1010011w		5+9n	5+9n	b	h

* If CPL ≤ IOPL ** If CPL > IOPL

5

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
REPEATED STRING MANIPULATION (Continued)					
REPNE CMPS = Compare String					
(Find Match)	11110010 1010011w — Clk Count Virtual 8086 Mode	5+9n	5+9n	b	h
REP INS = Input String	11110010 0110110w — †28+6n	14+6n	8+6n*/28+6n**	b	h, m
REP LODS = Load String	11110010 1010110w	5+6n	5+6n	b	h
REP MOVS = Move String	11110010 1010010w	8+4n	8+4n	b	h
REP OUTS = Output String	11110010 0110111w — †26+5n	12+5n	6+5n*/26+5n**	b	h, m
REPE SCAS = Scan String					
(Find Non-AL/AX/EAX)	11110011 1010111w	5+8n	5+8n	b	h
REPNE SCAS = Scan String					
(Find AL/AX/EAX)	11110010 1010111w	5+8n	5+8n	b	h
REP STOS = Store String	11110010 1010101w	5+5n	5+5n	b	h
BIT MANIPULATION					
BSF = Scan Bit Forward	00001111 10111100 mod reg r/m	11+3n	11+3n	b	h
BSR = Scan Bit Reverse	00001111 10111101 mod reg r/m	9+3n	9+3n	b	h
BT = Test Bit					
Register/Memory, Immediate	00001111 10111010 mod 100 r/m immed 8-bit data	3/6	3/6	b	h
Register/Memory, Register	00001111 10100011 mod reg r/m	3/12	3/12	b	h
BTC = Test Bit and Complement					
Register/Memory, Immediate	00001111 10111010 mod 111 r/m immed 8-bit data	6/8	6/8	b	h
Register/Memory, Register	00001111 10111011 mod reg r/m	6/13	6/13	b	h
BTR = Test Bit and Reset					
Register/Memory, Immediate	00001111 10111010 mod 110 r/m immed 8-bit data	6/8	6/8	b	h
Register/Memory, Register	00001111 10110011 mod reg r/m	6/13	6/13	b	h
BTS = Test Bit and Set					
Register/Memory, Immediate	00001111 10111010 mod 101 r/m immed 8-bit data	6/8	6/8	b	h
Register/Memory, Register	00001111 10101011 mod reg r/m	6/13	6/13	b	h
CONTROL TRANSFER					
CALL = Call					
Direct Within Segment	11101000 full displacement	7+m	7+m	b	r
Register/Memory					
Indirect Within Segment	11111111 mod 010 r/m	7+m/ 10+m	7+m/ 10+m	b	h, r
Direct Intersegment	10011010 unsigned full offset, selector	17+m	34+m	b	j,k,r

NOTES:
† Clock count shown applies if I/O permission allows I/O to the port in virtual 8086 mode. If I/O bit map denies permission exception 13 fault occurs; refer to clock counts for INT 3 instruction.
* If CPL ≤ IOPL ** If CPL > IOPL

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
CONTROL TRANSFER (Continued)					
Protected Mode Only (Direct Intersegment)					
Via Call Gate to Same Privilege Level			52+m		h,j,k,r
Via Call Gate to Different Privilege Level, (No Parameters)			86+m		h,j,k,r
Via Call Gate to Different Privilege Level, (x Parameters)			94+4x+m		h,j,k,r
From 80286 Task to 80286 TSS			273		h,j,k,r
From 80286 Task to 386™ DX TSS			298		h,j,k,r
From 80286 Task to Virtual 8086 Task (386™ DX TSS)			218		h,j,k,r
From 386™ DX Task to 80286 TSS			273		h,j,k,r
From 386™ DX Task to 386™ DX TSS			300		h,j,k,r
From 386™ DX Task to Virtual 8086 Task (386™ DX TSS)			218		h,j,k,r
Indirect Intersegment	11111111 mod011 r/m	22+m	38+m	b	h,j,k,r
Protected Mode Only (Indirect Intersegment)					
Via Call Gate to Same Privilege Level			56+m		h,j,k,r
Via Call Gate to Different Privilege Level, (No Parameters)			90+m		h,j,k,r
Via Call Gate to Different Privilege Level, (x Parameters)			98+4x+m		h,j,k,r
From 80286 Task to 80286 TSS			278		h,j,k,r
From 80286 Task to 386™ DX TSS			303		h,j,k,r
From 80286 Task to Virtual 8086 Task (386™ DX TSS)			222		h,j,k,r
From 386™ DX Task to 80286 TSS			278		h,j,k,r
From 386™ DX Task to 386™ DX TSS			305		h,j,k,r
From 386™ DX Task to Virtual 8086 Task (386™ DX TSS)			222		h,j,k,r
JMP = Unconditional Jump					
Short	11101011 8-bit displacement	7+m	7+m		r
Direct within Segment	11101001 full displacement	7+m	7+m		r
Register/Memory Indirect within Segment	11111111 mod100 r/m	7+m/ 10+m	7+m/ 10+m	b	h,r
Direct Intersegment	11101010 unsigned full offset, selector	12+m	27+m		j,k,r
Protected Mode Only (Direct Intersegment)					
Via Call Gate to Same Privilege Level			45+m		h,j,k,r
From 80286 Task to 80286 TSS			274		h,j,k,r
From 80286 Task to 386™ DX TSS			301		h,j,k,r
From 80286 Task to Virtual 8086 Task (386™ DX TSS)			219		h,j,k,r
From 386™ DX Task to 80286 TSS			270		h,j,k,r
From 386™ DX Task to 386™ DX TSS			303		h,j,k,r
From 386™ DX Task to Virtual 8086 Task (386™ DX TSS)			221		h,j,k,r
Indirect Intersegment	11111111 mod101 r/m	17+m	31+m	b	h,j,k,r
Protected Mode Only (Indirect Intersegment)					
Via Call Gate to Same Privilege Level			49+m		h,j,k,r
From 80286 Task to 80286 TSS			279		h,j,k,r
From 80286 Task to 386™ DX TSS			306		h,j,k,r
From 80286 Task to Virtual 8086 Task (386™ DX TSS)			223		h,j,k,r
From 386™ DX Task to 80286 TSS			275		h,j,k,r
From 386™ DX Task to 386™ DX TSS			308		h,j,k,r
From 386™ DX Task to Virtual 8086 Task (386™ DX TSS)			225		h,j,k,r

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
CONTROL TRANSFER (Continued)					
RET = Return from CALL:					
Within Segment	11000011	10 + m	10 + m	b	g, h, r
Within Segment Adding Immediate to SP	11000010 | 16-bit displ	10 + m	10 + m	b	g, h, r
Intersegment	11001011	18 + m	32+m	b	g, h, j, k, r
Intersegment Adding Immediate to SP	11001010 | 16-bit displ	18 + m	32+m	b	g, h, j, k, r
Protected Mode Only (RET):					
to Different Privilege Level					
Intersegment			69		h, j, k, r
Intersegment Adding Immediate to SP			69		h, j, k, r
CONDITIONAL JUMPS					
NOTE: Times Are Jump "Taken or Not Taken"					
JO = Jump on Overflow					
8-Bit Displacement	01110000 | 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10000000 | full displacement	7 + m or 3	7 + m or 3		r
JNO = Jump on Not Overflow					
8-Bit Displacement	01110001 | 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10000001 | full displacement	7 + m or 3	7 + m or 3		r
JB/JNAE = Jump on Below/Not Above or Equal					
8-Bit Displacement	01110010 | 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10000010 | full displacement	7 + m or 3	7 + m or 3		r
JNB/JAE = Jump on Not Below/Above or Equal					
8-Bit Displacement	01110011 | 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10000011 | full displacement	7 + m or 3	7 + m or 3		r
JE/JZ = Jump on Equal/Zero					
8-Bit Displacement	01110100 | 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10000100 | full displacement	7 + m or 3	7 + m or 3		r
JNE/JNZ = Jump on Not Equal/Not Zero					
8-Bit Displacement	01110101 | 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10000101 | full displacement	7 + m or 3	7 + m or 3		r
JBE/JNA = Jump on Below or Equal/Not Above					
8-Bit Displacement	01110110 | 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10000110 | full displacement	7 + m or 3	7 + m or 3		r
JNBE/JA = Jump on Not Below or Equal/Above					
8-Bit Displacement	01110111 | 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10000111 | full displacement	7 + m or 3	7 + m or 3		r
JS = Jump on Sign					
8-Bit Displacement	01111000 | 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10001000 | full displacement	7 + m or 3	7 + m or 3		r

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
CONDITIONAL JUMPS (Continued)					
JNS = Jump on Not Sign					
8-Bit Displacement	01111001 | 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10001001 | full displacement	7 + m or 3	7 + m or 3		r
JP/JPE = Jump on Parity/Parity Even					
8-Bit Displacement	01111010 | 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10001010 | full displacement	7 + m or 3	7 + m or 3		r
JNP/JPO = Jump on Not Parity/Parity Odd					
8-Bit Displacement	01111011 | 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10001011 | full displacement	7 + m or 3	7 + m or 3		r
JL/JNGE = Jump on Less/Not Greater or Equal					
8-Bit Displacement	01111100 | 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10001100 | full displacement	7 + m or 3	7 + m or 3		r
JNL/JGE = Jump on Not Less/Greater or Equal					
8-Bit Displacement	01111101 | 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10001101 | full displacement	7 + m or 3	7 + m or 3		r
JLE/JNG = Jump on Less or Equal/Not Greater					
8-Bit Displacement	01111110 | 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10001110 | full displacement	7 + m or 3	7 + m or 3		r
JNLE/JG = Jump on Not Less or Equal/Greater					
8-Bit Displacement	01111111 | 8-bit displ	7 + m or 3	7 + m or 3		r
Full Displacement	00001111 | 10001111 | full displacement	7 + m or 3	7 + m or 3		r
JCXZ = Jump on CX Zero	11100011 | 8-bit displ	9 + m or 5	9 + m or 5		r
JECXZ = Jump on ECX Zero	11100011 | 8-bit displ	9 + m or 5	9 + m or 5		r
(Address Size Prefix Differentiates JCXZ from JECXZ)					
LOOP = Loop CX Times	11100010 | 8-bit displ	11 + m	11 + m		r
LOOPZ/LOOPE = Loop with Zero/Equal	11100001 | 8-bit displ	11 + m	11 + m		r
LOOPNZ/LOOPNE = Loop While Not Zero	11100000 | 8-bit displ	11 + m	11 + m		r
CONDITIONAL BYTE SET					
NOTE: Times Are Register/Memory					
SETO = Set Byte on Overflow					
To Register/Memory	00001111 | 10010000 | mod 000 r/m	4/5	4/5		h
SETNO = Set Byte on Not Overflow					
To Register/Memory	00001111 | 10010001 | mod 000 r/m	4/5	4/5		h
SETB/SETNAE = Set Byte on Below/Not Above or Equal					
To Register/Memory	00001111 | 10010010 | mod 000 r/m	4/5	4/5		h

5

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
CONDITIONAL BYTE SET (Continued)					
SETNB = Set Byte on Not Below/Above or Equal					
To Register/Memory	00001111 10010011 mod000 r/m	4/5	4/5		h
SETE/SETZ = Set Byte on Equal/Zero					
To Register/Memory	00001111 10010100 mod000 r/m	4/5	4/5		h
SETNE/SETNZ = Set Byte on Not Equal/Not Zero					
To Register/Memory	00001111 10010101 mod000 r/m	4/5	4/5		h
SETBE/SETNA = Set Byte on Below or Equal/Not Above					
To Register/Memory	00001111 10010110 mod000 r/m	4/5	4/5		h
SETNBE/SETA = Set Byte on Not Below or Equal/Above					
To Register/Memory	00001111 10010111 mod000 r/m	4/5	4/5		h
SETS = Set Byte on Sign					
To Register/Memory	00001111 10011000 mod000 r/m	4/5	4/5		h
SETNS = Set Byte on Not Sign					
To Register/Memory	00001111 10011001 mod000 r/m	4/5	4/5		h
SETP/SETPE = Set Byte on Parity/Parity Even					
To Register/Memory	00001111 10011010 mod000 r/m	4/5	4/5		h
SETNP/SETPO = Set Byte on Not Parity/Parity Odd					
To Register/Memory	00001111 10011011 mod000 r/m	4/5	4/5		h
SETL/SETNGE = Set Byte on Less/Not Greater or Equal					
To Register/Memory	00001111 10011100 mod000 r/m	4/5	4/5		h
SETNL/SETGE = Set Byte on Not Less/Greater or Equal					
To Register/Memory	00001111 01111101 mod000 r/m	4/5	4/5		h
SETLE/SETNG = Set Byte on Less or Equal/Not Greater					
To Register/Memory	00001111 10011110 mod000 r/m	4/5	4/5		h
SETNLE/SETG = Set Byte on Not Less or Equal/Greater					
To Register/Memory	00001111 10011111 mod000 r/m	4/5	4/5		h
ENTER = Enter Procedure	11001000 16-bit displacement, 8-bit level				
L = 0		10	10	b	h
L = 1		12	12	b	h
L > 1		15 + 4(n − 1)	15 + 4(n − 1)	b	h
LEAVE = Leave Procedure	11001001	4	4	b	h

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
INTERRUPT INSTRUCTIONS					
INT = Interrupt:					
Type Specified	11001101 | type	37		b	
Type 3	11001100	33		b	
INTO = Interrupt 4 if Overflow Flag Set	11001110				
If OF = 1		35		b, e	
If OF = 0		3	3	b, e	
Bound = Interrupt 5 if Detect Value Out of Range	01100010 | mod reg r/m				
If Out of Range		44		b, e	e, g, h, j, k, r
If In Range		10	10	b, e	e, g, h, j, k, r
Protected Mode Only (INT)					
INT: Type Specified					
Via Interrupt or Trap Gate to Same Privilege Level			59		g, j, k, r
Via Interrupt or Trap Gate to Different Privilege Level			99		g, j, k, r
From 80286 Task to 80286 TSS via Task Gate			282		g, j, k, r
From 80286 Task to 386™ DX TSS via Task Gate			309		g, j, k, r
From 80286 Task to virt 8086 md via Task Gate			226		g, j, k, r
From 386™ DX Task to 80286 TSS via Task Gate			284		g, j, k, r
From 386™ DX Task to 386™ DX TSS via Task Gate			311		g, j, k, r
From 386™ DX Task to virt 8086 md via Task Gate			228		g, j, k, r
From virt 8086 md to 80286 TSS via Task Gate			289		g, j, k, r
From virt 8086 md to 386™ DX TSS via Task Gate			316		g, j, k, r
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate			119		
INT: TYPE 3					
Via Interrupt or Trap Gate to Same Privilege Level			59		g, j, k, r
Via Interrupt or Trap Gate to Different Privilege Level			99		g, j, k, r
From 80286 Task to 80286 TSS via Task Gate			278		g, j, k, r
From 80286 Task to 386™ DX TSS via Task Gate			305		g, j, k, r
From 80286 Task to Virt 8086 md via Task Gate			222		g, j, k, r
From 386™ DX Task to 80286 TSS via Task Gate			280		g, j, k, r
From 386™ DX Task to 386™ DX TSS via Task Gate			307		g, j, k, r
From 386™ DX Task to Virt 8086 md via Task Gate			224		g, j, k, r
From virt 8086 md to 80286 TSS via Task Gate			285		g, j, k, r
From virt 8086 md to 386™ DX TSS via Task Gate			312		g, j, k, r
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate			119		
INTO:					
Via Interrupt or Trap Grate to Same Privilege Level			59		g, j, k, r
Via Interrupt or Trap Gate to Different Privilege Level			99		g, j, k, r
From 80286 Task to 80286 TSS via Task Gate			280		g, j, k, r
From 80286 Task to 386™ DX TSS via Task Gate			307		g, j, k, r
From 80286 Task to virt 8086 md via Task Gate			224		g, j, k, r
From 386™ DX Task to 80286 TSS via Task Gate			282		g, j, k, r
From 386™ DX Task to 386™ DX TSS via Task Gate			309		g, j, k, r
From 386™ DX Gate			225		g, j, k, r
From virt 8086 md to 80286 TSS via Task Gate			287		g, j, k, r
From virt 8086 md to 386™ DX TSS via Task Gate			314		g, j, k, r
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate			119		

5

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
INTERRUPT INSTRUCTIONS (Continued)					
BOUND:					
Via Interrupt or Trap Gate to Same Privilege Level			59		g, j, k, r
Via Interrupt or Trap Gate to Different Privilege Level			99		g, j, k, r
From 80286 Task to 80286 TSS via Task Gate			254		g, j, k, r
From 80286 Task to 386™ DX TSS via Task Gate			284		g, j, k, r
From 80268 Task to virt 8086 Mode via Task Gate			231		g, j, k, r
From 386™ DX Task to 80286 TSS via Task Gate			264		g, j, k, r
From 386™ DX Task to 386™ DX TSS via Task Gate			294		g, j, k, r
From 80368 Task to virt 8086 Mode via Task Gate			243		g, j, k, r,
From virt 8086 Mode to 80286 TSS via Task Gate			264		g, j, k, r
From virt 8086 Mode to 386™ DX TSS via Task Gate			294		g, j, k, r
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate			119		
INTERRUPT RETURN					
IRET = Interrupt Return	11001111	22			g, h, j, k, r
Protected Mode Only (IRET)					
To the Same Privilege Level (within task)			38		g, h, j, k, r
To Different Privilege Level (within task)			82		g, h, j, k, r
From 80286 Task to 80286 TSS			232		h, j, k, r
From 80286 Task to 386™ DX TSS			265		h, j, k, r
From 80286 Task to Virtual 8086 Task			213		h, j, k, r
From 80286 Task to Virtual 8086 Mode (within task)			60		
From 386™ DX Task to 80286 TSS			271		h, j, k, r
From 386™ DX Task to 386™ DX TSS			275		h, j, k, r
From 386™ DX Task to Virtual 8086 Task			223		h, j, k, r
From 386™ DX Task to Virtual 8086 Mode (within task)			60		
PROCESSOR CONTROL					
HLT = HALT	11110100	5	5		l
MOV = Move to and From Control/Debug/Test Registers					
CR0/CR2/CR3 from register	00001111 00100010 11eee reg	11/4/5	11/4/5		l
Register From CR0–3	00001111 00100000 11eee reg	6	6		l
DR0–3 From Register	00001111 00100011 11eee reg	22	22		l
DR6–7 From Register	00001111 00100011 11eee reg	16	16		l
Register from DR6–7	00001111 00100001 11eee reg	14	14		l
Register from DR0–3	00001111 00100001 11eee reg	22	22		l
TR6–7 from Register	00001111 00100110 11eee reg	12	12		l
Register from TR6–7	00001111 00100100 11eee reg	12	12		l
NOP = No Operation	10010000	3	3		
WAIT = Wait until BUSY# pin is negated	10011011	7	7		

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
PROCESSOR EXTENSION INSTRUCTIONS					
Processor Extension Escape	11011TTT | mod LLL r/m TTT and LLL bits are opcode information for coprocessor.	See 80287/80387 data sheets for clock counts			h
PREFIX BYTES					
Address Size Prefix	01100111	0	0		
LOCK = Bus Lock Prefix	11110000	0	0		m
Operand Size Prefix	01100110	0	0		
Segment Override Prefix					
CS:	00101110	0	0		
DS:	00111110	0	0		
ES:	00100110	0	0		
FS:	01100100	0	0		
GS:	01100101	0	0		
SS:	00110110	0	0		
PROTECTION CONTROL					
ARPL = Adjust Requested Privilege Level					
From Register/Memory	01100011 | mod reg r/m	N/A	20/21	a	h
LAR = Load Access Rights					
From Register/Memory	00001111 | 00000010 | mod reg r/m	N/A	15/16	a	g, h, j, p
LGDT = Load Global Descriptor					
Table Register	00001111 | 00000001 | mod 010 r/m	11	11	b, c	h, l
LIDT = Load Interrupt Descriptor					
Table Register	00001111 | 00000001 | mod 011 r/m	11	11	b, c	h, l
LLDT = Load Local Descriptor					
Table Register to Register/Memory	00001111 | 00000000 | mod 010 r/m	N/A	20/24	a	g, h, j, l
LMSW = Load Machine Status Word					
From Register/Memory	00001111 | 00000001 | mod 110 r/m	11/14	11/14	b, c	h, l
LSL = Load Segment Limit					
From Register/Memory	00001111 | 00000011 | mod reg r/m				
Byte-Granular Limit		N/A	21/22	a	g, h, j, p
Page-Granular Limit		N/A	25/26	a	g, h, j, p
LTR = Load Task Register					
From Register/Memory	00001111 | 00000000 | mod 011 r/m	N/A	23/27	a	g, h, j, l
SGDT = Store Global Descriptor					
Table Register	00001111 | 00000001 | mod 000 r/m	9	9	b, c	h
SIDT = Store Interrupt Descriptor					
Table Register	00001111 | 00000001 | mod 001 r/m	9	9	b, c	h
SLDT = Store Local Descriptor Table Register					
To Register/Memory	00001111 | 00000000 | mod 000 r/m	N/A	2/2	a	h

5

Table 6-1. 386™ DX Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT		NOTES	
		Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
SMSW = **Store Machine Status Word**	00001111 | 00000001 | mod 1 0 0 r/m	2/2	2/2	b, c	h, l
STR = **Store Task Register** To Register/Memory	00001111 | 00000000 | mod 0 0 1 r/m	N/A	2/2	a	h
VERR = **Verify Read Accesss** Register/Memory	00001111 | 00000000 | mod 1 0 0 r/m	N/A	10/11	a	g, h, j, p
VERW = **Verify Write Accesss**	00001111 | 00000000 | mod 1 0 1 r/m	N/A	15/16	a	g, h, j, p

INSTRUCTION NOTES FOR TABLE 6-1

Notes a through c apply to 386 DX Real Address Mode only:

a. This is a Protected Mode instruction. Attempted execution in Real Mode will result in exception 6 (invalid opcode).

b. Exception 13 fault (general protection) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum CS, DS, ES, FS or GS limit, FFFFH. Exception 12 fault (stack segment limit violation or not present) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.

c. This instruction may be executed in Real Mode. In Real Mode, its purpose is primarily to initialize the CPU for Protected Mode.

Notes d through g apply to 386 DX Real Address Mode and 386 DX Protected Virtual Address Mode:

d. The 386 DX uses an early-out multiply algorithm. The actual number of clocks depends on the position of the most significant bit in the operand (multiplier).

Clock counts given are minimum to maximum. To calculate actual clocks use the following formula:

Actual Clock = if m < > 0 then max ([$\log_2$ |m|], 3) + b clocks:
if m = 0 then 3+b clocks

In this formula, m is the multiplier, and

b = 9 for register to register,
b = 12 for memory to register,
b = 10 for register with immediate to register,
b = 11 for memory with immediate to register.

e. An exception may occur, depending on the value of the operand.

f. LOCK# is automatically asserted, regardless of the presence or absence of the LOCK# prefix.

g. LOCK# is asserted during descriptor table accesses.

Notes h through r apply to 386 DX Protected Virtual Address Mode only:

h. Exception 13 fault (general protection violation) will occur if the memory operand in CS, DS, ES, FS or GS cannot be used due to either a segment limit violation or access rights violation. If a stack limit is violated, an exception 12 (stack segment limit violation or not present) occurs.

i. For segment load operations, the CPL, RPL, and DPL must agree with the privilege rules to avoid an exception 13 fault (general protection violation). The segment's descriptor must indicate "present" or exception 11 (CS, DS, ES, FS, GS not present). If the SS register is loaded and a stack segment not present is detected, an exception 12 (stack segment limit violation or not present) occurs.

j. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert LOCK# to maintain descriptor integrity in multiprocessor systems.

k. JMP, CALL, INT, RET and IRET instructions referring to another code segment will cause an exception 13 (general protection violation) if an applicable privilege rule is violated.

l. An exception 13 fault occurs if CPL is greater than 0 (0 is the most privileged level).

m. An exception 13 fault occurs if CPL is greater than IOPL.

n. The IF bit of the flag register is not updated if CPL is greater than IOPL. The IOPL and VM fields of the flag register are updated only if CPL = 0.

o. The PE bit of the MSW (CR0) cannot be reset by this instruction. Use MOV into CR0 if desiring to reset the PE bit.

p. Any violation of privilege rules as applied to the selector operand does not cause a protection exception; rather, the zero flag is cleared.

q. If the coprocessor's memory operand violates a segment limit or segment access rights, an exception 13 fault (general protection exception) will occur before the ESC instruction is executed. An exception 12 fault (stack segment limit violation or not present) will occur if the stack limit is violated by the operand's starting address.

r. The destination of a JMP, CALL, INT, RET or IRET must be in the defined limit of a code segment or an exception 13 fault (general protection violation) will occur.

6.2 INSTRUCTION ENCODING

6.2.1 Overview

All instruction encodings are subsets of the general instruction format shown in Figure 6-1. Instructions consist of one or two primary opcode bytes, possibly an address specifier consisting of the "mod r/m" byte and "scaled index" byte, a displacement if required, and an immediate data field if required.

Within the primary opcode or opcodes, smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary opcode byte(s). This byte, the mod r/m byte, specifies the address mode to be used. Certain encodings of the mod r/m byte indicate a second addressing byte, the scale-index-base byte, follows the mod r/m byte to fully specify the addressing mode.

Addressing modes can include a displacement immediately following the mod r/m byte, or scaled index byte. If a displacement is present, the possible sizes are 8, 16 or 32 bits.

If the instruction specifies an immediate operand, the immediate operand follows any displacement bytes. The immediate operand, if specified, is always the last field of the instruction.

Figure 6-1 illustrates several of the fields that can appear in an instruction, such as the mod field and the r/m field, but the Figure does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the opcode bytes themselves. Table 6-2 is a complete list of all fields appearing in the 386 DX instruction set. Further ahead, following Table 6-2, are detailed tables for each field.

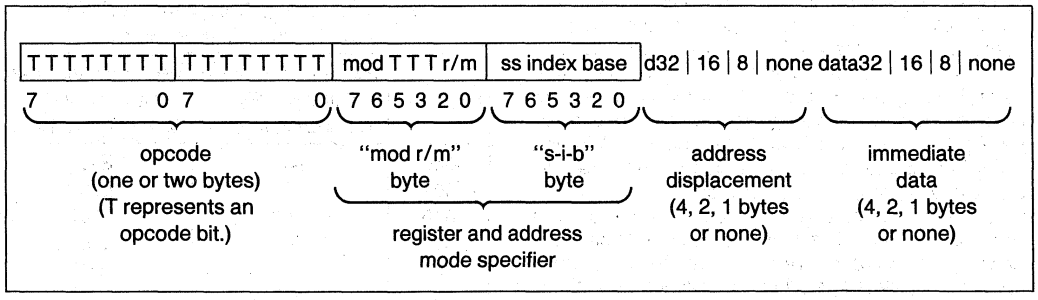


Figure 6-1. General Instruction Format

Table 6-2. Fields within 386™ DX Instructions

Field Name	Description	Number of Bits
w	Specifies if Data is Byte or Full Size (Full Size is either 16 or 32 Bits	1
d	Specifies Direction of Data Operation	1
s	Specifies if an Immediate Data Field Must be Sign-Extended	1
reg	General Register Specifier	3
mod r/m	Address Mode Specifier (Effective Address can be a General Register)	2 for mod; 3 for r/m
ss	Scale Factor for Scaled Index Address Mode	2
index	General Register to be used as Index Register	3
base	General Register to be used as Base Register	3
sreg2	Segment Register Specifier for CS, SS, DS, ES	2
sreg3	Segment Register Specifier for CS, SS, DS, ES, FS, GS	3
tttn	For Conditional Instructions, Specifies a Condition Asserted or a Condition Negated	4

Note: Table 6-1 shows encoding of individual instructions.

6.2.2 32-Bit Extensions of the Instruction Set

With the 386 DX, the 8086/80186/80286 instruction set is extended in two orthogonal directions: 32-bit forms of all 16-bit instructions are added to support the 32-bit data types, and 32-bit addressing modes are made available for all instructions referencing memory. This orthogonal instruction set extension is accomplished having a Default (D) bit in the code segment descriptor, and by having 2 prefixes to the instruction set.

Whether the instruction defaults to operations of 16 bits or 32 bits depends on the setting of the D bit in the code segment descriptor, which gives the default length (either 32 bits or 16 bits) for both operands and effective addresses when executing that code segment. In the Real Address Mode or Virtual 8086 Mode, no code segment descriptors are used, but a D value of 0 is assumed internally by the 386 DX when operating in those modes (for 16-bit default sizes compatible with the 8086/80186/80286).

Two prefixes, the Operand Size Prefix and the Effective Address Size Prefix, allow overriding individually the Default selection of operand size and effective address size. These prefixes may precede any opcode bytes and affect only the instruction they precede. If necessary, one or both of the prefixes may be placed before the opcode bytes. The presence of the Operand Size Prefix and the Effective Address Prefix will toggle the operand size or the effective address size, respectively, to the value "opposite" from the Default setting. For example, if the default operand size is for 32-bit data operations, then presence of the Operand Size Prefix toggles the instruction to 16-bit data operation. As another example, if the default effective address size is 16 bits, presence of the Effective Address Size prefix toggles the instruction to use 32-bit effective address computations.

These 32-bit extensions are available in all 386 DX modes, including the Real Address Mode or the Virtual 8086 Mode. In these modes the default is always 16 bits, so prefixes are needed to specify 32-bit operands or addresses. For instructions with more than one prefix, the order of prefixes is unimportant.

Unless specified otherwise, instructions with 8-bit and 16-bit operands do not affect the contents of the high-order bits of the extended registers.

6.2.3 Encoding of Instruction Fields

Within the instruction are several fields indicating register selection, addressing mode and so on. The exact encodings of these fields are defined immediately ahead.

6.2.3.1 ENCODING OF OPERAND LENGTH (w) FIELD

For any given instruction performing a data operation, the instruction is executing as a 32-bit operation or a 16-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or the full operation size, as shown in the table below.

w Field	Operand Size During 16-Bit Data Operations	Operand Size During 32-Bit Data Operations
0	8 Bits	8 Bits
1	16 Bits	32 Bits

6.2.3.2 ENCODING OF THE GENERAL REGISTER (reg) FIELD

The general register is specified by the reg field, which may appear in the primary opcode bytes, or as the reg field of the "mod r/m" byte, or as the r/m field of the "mod r/m" byte.

Encoding of reg Field When w Field is not Present in Instruction

reg Field	Register Selected During 16-Bit Data Operations	Register Selected During 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
110	SI	ESI
111	DI	EDI

Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field During 16-Bit Data Operations:		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Register Specified by reg Field During 32-Bit Data Operations		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

6.2.3.3 ENCODING OF THE SEGMENT REGISTER (sreg) FIELD

The sreg field in certain instructions is a 2-bit field allowing one of the four 80286 segment registers to be specified. The sreg field in other instructions is a 3-bit field, allowing the 386 DX FS and GS segment registers to be specified.

2-Bit sreg2 Field

2-Bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

3-Bit sreg3 Field

3-Bit sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	do not use
111	do not use

6.2.3.4 ENCODING OF ADDRESS MODE

Except for special instructions, such as PUSH or POP, where the addressing mode is pre-determined, the addressing mode for the current instruction is specified by addressing bytes following the primary opcode. The primary addressing byte is the "mod r/m" byte, and a second byte of addressing information, the "s-i-b" (scale-index-base) byte, can be specified.

The s-i-b byte (scale-index-base byte) is specified when using 32-bit addressing mode and the "mod r/m" byte has r/m = 100 and mod = 00, 01 or 10. When the sib byte is present, the 32-bit addressing mode is a function of the mod, ss, index, and base fields.

The primary addressing byte, the "mod r/m" byte, also contains three bits (shown as TTT in Figure 6-1) sometimes used as an extension of the primary opcode. The three bits, however, may also be used as a register field (reg).

When calculating an effective address, either 16-bit addressing or 32-bit addressing is used. 16-bit addressing uses 16-bit address components to calculate the effective address while 32-bit addressing uses 32-bit address components to calculate the effective address. When 16-bit addressing is used, the "mod r/m" byte is interpreted as a 16-bit addressing mode specifier. When 32-bit addressing is used, the "mod r/m" byte is interpreted as a 32-bit addressing mode specifier.

Tables on the following three pages define all encodings of all 16-bit addressing modes and 32-bit addressing modes.

Encoding of 16-bit Address Mode with "mod r/m" Byte

mod r/m	Effective Address
00 000	DS:[BX+SI]
00 001	DS:[BX+DI]
00 010	SS:[BP+SI]
00 011	SS:[BP+DI]
00 100	DS:[SI]
00 101	DS:[DI]
00 110	DS:d16
00 111	DS:[BX]
01 000	DS:[BX+SI+d8]
01 001	DS:[BX+DI+d8]
01 010	SS:[BP+SI+d8]
01 011	SS:[BP+DI+d8]
01 100	DS:[SI+d8]
01 101	DS:[DI+d8]
01 110	SS:[BP+d8]
01 111	DS:[BX+d8]

mod r/m	Effective Address
10 000	DS:[BX+SI+d16]
10 001	DS:[BX+DI+d16]
10 010	SS:[BP+SI+d16]
10 011	SS:[BP+DI+d16]
10 100	DS:[SI+d16]
10 101	DS:[DI+d16]
10 110	SS:[BP+d16]
10 111	DS:[BX+d16]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by r/m During 16-Bit Data Operations		
mod r/m	Function of w Field	
	(when w=0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by r/m During 32-Bit Data Operations		
mod r/m	Function of w Field	
	(when w=0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode with "mod r/m" byte (no "s-i-b" byte present):

mod r/m	Effective Address
00 000	DS:[EAX]
00 001	DS:[ECX]
00 010	DS:[EDX]
00 011	DS:[EBX]
00 100	s-i-b is present
00 101	DS:d32
00 110	DS:[ESI]
00 111	DS:[EDI]
01 000	DS:[EAX + d8]
01 001	DS:[ECX + d8]
01 010	DS:[EDX + d8]
01 011	DS:[EBX + d8]
01 100	s-i-b is present
01 101	SS:[EBP + d8]
01 110	DS:[ESI + d8]
01 111	DS:[EDI + d8]

mod r/m	Effective Address
10 000	DS:[EAX + d32]
10 001	DS:[ECX + d32]
10 010	DS:[EDX + d32]
10 011	DS:[EBX + d32]
10 100	s-i-b is present
10 101	SS:[EBP + d32]
10 110	DS:[ESI + d32]
10 111	DS:[EDI + d32]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by reg or r/m during 16-Bit Data Operations:		
mod r/m	function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by reg or r/m during 32-Bit Data Operations:		
mod r/m	function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode ("mod r/m" byte and "s-i-b" byte present):

mod base	Effective Address
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

NOTE:
Mod field in "mod r/m" byte; ss, index, base fields in "s-i-b" byte.

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8

index	Index Register
000	EAX
001	ECX
010	EDX
011	EBX
100	no index reg**
101	EBP
110	ESI
111	EDI

****IMPORTANT NOTE:**
When index field is 100, indicating "no index register," then ss field MUST equal 00. If index is 100 and ss does not equal 00, the effective address is undefined.

6.2.3.5 ENCODING OF OPERATION DIRECTION (d) FIELD

In many two-operand instructions the d field is present to indicate which operand is considered the source and which is the destination.

d	Direction of Operation
0	Register/Memory <- - Register "reg" Field Indicates Source Operand; "mod r/m" or "mod ss index base" Indicates Destination Operand
1	Register <- - Register/Memory "reg" Field Indicates Destination Operand; "mod r/m" or "mod ss index base" Indicates Source Operand

6.2.3.6 ENCODING OF SIGN-EXTEND (s) FIELD

The s field occurs primarily to instructions with immediate data fields. The s field has an effect only if the size of the immediate data is 8 bits and is being placed in a 16-bit or 32-bit destination.

s	Effect on Immediate Data8	Effect on Immediate Data 16|32
0	None	None
1	Sign-Extend Data8 to Fill 16-Bit or 32-Bit Destination	None

6.2.3.7 ENCODING OF CONDITIONAL TEST (tttn) FIELD

For the conditional instructions (conditional jumps and set on condition), tttn is encoded with n indicating to use the condition (n = 0) or its negation (n = 1), and ttt giving the condition to test.

Mnemonic	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less or Equal/Greater Than	1111

6.2.3.8 ENCODING OF CONTROL OR DEBUG OR TEST REGISTER (eee) FIELD

For the loading and storing of the Control, Debug and Test registers.

When Interpreted as Control Register Field

eee Code	Reg Name
000	CR0
010	CR2
011	CR3
Do not use any other encoding	

When Interpreted as Debug Register Field

eee Code	Reg Name
000	DR0
001	DR1
010	DR2
011	DR3
110	DR6
111	DR7
Do not use any other encoding	

When Interpreted as Test Register Field

eee Code	Reg Name
110	TR6
111	TR7
Do not use any other encoding	

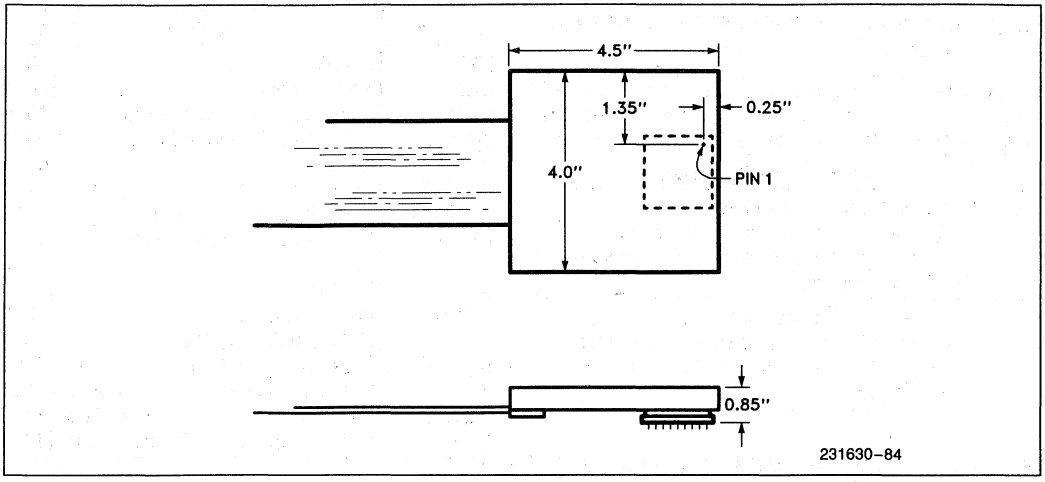


Figure 7-1. Processor Module Dimensions

7. DESIGNING FOR ICE™-386 DX EMULATOR USE

The 386 DX in-circuit emulator products are ICE-386 DX 25 MHz or 33 MHz (both referred to as ICE-386 DX emulator). The ICE-386 DX emulator probe module has several electrical and mechanical characteristics that should be taken into consideration when designing the hardware.

Capacitive loading: The ICE-386 DX emulator adds up to 25 pF to each line.

Drive requirement: The ICE-386 DX emulator adds one standard TTL load on the CLK2 line, up to one advanced low-power Schottky TTL load per control signal line, and one advanced low-power Schottky TTL load per address, byte enable, and data line. These loads are within the probe module and are driven by the probe's 386 DX component, which has standard drive and loading capability listed in the A.C. and D.C. Specification Tables in Sections 9.4 and 9.5.

Power requirement: For noise immunity the ICE-386 DX emulator probe is powered by the user system. This high-speed probe circuitry draws up to 1.5A plus the maximum I_{CC} from the user 386 DX component socket.

386 DX location and orientation: The ICE-386 DX processor module, target-adaptor cable (which does not exist for the ICE-386 DX 33 MHz emulator), and the isolation board used for extra electrical buffering of the emulator initially, require clearance as illustrated in Figures 7-1 and 7-2.

Interface Board and CLK2 speed reduction: When the ICE-386 DX emulator probe is first attached to an unverified user system, the interface board helps the ICE-386 DX emulator function in user systems with bus faults (shorted signals, etc.). After electrical verification it may be removed. Only when the interface board is installed, the user system must have a reduced CLK2 frequency of 25 MHz maximum.

Cache coherence: The ICE-386 DX emulator loads user memory by performing 386 DX component write cycles. Note that if the user system is not designed to update or invalidate its cache (if it has a cache) upon processor writes to memory, the cache could contain stale instruction code and/or data. For best use of the ICE-386 DX emulator, the user should consider designing the cache (if any) to update itself automatically when processor writes occur, or find another method of maintaining cache data coherence with main user memory.

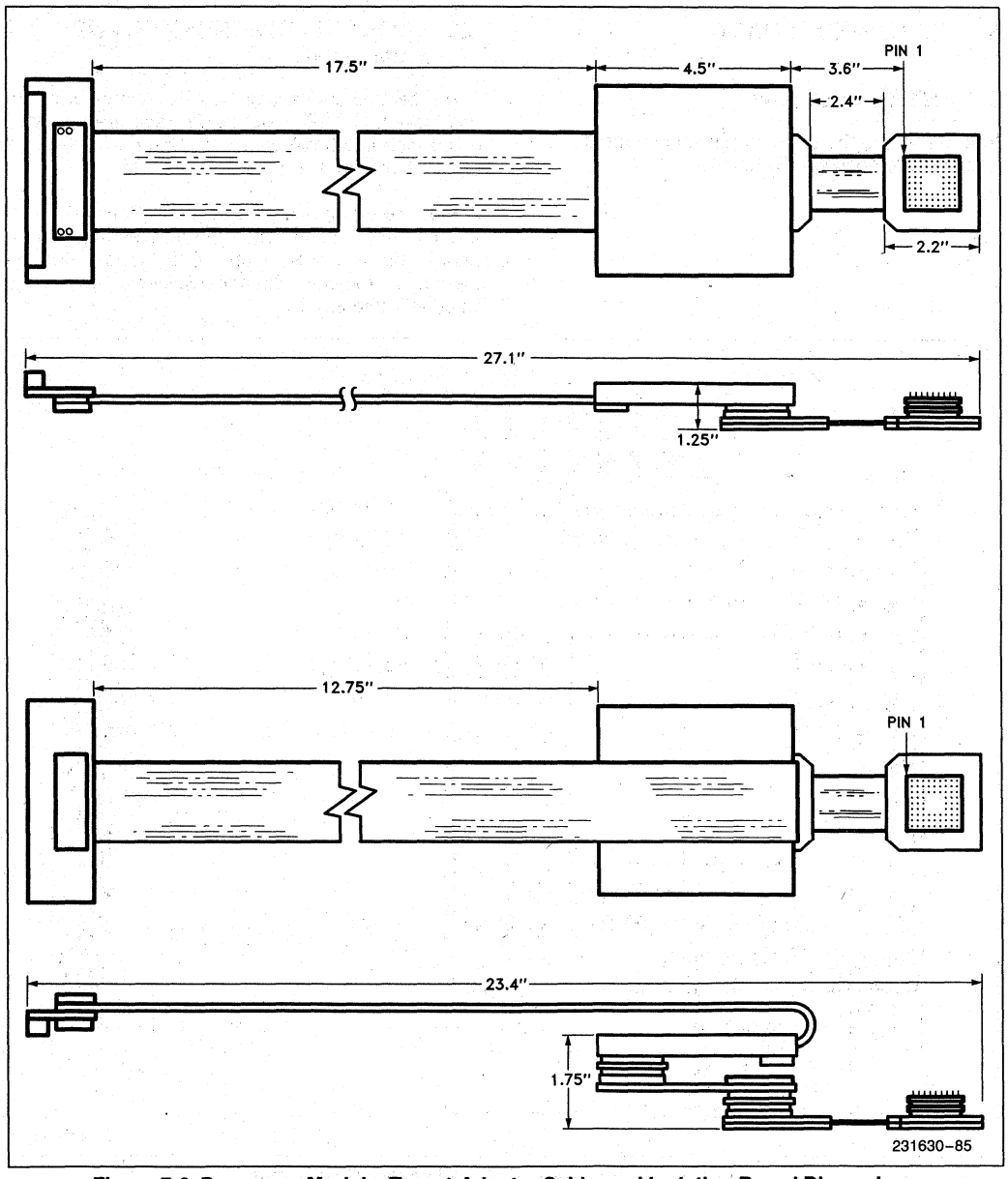


Figure 7-2. Processor Module, Target-Adapter Cable, and Isolation Board Dimensions

8. MECHANICAL DATA

8.1 INTRODUCTION

In this section, the physical packaging and its connections are described in detail.

8.2 PACKAGE DIMENSIONS AND MOUNTING

The initial 386 DX package is a 132-pin ceramic pin grid array (PGA). Pins of this package are arranged 0.100 inch (2.54mm) center-to-center, in a 14 x 14 matrix, three rows around.

A wide variety of available sockets allow low insertion force or zero insertion force mountings, and a choice of terminals such as soldertail, surface mount, or wire wrap. Several applicable sockets are listed in Table 8.1.

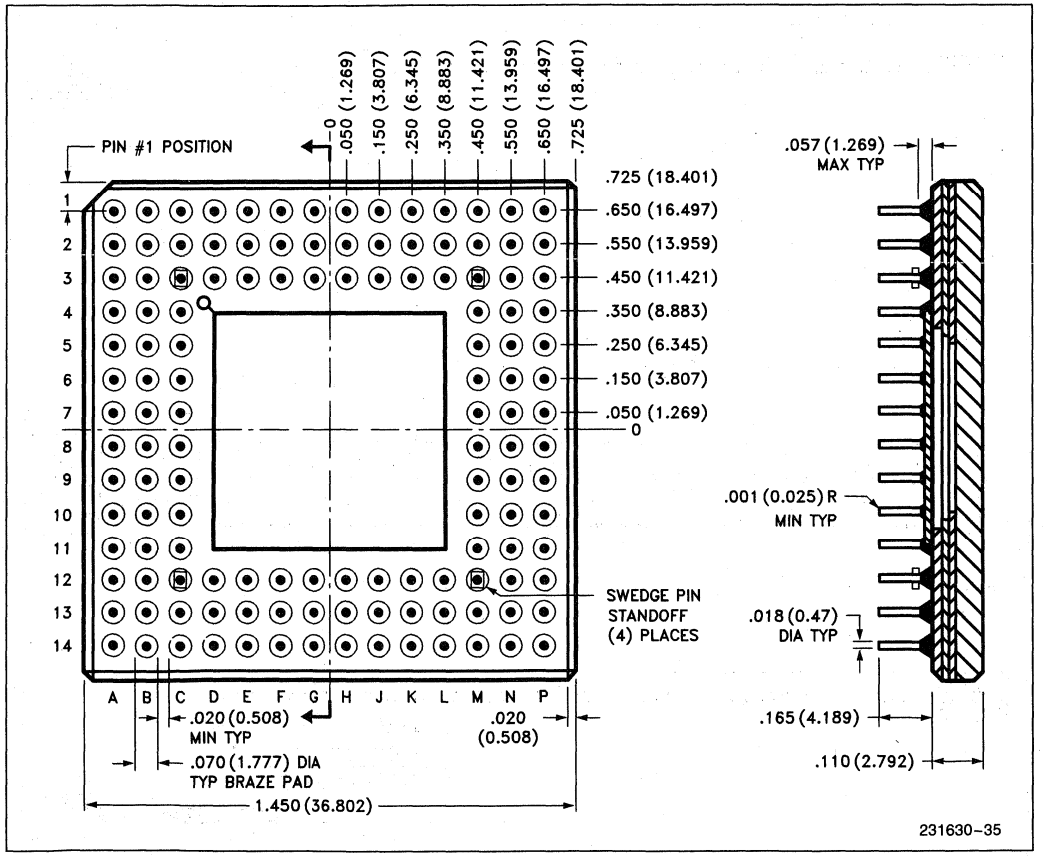


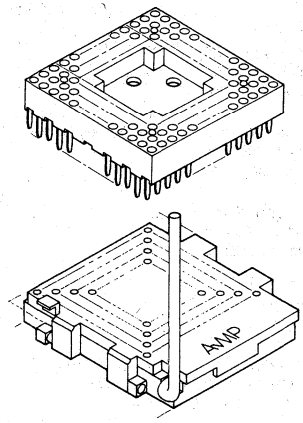
Figure 8.1. 132-Pin Ceramic PGA Package Dimensions

Table 8.1. Several Socket Options for 132-Pin PGA

* Low insertion force (LIF) soldertail 55274-1
* Amp tests indicate 50% reduction in insertion force compared to machined sockets

Other socket options
* Zero insertion force (ZIF) soldertail 55583-1
* Zero insertion force (ZIF) Burn-in version 55573-2

Amp Incorporated
(Harrisburg, PA 17105 U.S.A.
Phone 717-564-0100)



231630-45
Cam handle locks in low profile position when substrate is installed (handle UP for open and DOWN for closed positions)

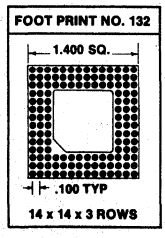
courtesy Amp Incorporated

Peel-A-Way™ Mylar and Kapton Socket Terminal Carriers
* Low insertion force surface mount CS132-37TG
* Low insertion force soldertail CS132-01TG
* Low insertion force wire-wrap CS132-02TG (two level) CS132-03TG (three-level)
* Low insertion force press-fit CS132-05TG

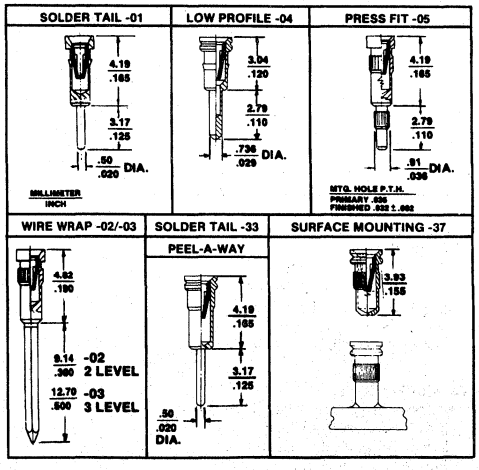
Advanced Interconnections
(5 Division Street
Warwick, RI 02818 U.S.A.
Phone 401-885-0485)

Peel-A-Way Carrier No. 132:
Kapton Carrier is KS132
Mylar Carrier is MS132

Molded Plastic Body KS132 is shown below:



231630-46



231630-47
courtesy Advanced Interconnections
(Peel-A-Way Terminal Carriers
U.S. Patent No. 4442938)

5

Table 8.1. Several Socket Options for 132-Pin PGA (Continued)

PIN GRID ARRAY
DECOUPLING SOCKETS

* Low insertion force soldertail 0.125 length PGD-005-1A1 Finish: Term/Contact Tin-Lead/Gold
* Low insertion force soldertail 0.180 length PGD-005-1B1 Finish: Term/Contact: Tin-Lead/Gold
* Low insertion 3 level Wire/Wrap PGD-005-1C1 Finish: Term/Contact Tin-Lead/Gold

Includes 0.10 μF & 1.0 μF Decoupling Capacitors

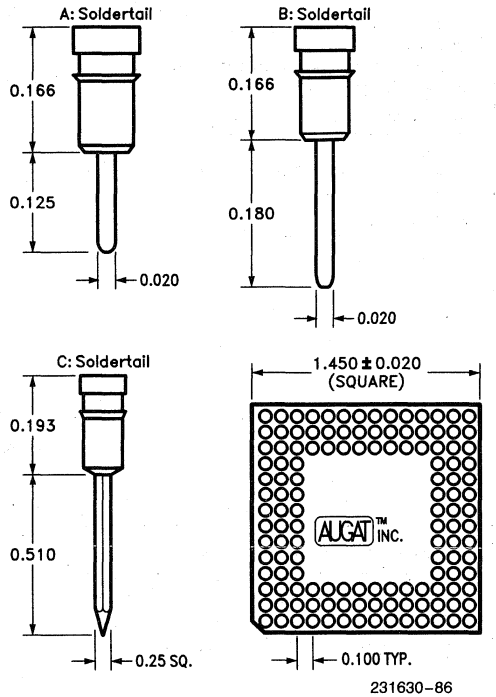
VisinPak Kapton Carrier

PKC Series

Pin Grid Array

PGM (Plastic) or PPS (Glass Epoxy) Series

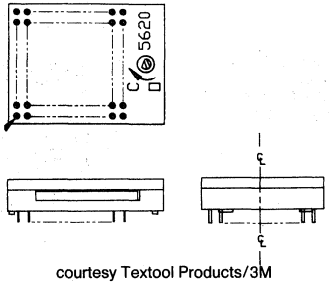
AUGAT INC.
33 Perry Ave., P.O. Box 779 Attleboro, MA 02703
TECHNICAL INFORMATION: (508) 222-2202
CUSTOMER SERVICE: (508) 699-9800



231630-86

* Low insertion force socket soldertail (for production use)
  2XX-6576-00-3308 (new style)
  2XX-6003-00-3302 (older style)
* Zero insertion force soldertail (for test and burn-in use)
  2XX-6568-00-3302

Textool Products
Electronic Products Division/3M
(1410 West Pioneer Drive
Irving, Texas 75601 U.S.A.
Phone 214-259-2676)



courtesy Textool Products/3M

231630-48

8.3 PACKAGE THERMAL SPECIFICATION

The 386 DX is specified for operation when case temperature is within the range of 0°C–85°C. The case temperature may be measured in any environment, to determine whether the 386 DX is within specified operating range.

The PGA case temperature should be measured at the center of the top surface opposite the pins, as in Figure 8.2.

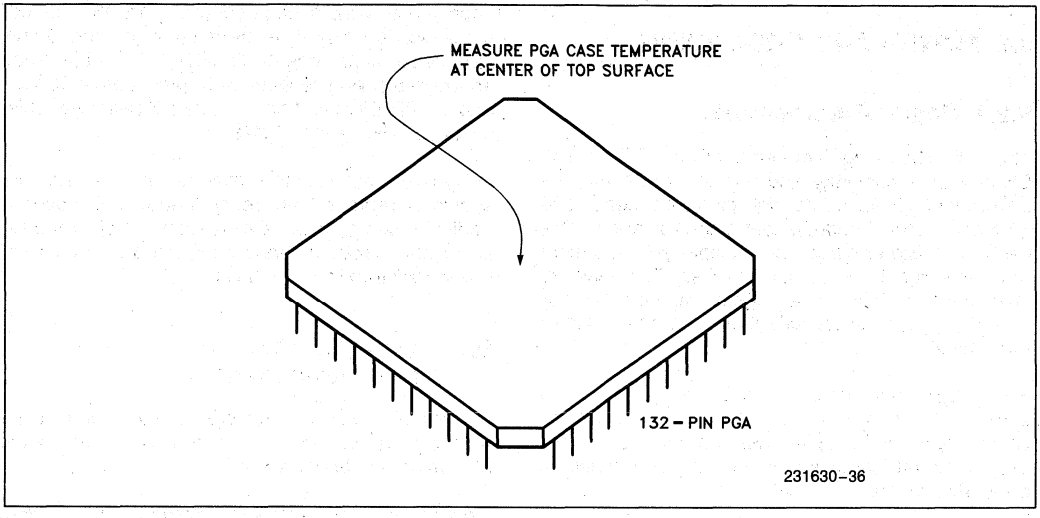
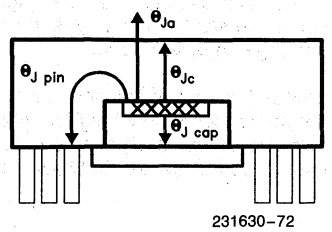


Figure 8.2. Measuring 386™ DX PGA Case Temperature

5

Table 8.2. 386™ DX PGA Package Thermal Characteristics

Parameter	Thermal Resistance — °C/Watt Airflow — ft./min (m/sec) 0 (0)	50 (0.25)	100 (0.50)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)
θ Junction-to-Case (case measured as Fig. 8-2)	2	2	2	2	2	2	2
θ Case-to-Ambient (no heatsink)	19	18	17	15	12	10	9
θ Case-to-Ambient (with omnidirectional heatsink)	16	15	14	12	9	7	6
θ Case-to-Ambient (with unidirectional heatsink)	15	14	13	11	8	6	5



NOTES:
1. Table 8.2 applies to 386™ DX PGA plugged into socket or soldered directly into board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.
3. $\theta_{J\text{-}CAP}$ = 4°C/w (approx.)
   $\theta_{J\text{-}PIN}$ = 4°C/w (inner pins) (approx.)
   $\theta_{J\text{-}PIN}$ = 8°C/w (outer pins) (approx.)
4. $T_A = T_C - P * \theta_{CA}$ (ambient temperature)

9. ELECTRICAL DATA

9.1 INTRODUCTION

The following sections describe recommended electrical connections for the 386 DX, and its electrical specifications.

9.2 POWER AND GROUNDING

9.2.1 Power Connections

The 386 DX is implemented in CHMOS III and CHMOS IV technology and has modest power requirements. However, its high clock frequency and 72 output buffers (address, data, control, and HLDA) can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 20 V_{CC} and 21 V_{SS} pins separately feed functional units of the 386 DX.

Power and ground connections must be made to all external V_{CC} and GND pins of the 386 DX. On the circuit board, all V_{CC} pins must be connected on a V_{CC} plane. All V_{SS} pins must be likewise connected on a GND plane.

9.2.2 Power Decoupling Recommendations

Liberal decoupling capacitance should be placed near the 386 DX. The 386 DX driving its 32-bit parallel address and data buses at high frequencies can cause transient power surges, particularly when driving large capacitive loads.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the 386 DX and decoupling capacitors as much as possible. Capacitors specifically for PGA packages are also commercially available, for the lowest possible inductance.

9.2.3 Resistor Recommendations

The ERROR# and BUSY# inputs have resistor pullups of approximately 20 KΩ built-in to the 386 DX to keep these signals negated when no 387 DX coprocessor is present in the system (or temporarily removed from its socket). The BS16# input also has an internal pullup resistor of approximately 20 KΩ, and the PEREQ input has an internal pulldown resistor of approximately 20 KΩ.

In typical designs, the external pullup resistors shown in Table 9-1 are recommended. However, a particular design may have reason to adjust the resistor values recommended here, or alter the use of pullup resistors in other ways.

9.2.4 Other Connection Recommendations

For reliable operation, always connect unused inputs to an appropriate signal level. N.C. pins should always remain unconnected.

Particularly when not using interrupts or bus hold, (as when first prototyping, perhaps) prevent any chance of spurious activity by connecting these associated inputs to GND:

Pin	Signal
B7	INTR
B8	NMI
D14	HOLD

If not using address pipelining, pullup D13 NA# to V_{CC}.

If not using 16-bit bus size, pullup C14 BS16# to V_{CC}.

Pullups in the range of 20 KΩ are recommended.

Table 9-1. Recommended Resistor Pullups to V_{CC}

Pin and Signal	Pullup Value	Purpose
E14 ADS#	20 KΩ ±10%	Lightly Pull ADS# Negated During 386™ DX Hold Acknowledge States
C10 LOCK#	20 KΩ ±10%	Lightly Pull LOCK# Negated During 386™ DX Hold Acknowledge States

9.3 MAXIMUM RATINGS

Table 9-2. Maximum Ratings

Parameter	386™ DX 20, 25, 33 MHz Maximum Rating
Storage Temperature	−65°C to +150°C
Case Temperature Under Bias	−65°C to +110°C
Supply Voltage with Respect to V_{SS}	−0.5V to +6.5V
Voltage on Other Pins	−0.5V to V_{CC} + 0.5V

Table 9-2 is a stress rating only, and functional operation at the maximums is not guaranteed. Functional operating conditions are given in **9.4 D.C. Specifications** and **9.5 A.C. Specifications**.

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the 386 DX contains protective circuitry to resist damage from static electric discharge, always take precautions to avoid high static voltages or electric fields.

9.4 D.C. SPECIFICATIONS

Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to 85°C

Table 9-3. 386™ DX D.C. Characteristics

Symbol	Parameter	386™ DX 20 MHz, 25 MHz, 33 MHz		Unit	Test Conditions
		Min	Max		
V_{IL}	Input Low Voltage	−0.3	0.8	V	(Note 1)
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{ILC}	CLK2 Input Low Voltage	−0.3	0.8	V	(Note 1)
V_{IHC}	CLK2 Input High Voltage 20 MHz 25 MHz and 33 MHz	 V_{CC} − 0.8 3.7	 V_{CC} + 0.3 V_{CC} + 0.3	 V V	
V_{OL}	Output Low Voltage I_{OL} = 4 mA: A2–A31, D0–D31 I_{OL} = 5 mA: BE0#–BE3#, W/R#, D/C#, M/IO#, LOCK#, ADS#, HLDA		 0.45 0.45	 V V	
V_{OH}	Output High Voltage I_{OH} = 1 mA: A2–A31, D0–D31 I_{OH} = 0.9 mA: BE0#–BE3#, W/R#, D/C#, M/IO#, LOCK#, ADS#, HLDA	 2.4 2.4		 V V	
I_{LI}	Input Leakage Current (For All Pins except BS16#, PEREQ, BUSY#, and ERROR#)		±15	µA	0V ≤ V_{IN} ≤ V_{CC}
I_{IH}	Input Leakage Current (PEREQ Pin)		200	µA	V_{IH} = 2.4V (Note 2)
I_{IL}	Input Leakage Current (BS16#, BUSY#, and ERROR# Pins)		−400	µA	V_{IL} = 0.45 (Note 3)
I_{LO}	Output Leakage Current		±15	µA	0.45V ≤ V_{OUT} ≤ V_{CC}
I_{CC}	Supply Current CLK2 = 40 MHz: with 20 MHz 386™ DX CLK2 = 50 MHz: with 25 MHz 386™ DX CLK2 = 66 MHz: with 33 MHz 386™ DX		 500 550 550	 mA mA mA	 I_{CC} Typ. = 460 mA I_{CC} Typ. = 500 mA I_{CC} Typ. = 400 mA
C_{IN}	Input or I/O Capacitance		10	pF	F_C = 1 MHz (Note 4)
C_{OUT}	Output Capacitance		12	pF	F_C = 1 MHz (Note 4)
C_{CLK}	CLK2 Capacitance		20	pF	F_C = 1 MHz (Note 4)

NOTES:
1. The min value, −0.3, is not 100% tested.
2. PEREQ input has an internal pulldown resistor.
3. BS16#, BUSY# and ERROR# inputs each have an internal pullup resistor.
4. Not 100% tested.

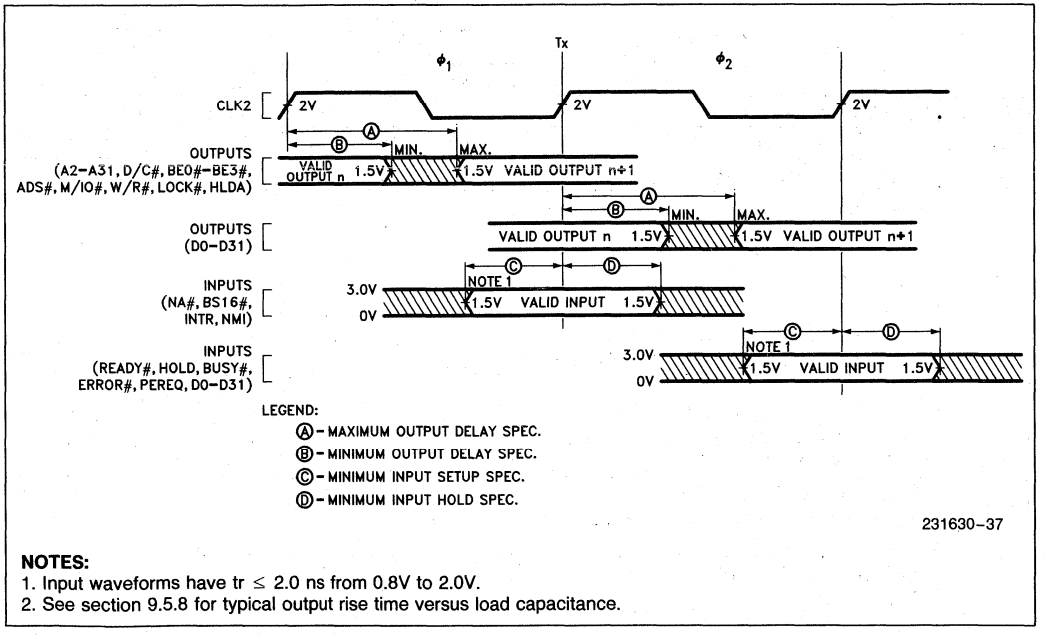
9.5 A.C. SPECIFICATIONS

9.5.1 A.C. Spec Definitions

The A.C. specifications, given in Tables 9-4, 9-5, and 9-6, consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the CLK2 rising edge crossing the 2.0V level.

A.C. spec measurement is defined by Figure 9-1. Inputs must be driven to the voltage levels indicated by Figure 9-1 when A.C. specifications are measured. 386 DX output delays are specified with minimum and maximum limits, measured as shown. The minimum 386 DX delay times are hold times provided to external circuitry. 386 DX input setup and hold times are specified as minimums, defining the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct 386 DX operation.

Outputs NA#, W/R#, D/C#, M/IO#, LOCK#, BE0#–BE3#, A2–A31 and HLDA only change at the beginning of phase one. D0–D31 (write cycles) only change at the beginning of phase two. The READY#, HOLD, BUSY#, ERROR#, PEREQ and D0–D31 (read cycles) inputs are sampled at the beginning of phase one. The NA#, BS16#, INTR and NMI inputs are sampled at the beginning of phase two.



NOTES:
1. Input waveforms have tr ≤ 2.0 ns from 0.8V to 2.0V.
2. See section 9.5.8 for typical output rise time versus load capacitance.

Figure 9-1. Drive Levels and Measurement Points for A.C. Specifications

9.5.2 A.C. Specification Tables

Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C

Table 9-4. 33 MHz 386™ DX A.C. Characteristics

Symbol	Parameter	33 MHz 386™ DX		Unit	Ref. Fig.	Notes
		Min	Max			
	Operating Frequency	8	33.3	MHz		Half of CLK2 Frequency
t1	CLK2 Period	15.0	62.5	ns	9-3	
t2a	CLK2 High Time	6.25		ns	9-3	at 2V
t2b	CLK2 High Time	4.5		ns	9-3	at 3.7V
t3a	CLK2 Low Time	6.25		ns	9-3	at 2V
t3b	CLK2 Low Time	4.5		ns	9-3	at 0.8V
t4	CLK2 Fall Time		4	ns	9-3	3.7V to 0.8V (Note 3)
t5	CLK2 Rise Time		4	ns	9-3	0.8V to 3.7V (Note 3)
t6	A2–A31 Valid Delay	4	15	ns	9-5	C_L = 50 pF
t7	A2–A31 Float Delay	4	20	ns	9-6	(Note 1)
t8	BE0#–BE3#, LOCK# Valid Delay	4	15	ns	9-5	C_L = 50 pF
t9	BE0#–BE3#, LOCK# Float Delay	4	20	ns	9-6	(Note 1)
t10	W/R#, M/IO#, D/C#, Valid Delay	4	15	ns	9-5	C_L = 50 pF
t10a	ADS# Valid Delay	4	14.5	ns	9-5	C_L = 50 pF
t11	W/R#, M/IO#, D/C#, ADS# Float Delay	4	20	ns	9-6	(Note 1)
t12	D0–D31 Write Data Valid Delay	7	24	ns	9-5a	C_L = 50 pF, (Note 4)
t12a	D0–D31 Write Data Hold Time	2			9-5b	C_L = 50 pF
t13	D0–D31 Float Delay	4	17	ns	9-6	(Note 1)
t14	HLDA Valid Delay	4	20	ns	9-6	C_L = 50 pF
t15	NA# Setup Time	5		ns	9-4	
t16	NA# Hold Time	2		ns	9-4	
t17	BS16# Setup Time	5		ns	9-4	
t18	BS16# Hold Time	2		ns	9-4	
t19	READY# Setup Time	7		ns	9-4	
t20	READY# Hold Time	4		ns	9-4	

9.5.2 A.C. Specification Tables (Continued)

Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C

Table 9-4. 33 MHz 386™ DX A.C. Characteristics (Continued)

Symbol	Parameter	33 MHz 386™ DX		Unit	Ref. Fig.	Notes
		Min	Max			
t21	D0–D31 Read Setup Time	5		ns	9-4	
t22	D0–D31 Read Hold Time	3		ns	9-4	
t23	HOLD Setup Time	11		ns	9-4	
t24	HOLD Hold Time	2		ns	9-4	
t25	RESET Setup Time	5		ns	9-7	
t26	RESET Hold Time	2		ns	9-7	
t27	NMI, INTR Setup Time	5		ns	9-4	(Note 2)
t28	NMI, INTR Hold Time	5		ns	9-4	(Note 2)
t29	PEREQ, ERROR#, BUSY# Setup Time	5		ns	9-4	(Note 2)
t30	PEREQ, ERROR#, BUSY# Hold Time	4		ns	9-4	(Note 2)

NOTES:

1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.
2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.
3. Rise and fall times are not tested.
4. Min. time not 100% tested.

9.5.2 A.C. Specification Tables (Continued)

Functional Operating Range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0°C$ to $+85°C$

Table 9-5. 25 MHz 386™ DX A.C. Characteristics

Symbol	Parameter	25 MHz 386™ DX		Unit	Ref. Fig.	Notes
		Min	Max			
	Operating Frequency	4	25	MHz		Half of CLK2 Frequency
t1	CLK2 Period	20	125	ns	9-3	
t2a	CLK2 High Time	7		ns	9-3	at 2V
t2b	CLK2 High Time	4		ns	9-3	at 3.7V
t3a	CLK2 Low Time	7		ns	9-3	at 2V
t3b	CLK2 Low Time	5		ns	9-3	at 0.8V
t4	CLK2 Fall Time		7	ns	9-3	3.7V to 0.8V
t5	CLK2 Rise Time		7	ns	9-3	0.8V to 3.7V
t6	A2–A31 Valid Delay	4	21	ns	9-5	$C_L = 50$ pF
t7	A2–A31 Float Delay	4	30	ns	9-6	(Note 1)
t8	BE0#–BE3# Valid Delay	4	24	ns	9-5	$C_L = 50$ pF
t8a	LOCK# Valid Delay	4	21	ns	9-5	$C_L = 50$ pF
t9	BE0#–BE3#, LOCK# Float Delay	4	30	ns	9-6	(Note 1)
t10	W/R#, M/IO#, D/C#, ADS# Valid Delay	4	21	ns	9-5	$C_L = 50$ pF
t11	W/R#, M/IO#, D/C#, ADS# Float Delay	4	30	ns	9-6	(Note 1)
t12	D0–D31 Write Data Valid Delay	7	27	ns	9-5a	$C_L = 50$ pF
t12a	D0–D31 Write Data Hold Time	2			9-5b	$C_L = 50$ pF
t13	D0–D31 Float Delay	4	22	ns	9-6	(Note 1)
t14	HLDA Valid Delay	4	22	ns	9-6	$C_L = 50$ pF
t15	NA# Setup Time	7		ns	9-4	
t16	NA# Hold Time	3		ns	9-4	
t17	BS16# Setup Time	7		ns	9-4	
t18	BS16# Hold Time	3		ns	9-4	
t19	READY# Setup Time	9		ns	9-4	
t20	READY# Hold Time	4		ns	9-4	

9.5.2 A.C. Specification Tables (Continued)

Functional Operating Range: $V_{CC} = 5V \pm 5\%$; $T_{CASE} = 0°C$ to $+85°C$

Table 9-5. 25 MHz 386™ DX A.C. Characteristics (Continued)

Symbol	Parameter	25 MHz 386™ DX		Unit	Ref. Fig.	Notes
		Min	Max			
t21	D0–D31 Read Setup Time	7		ns	9-4	
t22	D0–D31 Read Hold Time	5		ns	9-4	
t23	HOLD Setup Time	15		ns	9-4	
t24	HOLD Hold Time	3		ns	9-4	
t25	RESET Setup Time	10		ns	9-7	
t26	RESET Hold Time	3		ns	9-7	
t27	NMI, INTR Setup Time	6		ns	9-4	(Note 2)
t28	NMI, INTR Hold Time	6		ns	9-4	(Note 2)
t29	PEREQ, ERROR#, BUSY# Setup Time	6		ns	9-4	(Note 2)
t30	PEREQ, ERROR#, BUSY# Hold Time	5		ns	9-4	(Notes 2, 3)

NOTES:

1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.
2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.
3.

	Symbol	Parameter	Min
$T_C = 0°C$	t30	PEREQ, ERROR#, BUSY# Hold Time	4
$T_C = +85°C$	t30	PEREQ, ERROR#, BUSY# Hold Time	5

9.5.2 A.C. Specification Tables (Continued)

Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C

Table 9.6. 20 MHz 386™ DX A.C. Characteristics

Symbol	Parameter	20 MHz 386™ DX		Unit	Ref. Fig.	Notes
		Min	Max			
	Operating Frequency	4	20	MHz		Half of CLK2 Frequency
t_1	CLK2 Period	25	125	ns	9-3	
t_{2a}	CLK2 High Time	8		ns	9-3	at 2V
t_{2b}	CLK2 High Time	5		ns	9-3	at (V_{CC} − 0.8V)
t_{3a}	CLK2 Low Time	8		ns	9-3	at 2V
t_{3b}	CLK2 Low Time	6		ns	9-3	at 0.8V
t_4	CLK2 Fall Time		8	ns	9-3	(V_{CC} − 0.8V) to 0.8V
t_5	CLK2 Rise Time		8	ns	9-3	0.8V to (V_{CC} − 0.8V)
t_6	A2–A31 Valid Delay	4	30	ns	9-5	C_L = 120 pF
t_7	A2–A31 Float Delay	4	32	ns	9-6	(Note 1)
t_8	BE0#–BE3#, LOCK# Valid Delay	4	30	ns	9-5	C_L = 75 pF
t_9	BE0#–BE3#, LOCK# Float Delay	4	32	ns	9-6	(Note 1)
t_{10}	W/R#, M/IO#, D/C#, ADS# Valid Delay	6	28	ns	9-5	C_L = 75 pF
t_{11}	W/R#, M/IO#, D/C#, ADS# Float Delay	6	30	ns	9-6	(Note 1)
t_{12}	D0–D31 Write Data Valid Delay	4	38	ns	9-5c	C_L = 120 pF
t_{13}	D0–D31 Float Delay	4	27	ns	9-6	(Note 1)
t_{14}	HLDA Valid Delay	6	28	ns	9-6	C_L = 75 pF
t_{15}	NA# Setup Time	9		ns	9-4	
t_{16}	NA# Hold Time	14		ns	9-4	
t_{17}	BS16# Setup Time	13		ns	9-4	
t_{18}	BS16# Hold Time	21		ns	9-4	
t_{19}	READY# Setup Time	12		ns	9-4	
t_{20}	READY# Hold Time	4		ns	9-4	
t_{21}	D0–D31 Read Setup Time	11		ns	9-4	
t_{22}	D0–D31 Read Hold Time	6		ns	9-4	
t_{23}	HOLD Setup Time	17		ns	9-4	
t_{24}	HOLD Hold Time	5		ns	9-4	
t_{25}	RESET Setup Time	12		ns	9-7	

9.5.2 A.C. Specification Tables (Continued)

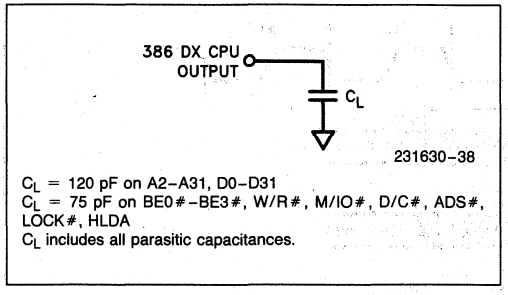
Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C

Table 9-6. 20 MHz 386™ DX A.C. Characteristics (Continued)

Symbol	Parameter	20 MHz 386™ DX		Unit	Ref. Fig.	Notes
		Min	Max			
t_{26}	RESET Hold Time	4		ns	9-7	
t_{27}	NMI, INTR Setup Time	16		ns	9-4	(Note 2)
t_{28}	NMI, INTR Hold Time	16		ns	9-4	(Note 2)
t_{29}	PEREQ, ERROR#, BUSY# Setup Time	14		ns	9-4	(Note 2)
t_{30}	PEREQ, ERROR#, BUSY# Hold Time	5		ns	9-4	(Note 2)

NOTES:
1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.
2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.

9.5.3 A.C. Test Loads



C_L = 120 pF on A2–A31, D0–D31
C_L = 75 pF on BE0#–BE3#, W/R#, M/IO#, D/C#, ADS#, LOCK#, HLDA
C_L includes all parasitic capacitances.

Figure 9-2. A.C. Test Load

9.5.4 A.C. Timing Waveforms

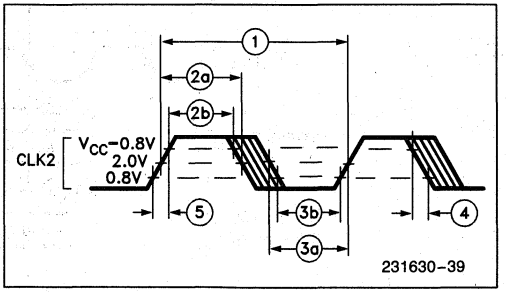


Figure 9-3. CLK2 Timing

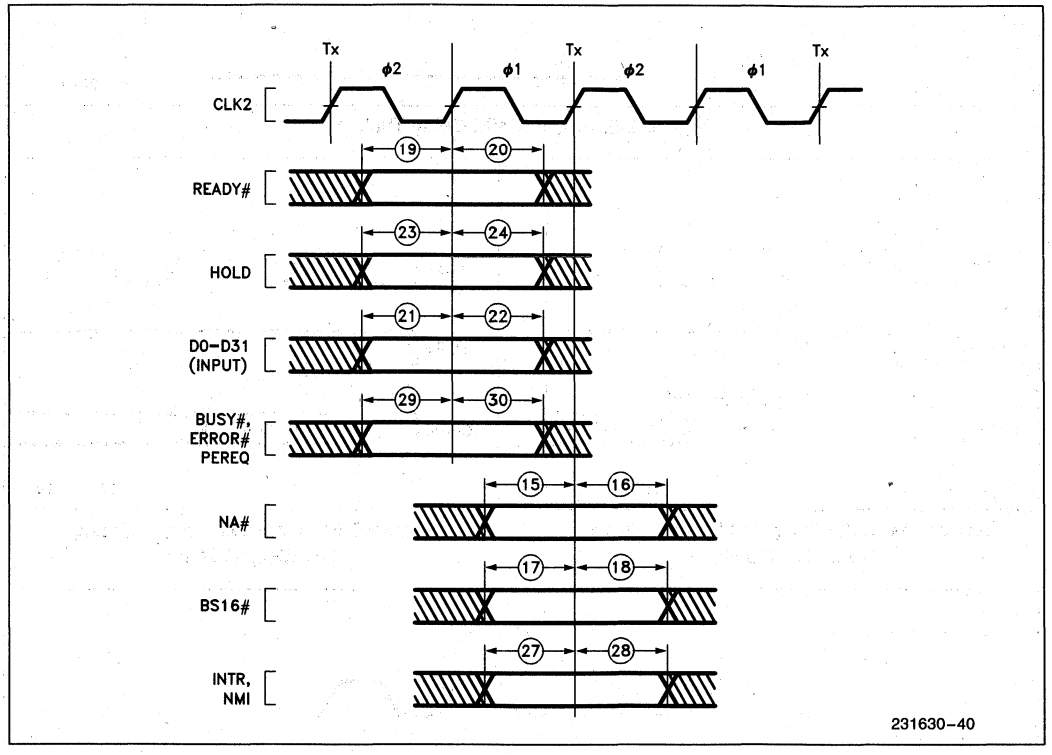


Figure 9-4. Input Setup and Hold Timing

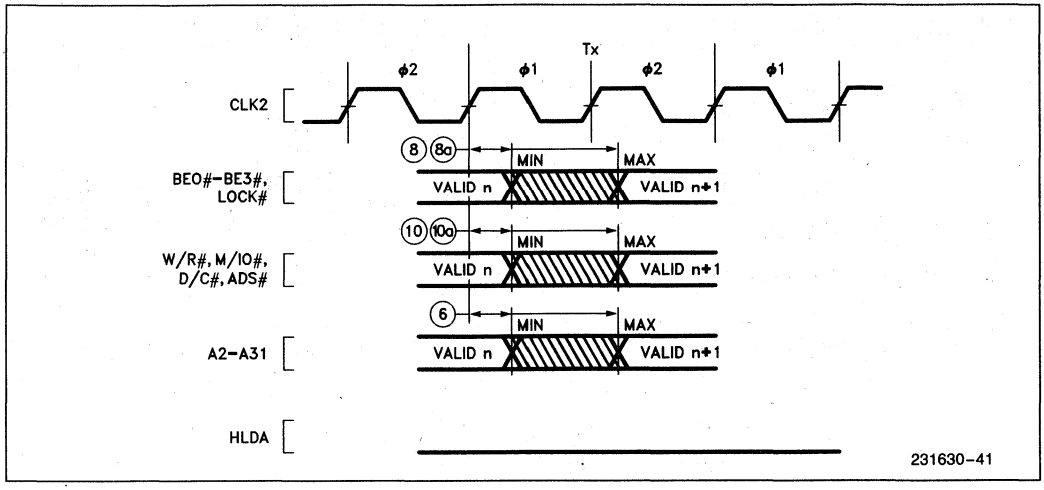


Figure 9-5. Output Valid Delay Timing

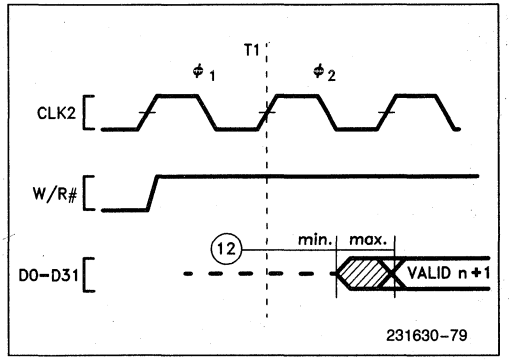


Figure 9-5a. Write Data Valid Delay Timing (25 MHz, 33 MHz)

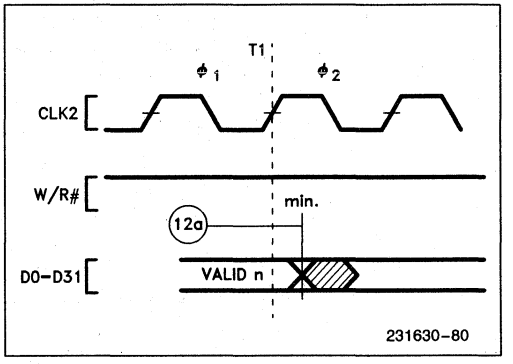


Figure 9-5b. Write Data Hold Timing (25 MHz, 33 MHz)

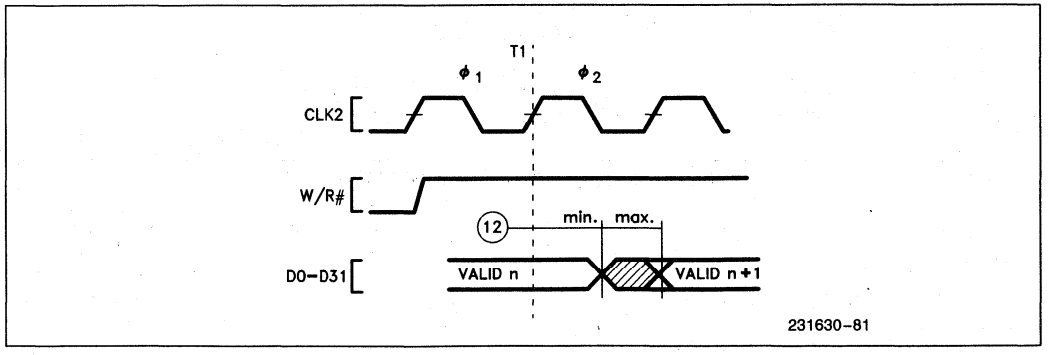
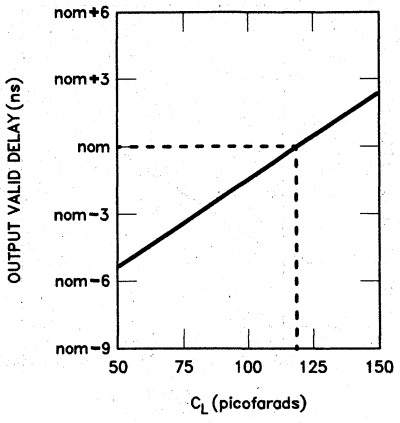


Figure 9-5c. Write Data Valid Delay Timing (20 MHz)

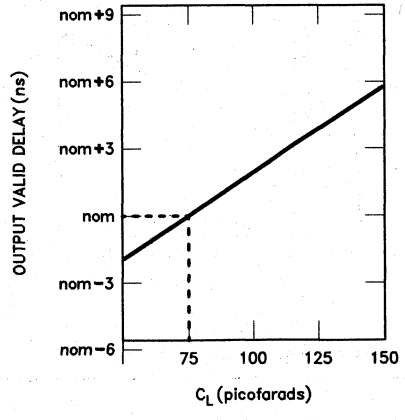
9.5.5 Typical Output Valid Delay Versus Load Capacitance at Maximum Operating Temperature (C_L = 120 pF)



231630-77

NOTE:
This graph will not be linear outside of the C_L range shown.

9.5.6 Typical Output Valid Delay Versus Load Capacitance at Maximum Operating Temperature (C_L = 75 pF)

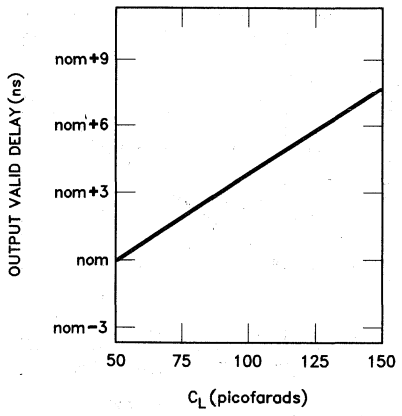


231630-82

NOTE:
This graph will not be linear outside of the C_L range shown.

5

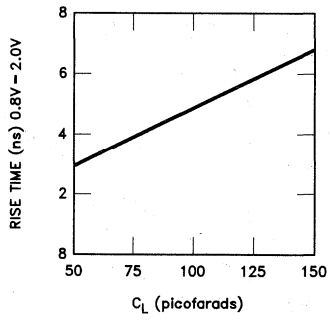
9.5.7 Typical Output Valid Delay Versus Load Capacitance at Maximum Operating Temperature (C_L = 50 pF)



231630-83

NOTE:
This graph will not be linear outside of the C_L range shown.

9.5.8 Typical Output Rise Time Versus Load Capacitance at Maximum Operating Temperature



231630-78

NOTE:
This graph will not be linear outside of the C_L range shown.

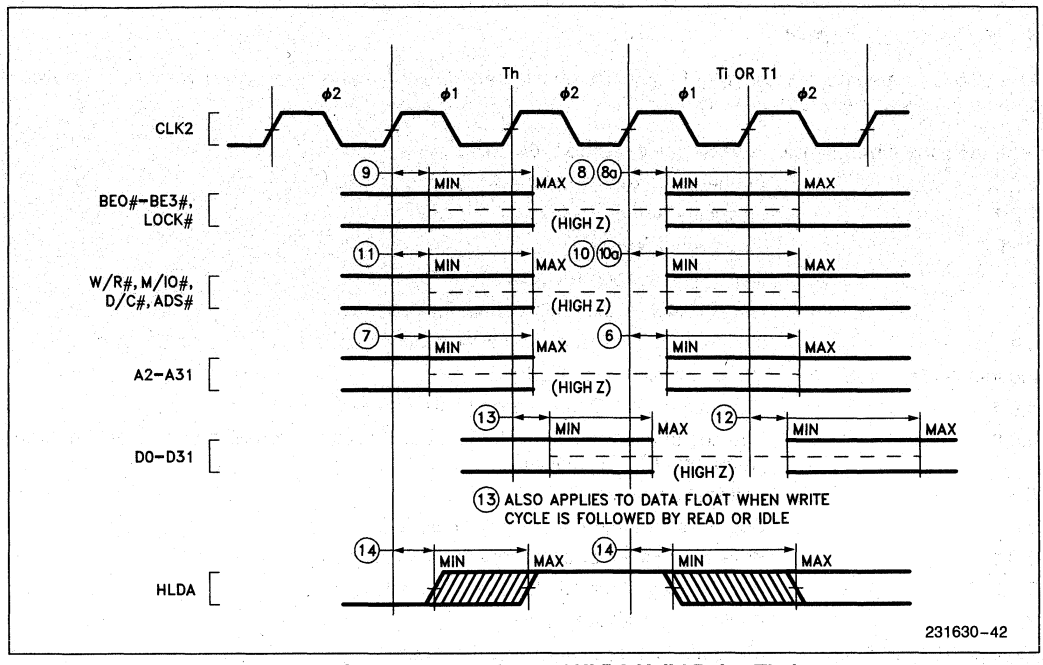
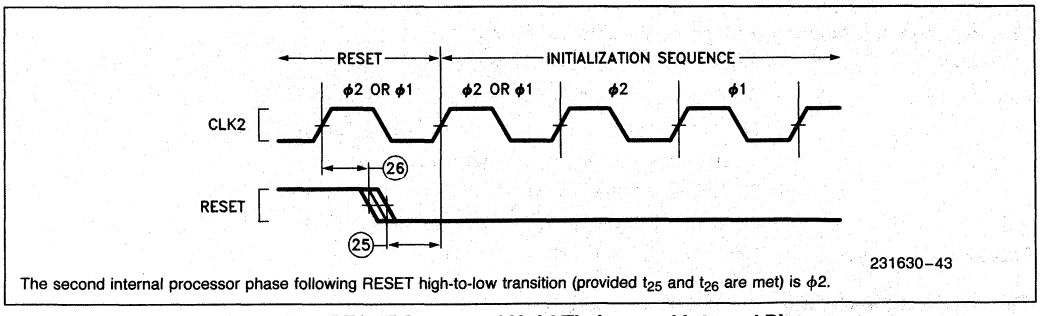


Figure 9-6. Output Float Delay and HLDA Valid Delay Timing



The second internal processor phase following RESET high-to-low transition (provided t_{25} and t_{26} are met) is $\phi 2$.

Figure 9-7. RESET Setup and Hold Timing, and Internal Phase

10.0 Revision History

This 386 DX data sheet, version -005, contains updates and improvements to previous versions. A revision summary is listed here for your convenience.

The sections significantly revised since version -001 are:

2.9.6	Sequence of exception checking table added.
2.9.7	Instruction restart revised.
2.11.2	TLB testing revised.
2.12	Debugging support revised.
3.1	LOCK prefix restricted to certain instructions.
4.4.3.3	I/O privilege level and I/O permission bitmap added.
Figures 4-15a, 4-15b	I/O permission bitmap added.
4.6.4	Protection and I/O permission bitmap revised.
4.6.6	Entering and leaving virtual 8086 mode through task switches, trap and interrupt gates, and IRET explained.
5.6	Self-test signature stored in EAX.
5.8	Coprocessor interface description added.
5.8.1	Software testing for coprocessor presence added.
Table 6-3	PGA package thermal characteristics added.
7.	Designing for ICE-386 revised.
Figures 7-8, 7-9, 7-10	ICE-386 clearance requirements added.
6.2.3.4	Encoding of 32-bit address mode with no "sib" byte corrected.

The sections significantly revised since version -002 are:

Table 2-5	Interrupt vector assignments updated.
Figure 4-15a	Bit_map_offset must be less than or equal to DFFFH.
Figure 5-28	386 DX outputs remain in their reset state during self-test.
5.7	Component and revision identifier history updated.
9.4	20 MHz D.C. specifications added.
9.5	16 MHz A.C. specifications updated. 20 MHz A.C. specifications added.
Table 6-1	Clock counts updated.

The sections significantly revised since version -003 are:

Table 2-6b	Interrupt priorities 2 and 3 interchanged.
2.9.8	Double page faults do not raise double fault exception.
Figure 4-5	Maximum-sized segments must have segments $Base_{11..0} = 0$.
5.4.3.4	BS16# timing corrected.
Figures 5-16, 5-17, 5-19, 5-22	BS16# timing corrected. BS16# must not be asserted once NA# has been sampled asserted in the current bus cycle.
9.5	16 MHz and 20 MHz A.C. specifications revised. All timing parameters are now guaranteed at 1.5V test levels. The timing parameters have been adjusted to remain compatible with previous 0.8V/2.0V specifications.

The sections significantly revised since version -004 are:

Chapter 4	25 MHz Clock data included.
Table 2-4	Segment Register Selection Rules updated.
5.4.4	Interrupt Acknowledge Cycles discussion corrected.
Table 5-10	Additional Stepping Information added.
Table 9-3	I_{CC} values updated.
9.5.2	Table for 25 MHz A.C. Characteristics added. A.C. Characteristics tables reordered.
Figure 9-5	Output Valid Delay Timing Figure reconfigured. Partial data now provided in additional Figures 9-5a and 9-5b.
Table 6-1	Clock counts updated and formats corrected.

The sections significantly revised since version -005 are:

Table of Contents	Simplified.
Chapter 1	Pin Assignment.
2.3.6	Control Register 0.
Table 2-4	Segment override prefixes possible.
Figure 4-6	Note added.
Figure 4-7	Note added.
5.2.3	Data bus state at end of cycle.
5.2.8.4	Coprocessor error.
5.5.3	Bus activity during and following reset.
Figure 5-28	ERROR#.
Chapter 6	Moved forward in datasheet.
Chapter 7	Moved forward in datasheet.
Chapter 8	Upgraded to chapter.
Table 9-3	25 MHz I_{CC} Typ. value corrected.
Table 9-3	33 MHz D.C. Specifications added.
Table 9-4	33 MHz A.C. Specifications added.
Figure 9-5	t8a and t10a added.
Figure 9-5c	Added.
9.5.6	Added derating for C_L = 75 pF.
9.5.7	Added derating for C_L = 50 pF.
Figure 9.6	t8a and t10a added.

The sections significantly revised since version -006 are:

2.3.4	Alignment of maximum sized segments.
2.9.8	Double page faults do not raise double fault exception.
5.5.3	ERROR# and BUSY# sampling after RESET.
Figure 5-21	BS16# timing altered.
Figure 5-26	READY# timing altered.
Figure 5-28	ERROR# timing corrected.
6.2.3.1	Corrected Encoding of Register Field Chart.
Chapter 7	Updated ICE-386 DX information.
9.5.2	Remove preliminary stamp on 25 MHz A.C. Specifications.
9.5.2	Remove preliminary stamp on 33 MHz A.C. Specifications.

The sections significantly revised since version -007 are:

Table of Contents	Page numbers revised.
Figure 5-15	BS16# timing altered.
Figure 5-22	Previous cycle, T2 changed to Idle cycle, Ti.
6.1	Note about wait states added.
Table 6-1	Opcodes for AND, OR, and XOR instructions corrected.
Table 6-1	Bits 3, 4, and 5 of the "mod r/m" byte corrected for the LTR instruction.
Table 8-2	Reference to Figure 6-4 should be reference Figure 8-2.
Table 8-2	Note #4 added.

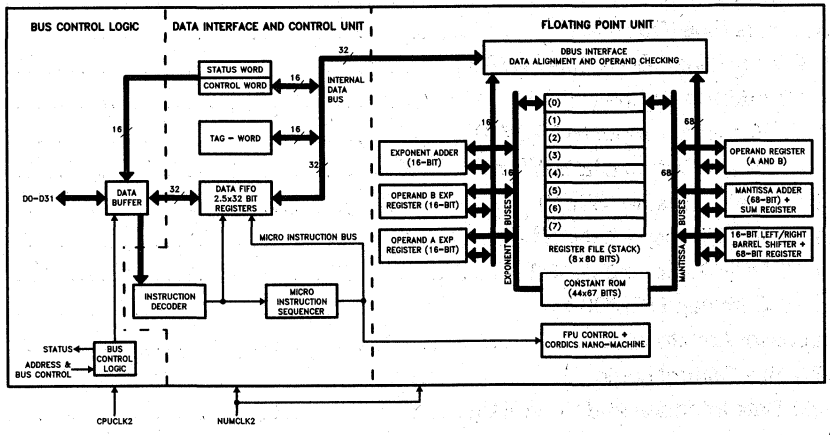
387™ DX MATH COPROCESSOR

- **High Performance 80-Bit Internal Architecture**
- **Implements ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic**
- **Six to Eleven Times 8087/80287 Performance**
- **Expands 386™ DX CPU Data Types to Include 32-, 64-, 80-Bit Floating Point, 32-, 64-Bit Integers and 18-Digit BCD Operands**
- **Directly Extends 386™ DX CPU Instruction Set to Include Trigonometric, Logarithmic, Exponential and Arithmetic Instructions for All Data Types**
- **Upward Object-Code Compatible from 8087 and 80287**
- **Full-Range Transcendental Operations for SINE, COSINE, TANGENT, ARCTANGENT and LOGARITHM**
- **Built-In Exception Handling**
- **Operates Independently of Real, Protected and Virtual-8086 Modes of the 386™ DX Microprocessor**
- **Eight 80-Bit Numeric Registers, Usable as Individually Addressable General Registers or as a Register Stack**
- **Available in 68-Pin PGA Package**
  (See Packaging Spec: Order #231369)

The Intel 387™ DX Math CoProcessor (MCP) is an extension to the Intel 386™ microprocessor architecture. The combination of the 387 DX with the 386™ DX Microprocessor dramatically increases the processing speed of computer application software which utilize mathmatical operations. This makes an ideal computer workstation platform for applications such as financial modeling and spreadsheets, CAD/CAM, or graphics.

The 387 DX Math CoProcessor adds over seventy mnemonics to the 386 DX Microprocessor instruction set. Specific 387 DX math operations include logarithmic, arithmetic, exponentional, and triginometric functions. The 387 DX supports integer, extended integer, floating point and BCD data formats, and fully conforms to the ANSI/IEEE floating point standard.

The 387 DX Math CoProcessor is object code compatible with the 80387SX, and upward object code compatible from the 80287 and 8087 math coprocessors. Object code for 386 DX/387 DX is also compatible with the Intel 486™ microprocessor. The 387 DX is manufactured on 1 micron, CHMOS IV technology and packaged in a 68-pin PGA package.



240448-1

Figure 0.1. 387™ DX Math Coprocessor Block Diagram

September 1990
Order Number: 240448-003

CONTENTS

PAGE

1.0 FUNCTIONAL DESCRIPTION 5-429

2.0 PROGRAMMING INTERFACE 5-430
2.1 Data Types 5-430
2.2 Numeric Operands 5-430
2.3 Register Set 5-432
2.3.1 Data Registers 5-432
2.3.2 Tag Word 5-432
2.3.3 Status Word 5-433
2.3.4 Instruction and Data Pointers 5-436
2.3.5 Control Word 5-438
2.4 Interrupt Description 5-438
2.5 Exception Handling 5-439
2.6 Initialization 5-439
2.7 8087 and 80287 Compatibility 5-440
2.7.1 General Differences 5-440
2.7.2 Exceptions 5-441

3.0 HARDWARE INTERFACE 5-441
3.1 Signal Description 5-441
3.1.1 386™ DX CPU Clock 2 (CPUCLK2) 5-444
3.1.2 387™ DX MCP Clock 2 (NUMCLK2) 5-444
3.1.3 387™ DX MCP Clocking Mode (CKM) 5-444
3.1.4 System Reset (RESETIN) 5-445
3.1.5 Processor Extension Request (PEREQ) 5-445
3.1.6 Busy Status (BUSY#) 5-445
3.1.7 Error Status (ERROR#) 5-445
3.1.8 Data Pins (D31–D0) 5-445
3.1.9 Write/Read Bus Cycle (W/R#) 5-445
3.1.10 Address Strobe (ADS#) 5-445
3.1.11 Bus Ready Input (READY#) 5-446
3.1.12 Ready Output (READYO#) 5-446
3.1.13 Status Enable (STEN) 5-446
3.1.14 MCP Select #1 (NPS1#) 5-446
3.1.15 MCP Select #2 (NPS2) 5-446
3.1.16 Command (CMD0#) 5-446
3.2 Processor Architecture 5-446
3.2.1 Bus Control Logic 5-447
3.2.2 Data Interface and Control Unit 5-447
3.2.3 Floating Point Unit 5-447

CONTENTS

PAGE

3.3 System Configuration 5-447
3.3.1 Bus Cycle Tracking 5-448
3.3.2 MCP Addressing 5-448
3.3.3 Function Select 5-448
3.3.4 CPU/MCP Synchronization 5-448
3.3.5 Synchronous or Asynchronous Modes 5-449
3.3.6 Automatic Bus Cycle Termination 5-449
3.4 Bus Operation 5-449
3.4.1 Nonpipelined Bus Cycles 5-450
3.4.1.1 Write Cycle 5-450
3.4.1.2 Read Cycle 5-450
3.4.2 Pipelined Bus Cycles 5-451
3.4.3 Bus Cycles of Mixed Type 5-452
3.4.4 BUSY# and PEREQ Timing Relationship 5-452

4.0 ELECTRICAL DATA 5-454
4.1 Absolute Maximum Ratings 5-454
4.2 DC Characteristics 5-454
4.3 AC Characteristics 5-455

5.0 387™ DX MCP EXTENSIONS TO THE 386™ DX CPU INSTRUCTION SET 5-460

APPENDIX A—COMPATIBILITY BETWEEN THE 80287 MCP AND THE 8087 5-464

5

FIGURES

PAGE

Figure 0.1 387™ DX Math Coprocessor Block Diagram 5-425
Figure 1.1 386™ DX Microprocessor and 387™ DX Math Coprocessor Register Set 5-429
Figure 2.1 387™ DX MCP Tag Word 5-432
Figure 2.2 MCP Status Word 5-433
Figure 2.3 Protected Mode 387™ DX MCP Instruction and Data Pointer Image in Memory, 32-Bit Format 5-436
Figure 2.4 Real Mode 387™ DX MCP Instruction and Data Pointer Image in Memory, 32-Bit Format 5-437
Figure 2.5 Protected Mode 387™ DX MCP Instruction and Data Pointer Image in Memory, 16-Bit Format 5-437
Figure 2.6 Real Mode 387™ DX MCP Instruction and Data Pointer Image in Memory, 16-Bit Format 5-437
Figure 2.7 387™ DX MCP Control Word 5-438
Figure 3.1 387™ DX MCP Pin Configuration 5-443
Figure 3.2 Asynchronous Operation 5-444
Figure 3.3 386™ DX Microprocessor and 387™ DX MCP Coprocessor System Configuration 5-447
Figure 3.4 Bus State Diagram 5-449
Figure 3.5 Nonpipelined Read and Write Cycles 5-451

FIGURES

PAGE

Figure 3.6 Fastest Transitions to and from Pipelined Cycles 5-452
Figure 3.7 Pipelined Cycles with Wait States 5-453
Figure 3.8 STEN, BUSY# and PEREQ Timing Relationship 5-453
Figure 4.0a Typical Output Valid Delay vs Load Capacitance at Max Operating Temperature 5-456
Figure 4.0b Typical Output Rise Time vs Load Capacitance at Max Operating Temperature . 5-456
Figure 4.1 CPUCLK2/NUMCLK2 Waveform and Measurement Points for Input/Output A.C. Specifications 5-457
Figure 4.2 Output Signals 5-457
Figure 4.3 Input and I/O Signals 5-458
Figure 4.4 RESET Signal 5-458
Figure 4.5 Float from STEN 5-458
Figure 4.6 Other Parameters 5-459

TABLES

PAGE

Table 2.1 387™ DX MCP Data Type Representation in Memory 5-431
Table 2.2 Condition Code Interpretation 5-434
Table 2.3 Condition Code Interpretation after FPREM and FPREM1 Instructions 5-435
Table 2.4 Condition Code Resulting from Comparison 5-435
Table 2.5 Condition Code Defining Operand Class 5-435
Table 2.6 386™ DX Microprocessor Interrupt Vectors Reserved for MCP 5-439
Table 2.7 Exceptions 5-440
Table 3.1 387™ DX MCP Pin Summary 5-442
Table 3.2 387™ DX MCP Pin Cross-Reference 5-442
Table 3.3 Output Pin Status after Reset 5-445
Table 3.4 Bus Cycles Definition 5-448
Table 4.1 DC Specifications 5-454
Table 4.2 Timing Requirements 5-455
Table 4.2a Combinations of Bus Interface and Execution Speeds 5-455
Table 4.2b Timing Requirements of the Execution Unit 5-455
Table 4.2c Timing Requirements of the Bus Interface Unit 5-455
Table 4.3 Other Parameters 5-459

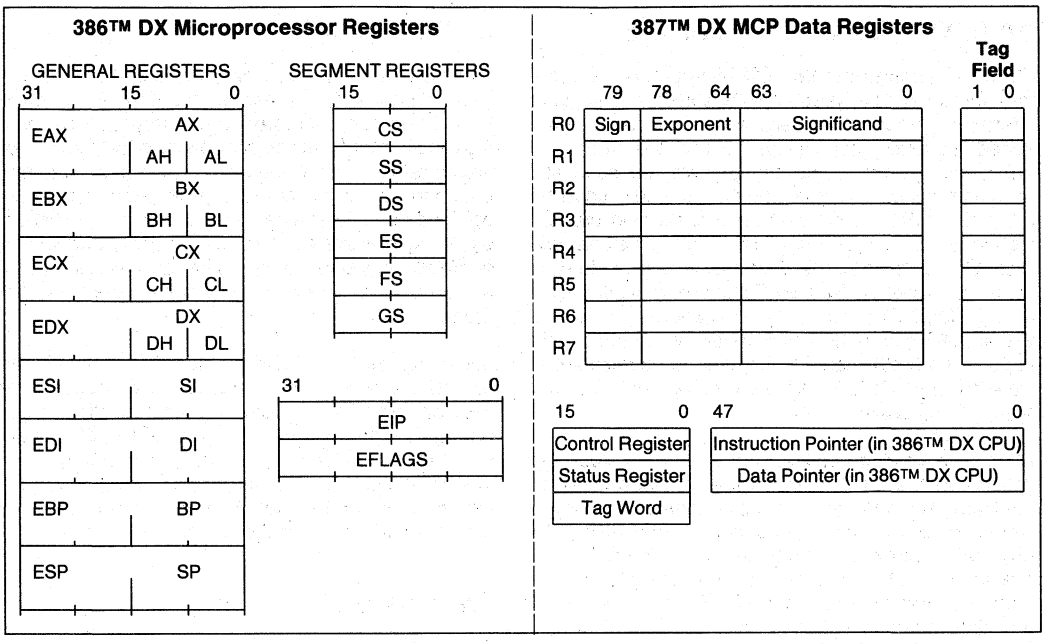


Figure 1.1. 386™ DX Microprocessor and 387™ DX Math Coprocessor Register Set

1.0 FUNCTIONAL DESCRIPTION

The 387™ DX Math Coprocessor provides arithmetic instructions for a variety of numeric data types in 386™ DX Microprocessor systems. It also executes numerous built-in transcendental functions (e.g. tangent, sine, cosine, and log functions). The 387 DX MCP effectively extends the register and instruction set of a 386 DX Microprocessor system for existing data types and adds several new data types as well. Figure 1.1 shows the model of registers visible to programs. Essentially, the 387 DX MCP can be treated as an additional resource or an extension to the 386 DX Microprocessor. The 386 DX Microprocessor together with a 387 DX MCP can be used as a single unified system.

The 387 DX MCP works the same whether the 386 DX Microprocessor is executing in real-address mode, protected mode, or virtual-8086 mode. All memory access is handled by the 386 DX Microprocessor; the 387 DX MCP merely operates on instructions and values passed to it by the 386 DX Microprocessor. Therefore, the 387 DX MCP is not sensitive to the processing mode of the 386 DX Microprocessor.

In real-address mode and virtual-8086 mode, the 386 DX Microprocessor and 387 DX MCP are completely upward compatible with software for 8086/8087, 80286/80287 real-address mode, and 386 DX Microprocessor and 80287 Coprocessor real-address mode systems.

In protected mode, the 386 DX Microprocessor and 387 DX MCP are completely upward compatible with software for 80286/80287 protected mode, and 386 DX Microprocessor and 80287 Coprocessor protected mode systems.

The only differences of operation that may appear when 8086/8087 programs are ported to a protected-mode 386 DX Microprocessor and 387 DX MCP system (*not* using virtual-8086 mode), is in the format of operands for the administrative instructions FLDENV, FSTENV, FRSTOR and FSAVE. These instructions are normally used only by exception handlers and operating systems, not by applications programs.

The 387 DX MCP contains three functional units that can operate in parallel to increase system performance. The 386 DX Microprocessor can be transferring commands and data to the MCP *bus control logic* for the next instruction while the MCP *floating-point unit* is performing the current numeric instruction.

2.0 PROGRAMMING INTERFACE

The MCP adds to the 386 DX Microprocessor system additional data types, registers, instructions, and interrupts specifically designed to facilitate high-speed numerics processing. To use the MCP requires no special programming tools, because all new instructions and data types are directly supported by the 386 DX CPU assembler and compilers for high-level languages. All 8086/8088 development tools that support the 8087 can also be used to develop software for the 386 DX Microprocessor and 387 DX Math Coprocessor in real-address mode or virtual-8086 mode. All 80286 development tools that support the 80287 can also be used to develop software for the 386 DX Microprocessor and 387 DX Math Coprocessor.

All communication between the 386 DX Microprocessor and the MCP is transparent to applications software. The CPU automatically controls the MCP whenever a numerics instruction is executed. All physical memory and virtual memory of the CPU are available for storage of the instructions and operands of programs that use the MCP. All memory addressing modes, including use of displacement, base register, index register, and scaling, are available for addressing numerics operands.

Section 6 at the end of this data sheet lists by class the instructions that the MCP adds to the instruction set of the 386 DX Microprocessor system.

2.1 Data Types

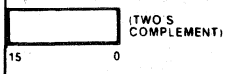
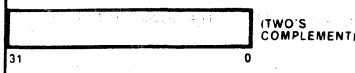
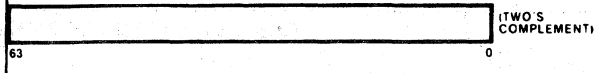
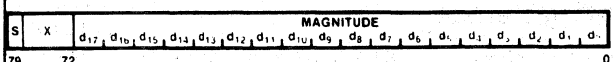
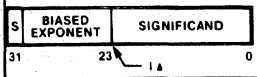
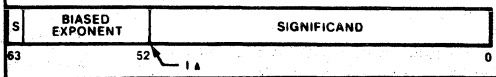
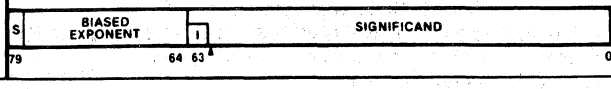
Table 2.1 lists the seven data types that the 387 DX MCP supports and presents the format for each type. Operands are stored in memory with the least significant digit at the lowest memory address. Programs retrieve these values by generating the lowest address. For maximum system performance, all operands should start at physical-memory addresses evenly divisible by four (doubleword boundaries); operands may begin at any other addresses, but will require extra memory cycles to access the entire operand.

Internally, the 387 DX MCP holds all numbers in the extended-precision real format. Instructions that load operands from memory automatically convert operands represented in memory as 16-, 32-, or 64-bit integers, 32- or 64-bit floating-point numbers, or 18-digit packed BCD numbers into extended-precision real format. Instructions that store operands in memory perform the inverse type conversion.

2.2 Numeric Operands

A typical MCP instruction accepts one or two operands and produces a single result. In two-operand instructions, one operand is the contents of an MCP register, while the other may be a memory location. The operands of some instructions are predefined; for example FSQRT always takes the square root of the number in the top stack element.

Table 2.1. 387™ DX MCP Data Type Representation in Memory

Data Formats	Range	Precision	Most Significant Byte = Highest Addressed Byte
Word Integer	$\pm 10^4$	16 Bits	
Short Integer	$\pm 10^9$	32 Bits	
Long Integer	$\pm 10^{18}$	64 Bits	
Packed BCD	$\pm 10^{\pm 18}$	18 Digits	
Single Precision	$\pm 10^{\pm 38}$	24 Bits	
Double Precision	$\pm 10^{\pm 308}$	53 Bits	
Extended Precision	$\pm 10^{\pm 4932}$	64 Bits	

240448-2

NOTES:
(1) S = Sign bit (0 = positive, 1 = negative)
(2) d_n = Decimal digit (two per byte)
(3) X = Bits have no significance; 387™ DX MCP ignores when loading, zeros when storing
(4)▲ = Position of implicit binary point
(5) I = Integer bit of significand; stored in temporary real, implicit in single and double precision
(6) Exponent Bias (normalized values):
Single: 127 (7FH)
Double: 1023 (3FFH)
Extended Real: 16383 (3FFFH)
(7) Packed BCD: $(-1)^S (D_{17}...D_0)$
(8) Real: $(-1)^S (2^{E-BIAS}) (F_0 F_1...)$

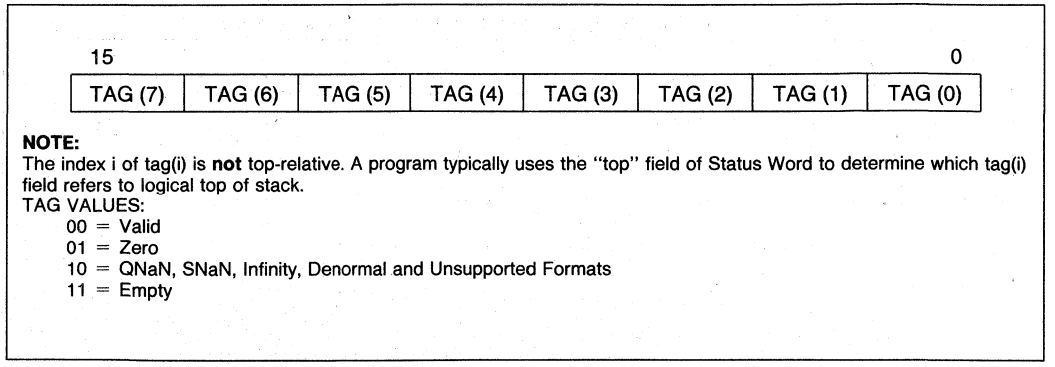


Figure 2.1. 387™ DX MCP Tag Word

2.3 Register Set

Figure 1.1 shows the 387 DX MCP register set. When an MCP is present in a system, programmers may use these registers in addition to the registers normally available on the 386 DX CPU.

2.3.1 DATA REGISTERS

387 DX MCP computations use the MCP's data registers. These eight 80-bit registers provide the equivalent capacity of twenty 32-bit registers. Each of the eight data registers in the MCP is 80 bits wide and is divided into "fields" corresponding to the MCPs extended-precision real data type.

The 387 DX MCP register set can be accessed either as a stack, with instructions operating on the top one or two stack elements, or as a fixed register set, with instructions operating on explicitly designated registers. The TOP field in the status word identifies the current top-of-stack register. A "push" operation decrements TOP by one and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments TOP by one. Like the 386 DX Microprocessor stacks in memory, the MCP register stack grows "down" toward lower-addressed registers.

Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the TOP of the stack. These instructions implicitly address the register at which TOP points. Other instructions allow the programmer to explicitly specify which register to user. This explicit register addressing is also relative to TOP.

2.3.2 TAG WORD

The tag word marks the content of each numeric data register, as Figure 2.1 shows. Each two-bit tag represents one of the eight numerics registers. The principal function of the tag word is to optimize the MCPs performance and stack handling by making it possible to distinguish between empty and nonempty register locations. It also enables exception handlers to check the contents of a stack location without the need to perform complex decoding of the actual data.

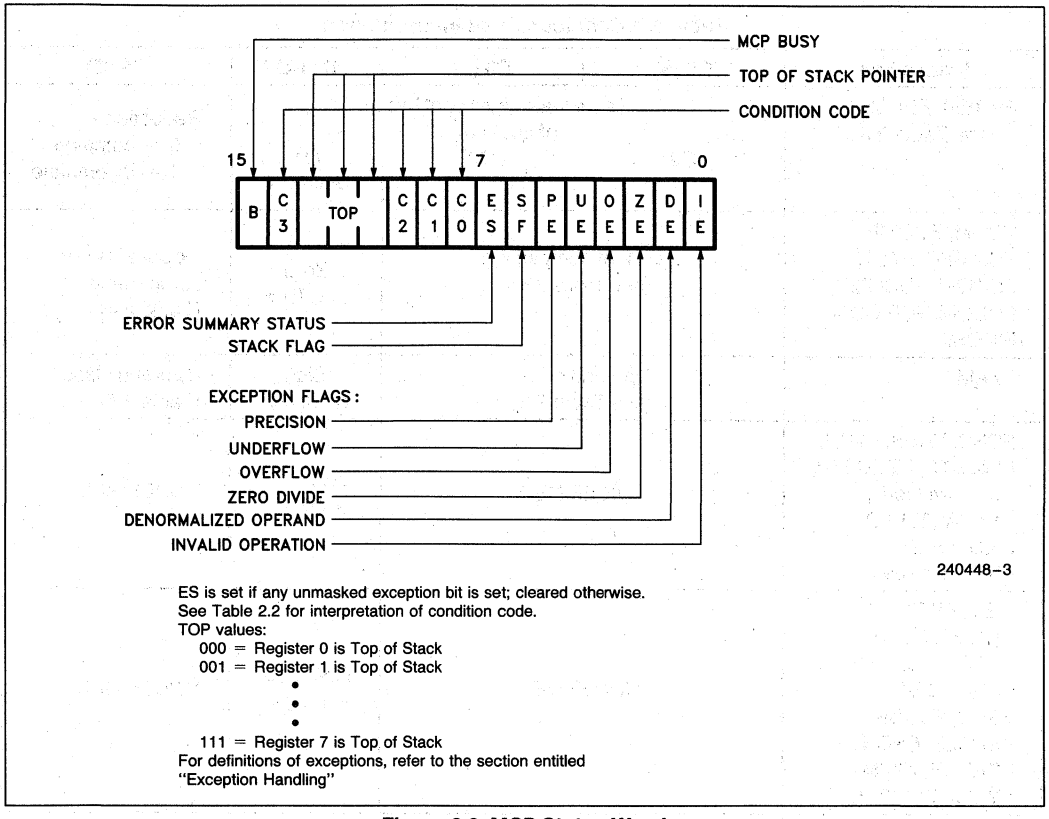


Figure 2.2. MCP Status Word

2.3.3 STATUS WORD

The 16-bit status word (in the status register) shown in Figure 2.2 reflects the overall state of the MCP. It may be read and inspected by CPU code.

Bit 15, the B-bit (busy bit) is included for 8087 compatibility only. It reflects the contents of the ES bit (bit 7 of the status word), not the status of the BUSY# output of the 387 DX MCP.

Bits 13–11 (TOP) point to the 387 DX MCP register that is the current top-of-stack.

The four numeric condition code bits (C_3–C_0) are similar to the flags in a CPU; instructions that perform arithmetic operations update these bits to reflect the outcome. The effects of these instructions on the condition code are summarized in Tables 2.2 through 2.5.

Bit 7 is the error summary (ES) status bit. This bit is set if any unmasked exception bit is set; it is clear otherwise. If this bit is set, the ERROR# signal is asserted.

Bit 6 is the stack flag (SF). This bit is used to distinguish invalid operations due to stack overflow or underflow from other kinds of invalid operations. When SF is set, bit 9 (C_1) distinguishes between stack overflow (C_1 = 1) and underflow (C_1 = 0).

Figure 2.2 shows the six exception flags in bits 5–0 of the status word. Bits 5–0 are set to indicate that the MCP has detected an exception while executing an instruction. A later section entitled "Exception Handling" explains how they are set and used.

Note that when a new value is loaded into the status word by the FLDENV or FRSTOR instruction, the value of ES (bit 7) and its reflection in the B-bit (bit 15) are not derived from the values loaded from memory but rather are dependent upon the values of the exception flags (bits 5–0) in the status word and their corresponding masks in the control word. If ES is set in such a case, the ERROR# output of the MCP is activated immediately.

Table 2.2. Condition Code Interpretation

Instruction	C0 (S)	C3 (Z)	C1 (A)	C2 (C)
FPREM, FPREM1 (see Table 2.3)	Three least significant bits of quotient Q2	Q0	Q1 or O/U#	Reduction 0 = complete 1 = incomplete
FCOM, FCOMP, FCOMPP, FTST, FUCOM, FUCOMP, FUCOMPP, FICOM, FICOMP	Result of comparison (see Table 2.4)		Zero or O/U#	Operand is not comparable (Table 2.4)
FXAM	Operand class (see Table 2.5)		Sign or O/U#	Operand class (Table 2.5)
FCHS, FABS, FXCH, FINCSTP, FDECSTP, Constant loads, FXTRACT, FLD, FILD, FBLD, FSTP (ext real)	UNDEFINED		Zero or O/U#	UNDEFINED
FIST, FBSTP, FRNDINT, FST, FSTP, FADD, FMUL, FDIV, FDIVR, FSUB, FSUBR, FSCALE, FSQRT, FPATAN, F2XM1, FYL2X, FYL2XP1	UNDEFINED		Roundup or O/U#	UNDEFINED
FPTAN, FSIN FCOS, FSINCOS	UNDEFINED		Roundup or O/U#, undefined if C2 = 1	Reduction 0 = complete 1 = incomplete
FLDENV, FRSTOR	Each bit loaded from memory			
FLDCW, FSTENV, FSTCW, FSTSW, FCLEX, FINIT, FSAVE	UNDEFINED			

O/U#	When both IE and SF bits of status word are set, indicating a stack exception, this bit distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).
Reduction	If FPREM or FPREM1 produces a remainder that is less than the modulus, reduction is complete. When reduction is incomplete the value at the top of the stack is a partial remainder, which can be used as input to further reduction. For FPTAN, FSIN, FCOS, and FSINCOS, the reduction bit is set if the operand at the top of the stack is too large. In this case the original operand remains at the top of the stack.
Roundup	When the PE bit of the status word is set, this bit indicates whether the last rounding in the instruction was upward.
UNDEFINED	Do not rely on finding any specific value in these bits.

Table 2.3. Condition Code Interpretation after FPREM and FPREM1 Instructions

Condition Code				Interpretation after FPREM and FPREM1	
C2	C3	C1	C0		
1	X	X	X	Incomplete Reduction: further interation required for complete reduction	
	Q1	Q0	Q2	Q MOD8	Complete Reduction: C0, C3, C1 contain three least significant bits of quotient
0	0	0	0	0	
	0	1	0	1	
	1	0	0	2	
	1	1	0	3	
	0	0	1	4	
	0	1	1	5	
	1	0	1	6	
	1	1	1	7	

Table 2.4. Condition Code Resulting from Comparison

Order	C3	C2	C0
TOP > Operand	0	0	0
TOP < Operand	0	0	1
TOP = Operand	1	0	0
Unordered	1	1	1

Table 2.5. Condition Code Defining Operand Class

C3	C2	C1	C0	Value at TOP
0	0	0	0	+ Unsupported
0	0	0	1	+ NaN
0	0	1	0	− Unsupported
0	0	1	1	− NaN
0	1	0	0	+ Normal
0	1	0	1	+ Infinity
0	1	1	0	− Normal
0	1	1	1	− Infinity
1	0	0	0	+ 0
1	0	0	1	+ Empty
1	0	1	0	− 0
1	0	1	1	− Empty
1	1	0	0	+ Denormal
1	1	1	0	− Denormal

2.3.4 INSTRUCTION AND DATA POINTERS

Because the MCP operates in parallel with the CPU, any errors detected by the MCP may be reported after the CPU has executed the ESC instruction which caused it. To allow identification of the failing numeric instruction, the 386 DX Microprocessor and 387 DX Math Coprocessor contains two pointer registers that supply the address of the failing numeric instruction and the address of its numeric memory operand (if appropriate).

The instruction and data pointers are provided for user-written error handlers. These registers are actually located in the 386 DX CPU, but appear to be located in the MCP because they are accessed by the ESC instructions FLDENV, FSTENV, FSAVE, and FRSTOR. (In the 8086/8087 and 80286/80287, these registers are located in the MCP.) Whenever the 386 DX CPU decodes a new ESC instruction, it saves the address of the instruction (including any prefixes that may be present), the address of the operand (if present), and the opcode.

The instruction and data pointers appear in one of four formats depending on the operating mode of the 386 DX Microprocessor (protected mode or real-address mode) and depending on the operand-size attribute in effect (32-bit operand or 16-bit operand). When the 386 DX Microprocessor is in virtual-8086 mode, the real-address mode formats are used. (See Figures 2.3 through 2.6.) The ESC instructions FLDENV, FSTENV, FSAVE, and FRSTOR are used to transfer these values between the 386 DX Microprocessor registers and memory. Note that the value of the data pointer is *undefined* if the prior ESC instruction did not have a memory operand.

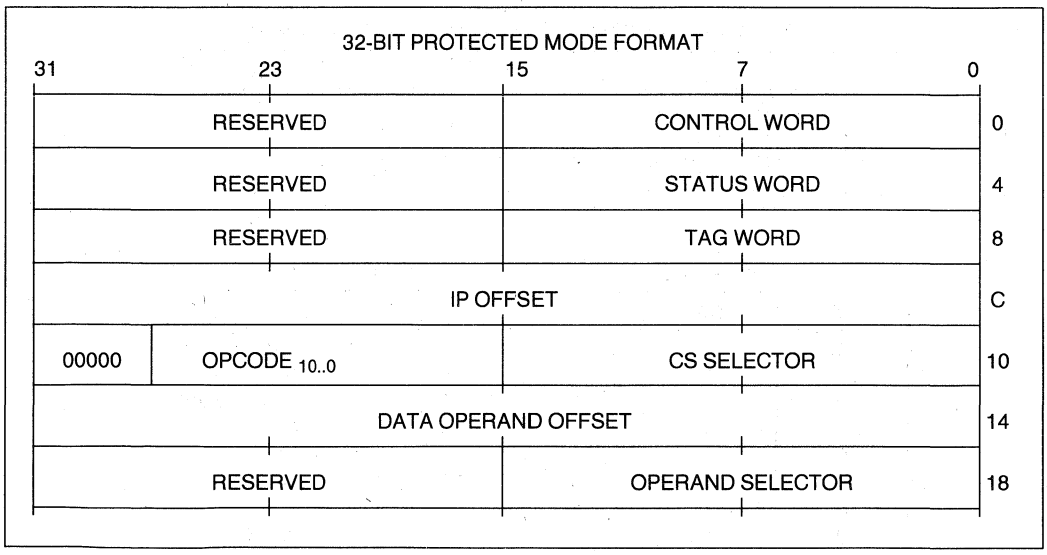


Figure 2.3. Protected Mode 387™ DX MCP Instruction and Data Pointer Image in Memory, 32-Bit Format

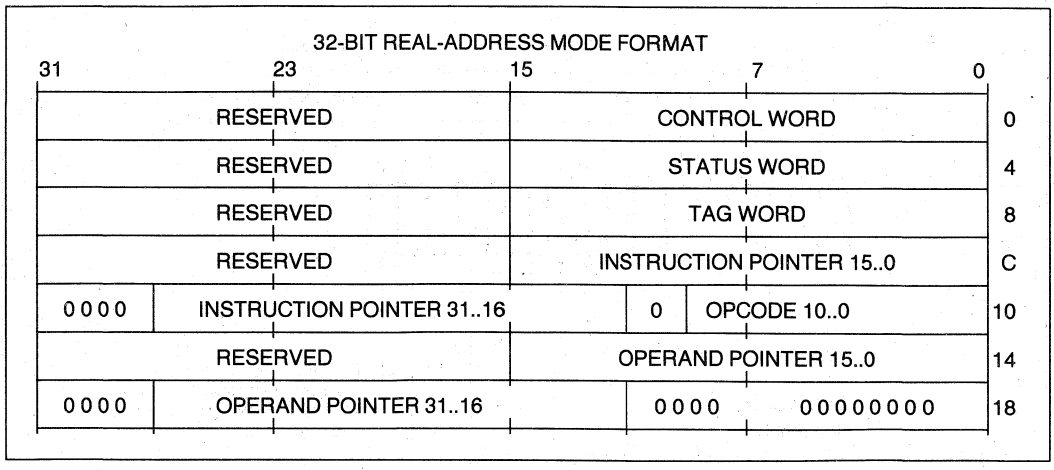


Figure 2.4. Real Mode 387™ DX MCP Instruction and Data Pointer Image in Memory, 32-Bit Format

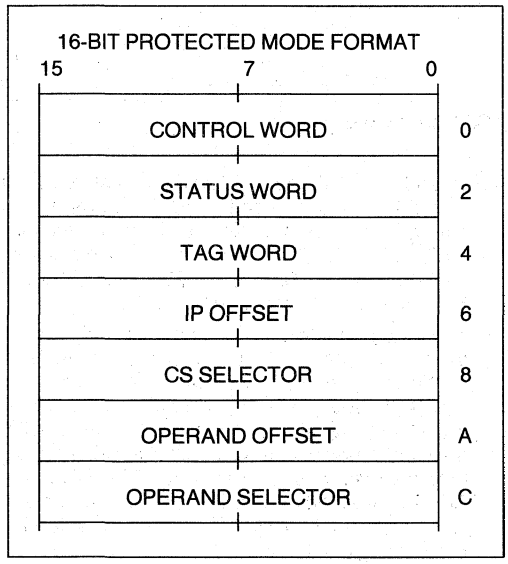


Figure 2.5. Protected Mode 387™ DX MCP Instruction and Data Pointer Image in Memory, 16-Bit Format

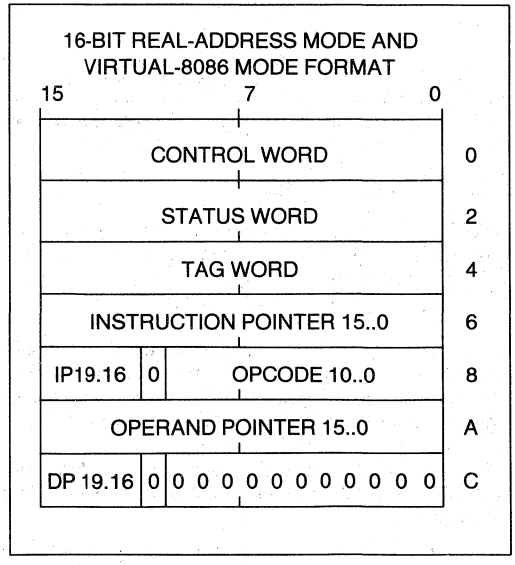


Figure 2.6. Real Mode 387™ DX MCP Instruction and Data Pointer Image in Memory, 16-Bit Format

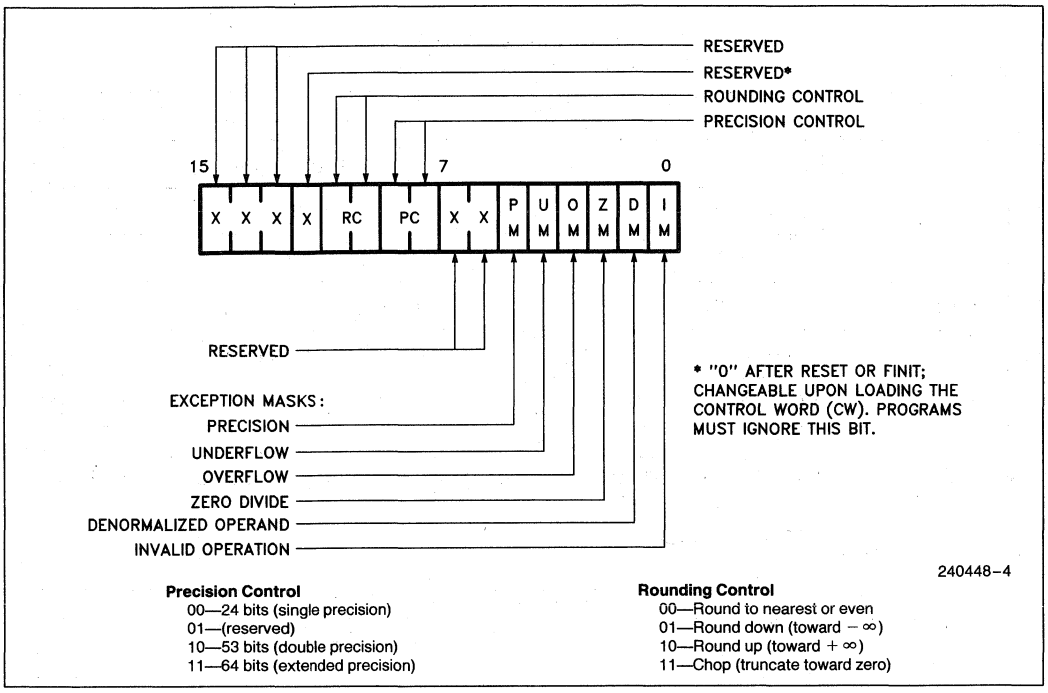


Figure 2.7. 387™ DX MCP Control Word

2.3.5 CONTROL WORD

The MCP provides several processing options that are selected by loading a control word from memory into the control register. Figure 2.7 shows the format and encoding of fields in the control word.

The low-order byte of this control word configures the MCP error and exception masking. Bits 5–0 of the control word contain individual masks for each of the six exceptions that the MCP recognizes.

The high-order byte of the control word configures the MCP operating mode, including precision and rounding.

- Bit 12 no longer defines infinity control and is a reserved bit. Only affine closure is supported for infinity arithmetic. The bit is initialized to zero after RESET or FINIT and is changeable upon loading the CW. Programs must ignore this bit.
- The rounding control (RC) bits (bits 11–10) provide for directed rounding and true chop, as well as the unbiased round to nearest even mode specified in the IEEE standard. Rounding control affects only those instructions that perform rounding at the end of the operation (and thus can generate a precision exception); namely, FST, FSTP, FIST, all arithmetic instructions (except FPREM, FPREM1, FXTRACT, FABS, and FCHS), and all transcendental instructions.
- The precision control (PC) bits (bits 9–8) can be used to set the MCP internal operating precision of the significand at less than the default of 64 bits (extended precision). This can be useful in providing compatibility with early generation arithmetic processors of smaller precision. PC affects only the instructions ADD, SUB, DIV, MUL, and SQRT. For all other instructions, either the precision is determined by the opcode or extended precision is used.

2.4 Interrupt Description

Several interrupts of the 386 DX CPU are used to report exceptional conditions while executing numeric programs in either real or protected mode. Table 2.6 shows these interrupts and their causes.

Table 2.6. 386™ DX Microprocessor Interrupt Vectors Reserved for MCP

Interrupt Number	Cause of Interrupt
7	An ESC instruction was encountered when EM or TS of the 386™ DX CPU control register zero (CR0) was set. EM = 1 indicates that software emulation of the instruction is required. When TS is set, either an ESC or WAIT instruction causes interrupt 7. This indicates that the current MCP context may not belong to the current task.
9	An operand of a coprocessor instruction wrapped around an addressing limit (0FFFFH for small segments, 0FFFFFFFFH for big segments, zero for expand-down segments) and spanned inaccessible addresses[a]. The failing numerics instruction is not restartable. The address of the failing numerics instruction and data operand may be lost; an FSTENV does not return reliable addresses. As with the 80286/80287, the segment overrun exception should be handled by executing an FNINIT instruction (i.e. an FINIT without a preceding WAIT). The return address on the stack does not necessarily point to the failing instruction nor to the following instruction. The interrupt can be avoided by never allowing numeric data to start within 108 bytes of the end of a segment.
13	The first word or doubleword of a numeric operand is not entirely within the limit of its segment. The return address pushed onto the stack of the exception handler points at the ESC instruction that caused the exception, including any prefixes. The 387™ DX MCP has not executed this instruction; the instruction pointer and data pointer register refer to a previous, correctly executed instruction.
16	The previous numerics instruction caused an unmasked exception. The address of the faulty instruction and the address of its operand are stored in the instruction pointer and data pointer registers. Only ESC and WAIT instructions can cause this interrupt. The 386™ DX CPU return address pushed onto the stack of the exception handler points to a WAIT or ESC instruction (including prefixes). This instruction can be restarted after clearing the exception condition in the MCP. FNINIT, FNCLEX, FNSTSW, FNSTENV, and FNSAVE cannot cause this interrupt.

a. An operand may wrap around an addressing limit when the segment limit is near an addressing limit and the operand is near the largest valid address in the segment. Because of the wrap-around, the beginning and ending addresses of such an operand will be at opposite ends of the segment. There are two ways that such an operand may also span inaccessible addresses: 1) if the segment limit is not equal to the addressing limit (e.g. addressing limit is FFFFH and segment limit is FFFDH) the operand will span addresses that are not within the segment (e.g. an 8-byte operand that starts at valid offset FFFC will span addresses FFFC–FFFF and 0000-0003; however addresses FFFE and FFFF are not valid, because they exceed the limit); 2) if the operand begins and ends in present and accessible pages but intermediate bytes of the operand fall in a not-present page or a page to which the procedure does not have access rights.

2.5 Exception Handling

The 387 DX MCP detects six different exception conditions that can occur during instruction execution. Table 2.7 lists the exception conditions in order of precedence, showing for each the cause and the default action taken by the MCP if the exception is masked by its corresponding mask bit in the control word.

Any exception that is not masked by the control word sets the corresponding exception flag of the status word, sets the ES bit of the status word, and asserts the ERROR# signal. When the CPU attempts to execute another ESC instruction or WAIT, exception 7 occurs. The exception condition must be resolved via an interrupt service routine. The 386 DX Microprocessor saves the address of the floating-point instruction that caused the exception and the address of any memory operand required by that instruction.

2.6 Initialization

387 DX MCP initialization software must execute an FNINIT instruction (i.e. an FINIT without a preceding WAIT) to clear ERROR#. After a hardware RESET, the ERROR# output is asserted to indicate that a 387 DX MCP is present. To accomplish this, the IE and ES bits of the status word are set, and the IM bit in the control word is reset. After FNINIT, the status word and the control word have the same values as in an 80287 after RESET.

2.7 8087 and 80287 Compatibility

This section summarizes the differences between the 387 DX MCP and the 80287. Any migration from the 8087 directly to the 387 DX MCP must also take into account the differences between the 8087 and the 80287 as listed in Appendix A.

Many changes have been designed into the 387 DX MCP to directly support the IEEE standard in hardware. These changes result in increased performance by eliminating the need for software that supports the standard.

2.7.1 GENERAL DIFFERENCES

The 387 DX MCP supports only affine closure for infinity arithmetic, not projective closure. Bit 12 of the Control Word (CW) no longer defines infinity control. It is a reserved bit; but it is initialized to zero after RESET or FINIT and is changeable upon loading the CW. Programs must ignore this bit.

Operands for FSCALE and FPATAN are no longer restricted in range (except for $\pm\infty$); F2XM1 and FPTAN accept a wider range of operands.

The results of transcendental operations may be slightly different from those computed by 80287.

In the case of FPTAN, the 387 DX MCP supplies a true tangent result in ST(1), and (always) a floating point 1 in ST.

Rounding control is in effect for FLD *constant.*

Software cannot change entries of the tag word to values (other than empty) that do not reflect the actual register contents.

After reset, FINIT, and incomplete FPREM, the 387 DX MCP resets to zero the condition code bits C_3–C_0 of the status word.

In conformance with the IEEE standard, the 387 DX MCP does not support the special data formats: pseudozero, pseudo-NaN, pseudoinfinity, and unnormal.

Table 2.7. Exceptions

Exception	Cause	Default Action (if exception is masked)
Invalid Operation	Operation on a signaling NaN, unsupported format, indeterminate form ($0*\infty$, 0/0, $(+\infty)+(-\infty)$, etc.), or stack overflow/underflow (SF is also set).	Result is a quiet NaN, integer indefinite, or BCD indefinite
Denormalized Operand	At least one of the operands is denormalized, i.e. it has the smallest exponent but a nonzero significand.	Normal processing continues
Zero Divisor	The divisor is zero while the dividend is a noninfinite, nonzero number.	Result is ∞
Overflow	The result is too large in magnitude to fit in the specified format.	Result is largest finite value or ∞
Underflow	The true result is nonzero but too small to be represented in the specified format, and, if underflow exception is masked, denormalization causes loss of accuracy.	Result is denormalized or zero
Inexact Result (Precision)	The true result is not exactly representable in the specified format (e.g. 1/3); the result is rounded according to the rounding mode.	Normal processing continues

2.7.2 EXCEPTIONS

A number of differences exist due to changes in the IEEE standard and to functional improvements to the architecture of the 387 DX MCP:

1. When the overflow or underflow exception is masked, the 387 DX MCP differs from the 80287 in rounding when overflow or underflow occurs. The 387 DX MCP produces results that are consistent with the rounding mode.
2. When the underflow exception is masked, the 387 DX MCP sets its underflow flag only if there is also a loss of accuracy during denormalization.
3. Fewer invalid-operation exceptions due to denormal operands, because the instructions FSQRT, FDIV, FPREM, and conversions to BCD or to integer normalize denormal operands before proceeding.
4. The FSQRT, FBSTP, and FPREM instructions may cause underflow, because they support denormal operands.
5. The denormal exception can occur during the transcendental instructions and the FXTRACT instruction.
6. The denormal exception no longer takes precedence over all other exceptions.
7. When the denormal exception is masked, the 387 DX MCP automatically normalizes denormal operands. The 8087/80287 performs unnormal arithmetic, which might produce an unnormal result.
8. When the operand is zero, the FXTRACT instruction reports a zero-divide exception and leaves $-\infty$ in ST(1).
9. The status word has a new bit (SF) that signals when invalid-operation exceptions are due to stack underflow or overflow.
10. FLD *extended precision* no longer reports denormal exceptions, because the instruction is not numeric.
11. FLD *single/double precision* when the operand is denormal converts the number to extended precision and signals the denormalized operand exception. When loading a signaling NaN, FLD *single/double precision* signals an invalid-operand exception.
12. The 387 DX MCP only generates quiet NaNs (as on the 80287); however, the 387 DX MCP distinguishes between quiet NaNs and signaling NaNs. Signaling NaNs trigger exceptions when they are used as operands; quiet NaNs do not (except for FCOM, FIST, and FBSTP which also raise IE for quiet NaNs).
13. When stack overflow occurs during FPTAN and overflow is masked, both ST(0) and ST(1) contain quiet NaNs. The 80287/8087 leaves the original operand in ST(1) intact.
14. When the scaling factor is $\pm\infty$, the FSCALE (ST(0), ST(1)) instruction behaves as follows (ST(0) and ST(1) contain the scaled and scaling operands respectively):
    - FSCALE(0, ∞) generates the invalid operation exception.
    - FSCALE(finite, $-\infty$) generates zero with the same sign as the scaled operand.
    - FSCALE(finite, $+\infty$) generates ∞ with the same sign as the scaled operand.

    The 8087/80287 returns zero in the first case and raises the invalid-operation exception in the other cases.
15. The 387 DX MCP returns signed infinity/zero as the unmasked response to massive overflow/underflow. The 8087 and 80287 support a limited range for the scaling factor; within this range either massive overflow/underflow do not occur or undefined results are produced.

3.0 HARDWARE INTERFACE

In the following description of hardware interface, the # symbol at the end of a signal name indicates that the active or asserted state occurs when the signal is at a low voltage. When no # is present after the signal name, the signal is asserted when at the high voltage level.

5

3.1 Signal Description

In the following signal descriptions, the 387 DX Math Coprocessor pins are grouped by function as follows:

1. Execution control—CPUCLK2, NUMCLK2, CKM, RESETIN
2. MCP handshake—PEREQ, BUSY#, ERROR#
3. Bus interface pins—D31–D0, W/R#, ADS#, READY#, READYO#
4. Chip/Port Select—STEN, NPS1#, NPS2, CMD0#
5. Power supplies—V_{CC}, V_{SS}

Table 3.1 lists every pin by its identifier, gives a brief description of its function, and lists some of its characteristics. All output signals are tristate; they leave floating state only when STEN is active. The output buffers of the bidirectional data pins D31–D0 are also tristate; they leave floating state only in read cycles when the MCP is selected (i.e. when STEN, NPS1#, and NPS2 are all active).

Figure 3.1 and Table 3.2 together show the location of every pin in the pin grid array.

Table 3.1. 387™ DX MCP Pin Summary

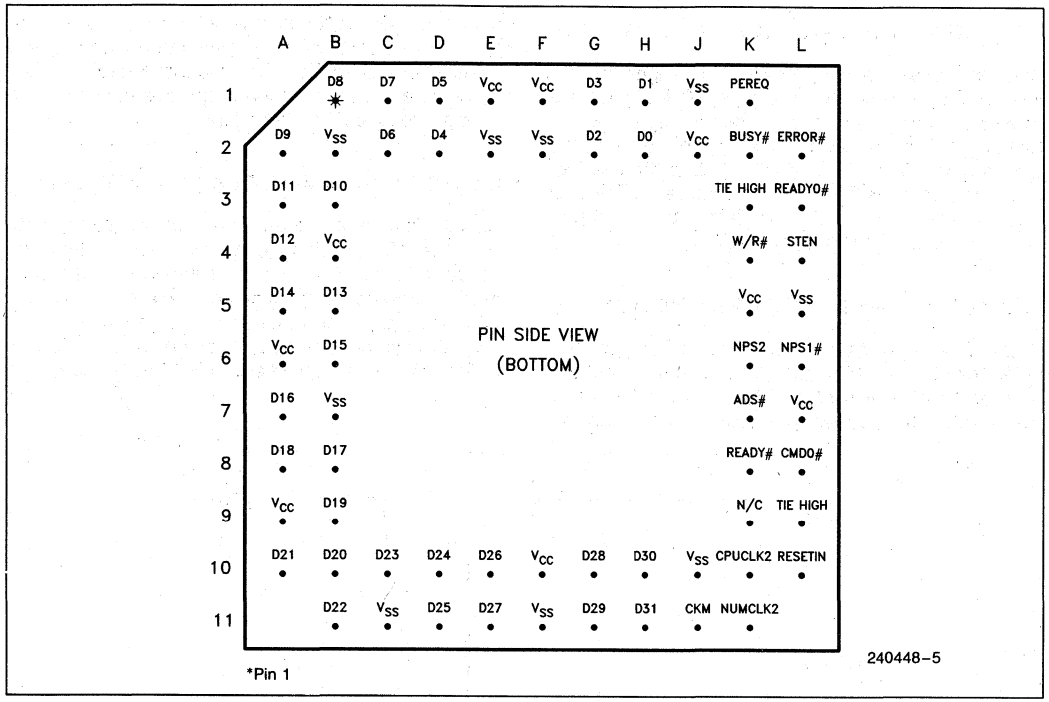
Pin Name	Function	Active State	Input/ Output	Referenced To
CPUCLK2	386™ DX CPU CLocK 2		I	
NUMCLK2	387™ DX MCP CLocK 2		I	
CKM	387™ DX MCP CLocKing Mode		I	
RESETIN	System reset	High	I	CPUCLK2
PEREQ	Processor Extension REQuest	High	O	CPUCLK2/STEN
BUSY#	Busy status	Low	O	CPUCLK2/STEN
ERROR#	Error status	Low	O	NUMCLK2/STEN
D31–D0	Data pins	High	I/O	CPUCLK2
W/R#	Write/Read bus cycle	Hi/Lo	I	CPUCLK2
ADS#	ADdress Strobe	Low	I	CPUCLK2
READY#	Bus ready input	Low	I	CPUCLK2
READYO#	Ready output	Low	O	CPUCLK2/STEN
STEN	STatus ENable	High	I	CPUCLK2
NPS1#	MCP select #1	Low	I	CPUCLK2
NPS2	MCP select #2	High	I	CPUCLK2
CMD0#	CoMmanD	Low	I	CPUCLK2
V_{CC}			I	
V_{SS}			I	

NOTE:
STEN is referenced to only when getting the output pins into or out of tristate mode.

Table 3.2. 387™ DX MCP Pin Cross-Reference

ADS#	—	K7	D18	—	A8	STEN	—	L4
BUSY#	—	K2	D19	—	B9	W/R#	—	K4
CKM	—	J11	D20	—	B10			
CPUCLK24	—	K10	D21	—	A10	V_{CC}	—	A6, A9, B4,
CMD0#	—	L8	D22	—	B11			E1, F1, F10,
D0	—	H2	D23	—	C10			J2, K5,
D1	—	H1	D24	—	D10			L7
D2	—	G2	D25	—	D11			
D3	—	G1	D26	—	E10	V_{SS}	—	B2, B7, C11,
D4	—	D2	D27	—	E11			E2, F2, F11,
D5	—	D1	D28	—	G10			J1, J10, L5
D6	—	C2	D29	—	G11			
D7	—	C1	D30	—	H10	NO CONNECT	—	K9
D8	—	B1	D31	—	H11	TIE HIGH	—	K3, L9*
D9	—	A2	ERROR#	—	L2			
D10	—	B3	NPS1#	—	L6			
D11	—	A3	NPS2	—	K6			
D12	—	A4	NUMCLK2	—	K11			
D13	—	B5	PEREQ	—	K1			
D14	—	A5	READY#	—	K8			
D15	—	B6	READYO#	—	L3			
D16	—	A7	RESETIN	—	L10			
D17	—	B8						

*Tie high pins may either be tied high with a pullup resistor or connected to V_{CC}.



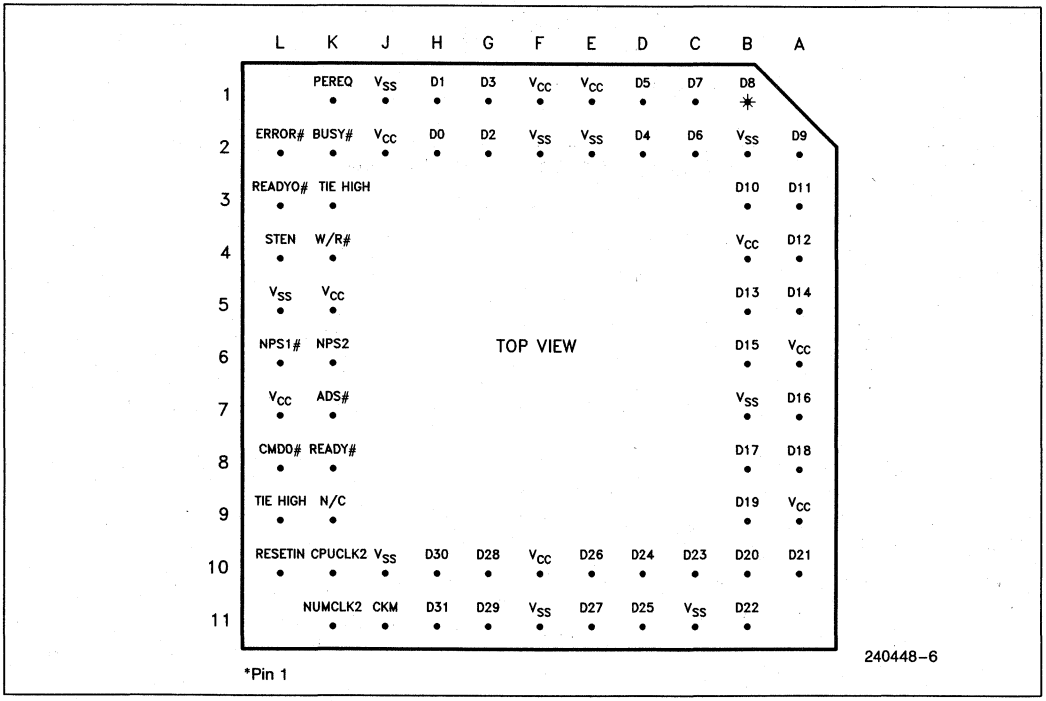


Figure 3.1. 387™ DX MCP Pin Configuration

3.1.1 386™ DX CPU CLOCK 2 (CPUCLK2)

This input uses the 386 DX CPU CLK2 signal to time the bus control logic. Several other MCP signals are referenced to the rising edge of this signal. When CKM = 1 (synchronous mode) this pin also clocks the data interface and control unit and the floating-point unit of the MCP. This pin requires MOS-level input. The signal on this pin is divided by two to produce the internal clock signal CLK.

3.1.2 387™ DX MCP CLOCK 2 (NUMCLK2)

When CKM = 0 (asynchronous mode) this pin provides the clock for the data interface and control unit and the floating-point unit of the MCP. In this case, the ratio of the frequency of NUMCLK2 to the frequency of CPUCLK2 must lie within the range 10:16 to 14:10. When CKM = 1 (synchronous mode) this pin is ignored; CPUCLK2 is used instead for the data interface and control unit and the floating-point unit. This pin requires TTL-level input.

3.1.3 387™ DX MCP CLOCKING MODE (CKM)

This pin is a strapping option. When it is strapped to V_{CC}, the MCP operates in synchronous mode; when strapped to V_{SS}, the MCP operates in asynchronous mode. These modes relate to clocking of the data interface and control unit and the floating-point unit only; the bus control logic always operates synchronously with respect to the 386 DX Microprocessor.

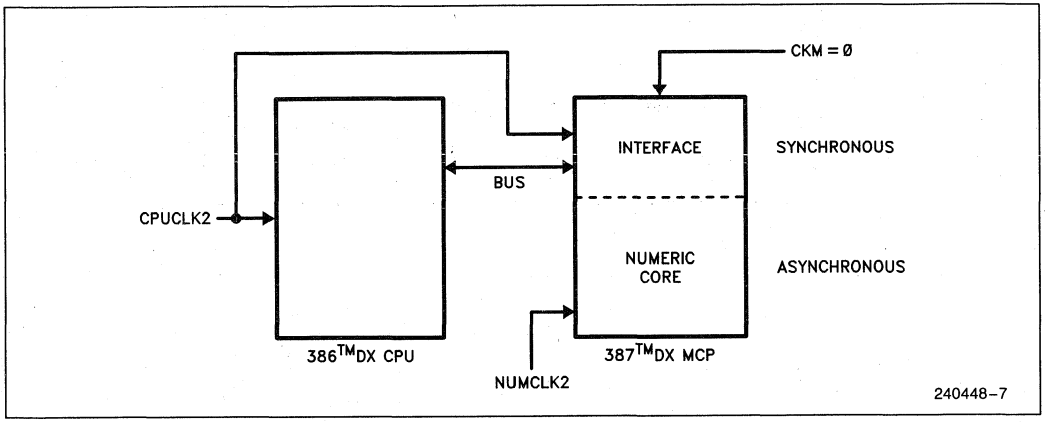


Figure 3.2. Asynchronous Operation

3.1.4 SYSTEM RESET (RESETIN)

A LOW to HIGH transition on this pin causes the MCP to terminate its present activity and to enter a dormant state. RESETIN must remain HIGH for at least 40 NUMCLK2 periods. The HIGH to LOW transitions of RESETIN must be synchronous with CPUCLK2, so that the phase of the internal clock of the bus control logic (which is the CPUCLK2 divided by 2) is the same as the phase of the internal clock of the 386 DX CPU. After RESETIN goes LOW, at least 50 NUMCLK2 periods must pass before the first MCP instruction is written into the 387 DX MCP. This pin should be connected to the 386 DX CPU RESET pin. Table 3.3 shows the status of other pins after a reset.

Table 3.3. Output Pin Status During Reset

Pin Value	Pin Name
HIGH	READYO#, BUSY#
LOW	PEREQ, ERROR#
Tri-State OFF	D31–D0

3.1.5 PROCESSOR EXTENSION REQUEST (PEREQ)

When active, this pin signals to the 386 DX CPU that the MCP is ready for data transfer to/from its data FIFO. When all data is written to or read from the data FIFO, PEREQ is deactivated. This signal always goes inactive before BUSY# goes inactive. This signal is referenced to CPUCLK2. It should be connected to the 386 DX CPU PEREQ input. Refer to Figure 3.8 for the timing relationships between this and the BUSY# and ERROR# pins.

3.1.6 BUSY STATUS (BUSY#)

When active, this pin signals to the 386 DX CPU that the MCP is currently executing an instruction. This signal is referenced to CPUCLK2. It should be connected to the 386 DX CPU BUSY# pin. Refer to Figure 3.8 for the timing relationships between this and the PEREQ and ERROR# pins.

3.1.7 ERROR STATUS (ERROR#)

This pin reflects the ES bits of the status register. When active, it indicates that an unmasked exception has occurred (except that, immediately after a reset, it indicates to the 386 DX Microprocessor that a 387 DX MCP is present in the system). This signal can be changed to inactive state only by the following instructions (without a preceding WAIT): FNINIT, FNCLEX, FNSTENV, and FNSAVE. This signal is referenced to NUMCLK2. It should be connected to the 386 DX CPU ERROR# pin. Refer to Figure 3.8 for the timing relationships between this and the PEREQ and BUSY# pins.

3.1.8 DATA PINS (D31–D0)

These bidirectional pins are used to transfer data and opcodes between the 386 DX CPU and 387 DX MCP. They are normally connected directly to the corresponding 386 DX CPU data pins. HIGH state indicates a value of one. D0 is the least significant data bit. Timings are referenced to CPUCLK2.

3.1.9 WRITE/READ BUS CYCLE (W/R#)

This signal indicates to the MCP whether the 386 DX CPU bus cycle in progress is a read or a write cycle. This pin should be connected directly to the 386 DX CPU W/R# pin. HIGH indicates a write cycle; LOW, a read cycle. This input is ignored if any of the signals STEN, NPS1#, or NPS2 is inactive. Setup and hold times are referenced to CPUCLK2.

3.1.10 ADDRESS STROBE (ADS#)

This input, in conjunction with the READY# input indicates when the MCP bus-control logic may sample W/R# and the chip-select signals. Setup and hold times are referenced to CPUCLK2. This pin should be connected to the 386 DX CPU ADS# pin.

3.1.11 BUS READY INPUT (READY#)

This input indicates to the MCP when a 386 DX CPU bus cycle is to be terminated. It is used by the bus-control logic to trace bus activities. Bus cycles can be extended indefinitely until terminated by READY#. This input should be connected to the same signal that drives the 386 DX CPU READY# input. Setup and hold times are referenced to CPUCLK2.

3.1.12 READY OUTPUT (READYO#)

This pin is activated at such a time that write cycles are terminated after two clocks (except FLDENV and FRSTOR) and read cycles after three clocks. In configurations where no extra wait states are required, this pin must directly or indirectly drive the 386 DX CPU READY# input. Refer to section 3.4 "Bus Operation" for details. This pin is activated only during bus cycles that select the MCP. This signal is referenced to CPUCLK2.

3.1.13 STATUS ENABLE (STEN)

This pin serves as a chip select for the MCP. When inactive, this pin forces BUSY#, PEREQ, ERROR#, and READYO# outputs into floating state. D31–D0 are normally floating and leave floating state only if STEN is active and additional conditions are met. STEN also causes the chip to recognize its other chip-select inputs. STEN makes it easier to do on-board testing (using the overdrive method) of other chips in systems containing the MCP. STEN should be pulled up with a resistor so that it can be pulled down when testing. In boards that do not use on-board testing, STEN should be connected to V_{CC}. Setup and hold times are relative to CPUCLK2. Note that STEN must maintain the same setup and hold times as NPS1#, NPS2, and CMD0# (i.e. if STEN changes state during a 387 DX MCP bus cycle, it should change state during the same CLK period as the NPS1#, NPS2, and CMD0# signals).

3.1.14 MCP Select #1 (NPS1#)

When active (along with STEN and NPS2) in the first period of a 386 DX CPU bus cycle, this signal indicates that the purpose of the bus cycle is to communicate with the MCP. This pin should be connected directly to the 386 DX CPU M/IO# pin, so that the MCP is selected only when the 386 DX CPU performs I/O cycles. Setup and hold times are referenced to CPUCLK2.

3.1.15 MCP SELECT #2 (NPS2)

When active (along with STEN and NPS1#) in the first period of an 386 DX CPU bus cycle, this signal indicates that the purpose of the bus cycle is to communicate with the MCP. This pin should be connected directly to the 386 DX CPU A31 pin, so that the MCP is selected only when the 386 DX CPU uses one of the I/O addresses reserved for the MCP (800000F8 or 800000FC). Setup and hold times are referenced to CPUCLK2.

3.1.16 COMMAND (CMD0#)

During a write cycle, this signal indicates whether an opcode (CMD0# active) or data (CMD0# inactive) is being sent to the MCP. During a read cycle, it indicates whether the control or status register (CMD0# active) or a data register (CMD0# inactive) is being read. CMD0# should be connected directly to the A2 output of the 386 DX Microprocessor. Setup and hold times are referenced to CPUCLK2.

3.2 Processor Architecture

As shown by the block diagram on the front page, the MCP is internally divided into three sections: the bus control logic (BCL), the data interface and control unit, and the floating point unit (FPU). The FPU (with the support of the control unit which contains the sequencer and other support units) executes all numerics instructions. The data interface and control unit is responsible for the data flow to and from the FPU and the control registers, for receiving the instructions, decoding them, and sequencing the microinstructions, and for handling some of the administrative instructions. The BCL is responsible for the 386 DX CPU bus tracking and interface. The BCL is the only unit in the 387 DX MCP that must run synchronously with the 386 DX CPU; the rest of the MCP can run asynchronously with respect to the 386 DX Microprocessor.

3.2.1 BUS CONTROL LOGIC

The BCL communicates solely with the CPU using I/O bus cycles. The BCL appears to the CPU as a special peripheral device. It is special in two respects: the CPU initiates I/O automatically when it encounters ESC instructions, and the CPU uses reserved I/O addresses to communicate with the BCL. The BCL does not communicate directly with memory. The CPU performs all memory access, transferring input operands from memory to the MCP and transferring outputs from the MCP to memory.

3.2.2 DATA INTERFACE AND CONTROL UNIT

The data interface and control unit latches the data and, subject to BCL control, directs the data to the FIFO or the instruction decoder. The instruction decoder decodes the ESC instructions sent to it by the CPU and generates controls that direct the data flow in the FIFO. It also triggers the microinstruction sequencer that controls execution of each instruction. If the ESC instruction is FINIT, FCLEX, FSTSW, FSTSW AX, or FSTCW, the control executes it independently of the FPU and the sequencer. The data interface and control unit is the one that generates the BUSY#, PEREQ and ERROR# signals that synchronize 387 DX MCP activities with the 386 DX CPU. It also supports the FPU in all operations that it cannot perform alone (e.g. exceptions handling, transcendental operations, etc.).

3.2.3 FLOATING POINT UNIT

The FPU executes all instructions that involve the register stack, including arithmetic, logical, transcendental, constant, and data transfer instructions. The data path in the FPU is 84 bits wide (68 significant bits, 15 exponent bits, and a sign bit) which allows internal operand transfers to be performed at very high speeds.

3.3 System Configuration

As an extension to the 386 DX Microprocessor, the 387 DX Math Coprocessor can be connected to the CPU as shown by Figure 3.3. A dedicated communi-

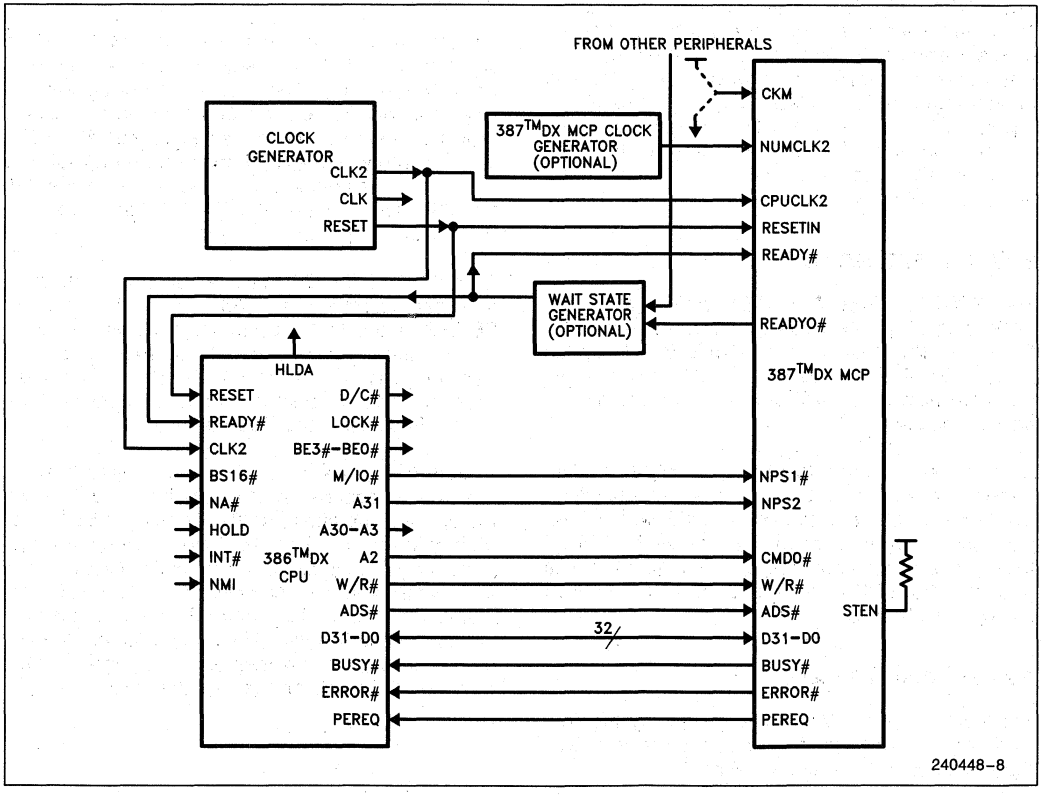


Figure 3.3. 386™ DX Microprocessor and 387™ DX Math Coprocessor System Configuration

Table 3.4. Bus Cycles Definition

STEN	NPS1#	NPS2	CMD0#	W/R#	Bus Cycle Type
0	x	x	x	x	MCP not selected and all outputs in floating state
1	1	x	x	x	MCP not selected
1	x	0	x	x	MCP not selected
1	0	1	0	0	CW or SW read from MCP
1	0	1	0	1	Opcode write to MCP
1	0	1	1	0	Data read from MCP
1	0	1	1	1	Data write to MCP

cation protocol makes possible high-speed transfer of opcodes and operands between the 386 DX CPU and 387 DX MCP. The 387 DX MCP is designed so that no additional components are required for interface with the 386 DX CPU. The 387 DX MCP shares the 32-bit wide local bus of the 386 DX CPU and most control pins of the 387 DX MCP are connected directly to pins of the 386 DX Microprocessor.

3.3.1 BUS CYCLE TRACKING

The ADS# and READY# signals allow the MCP to track the beginning and end of the 386 DX CPU bus cycles, respectively. When ADS# is asserted at the same time as the MCP chip-select inputs, the bus cycle is intended for the MCP. To signal the end of a bus cycle for the MCP, READY# may be asserted directly or indirectly by the MCP or by other bus-control logic. Refer to Table 3.4 for definition of the types of MCP bus cycles.

3.3.2 MCP ADDRESSING

The NPS1#, NPS2 and STEN signals allow the MCP to identify which bus cycles are intended for the MCP. The MCP responds only to I/O cycles when bit 31 of the I/O address is set. In other words, the MCP acts as an I/O device in a reserved I/O address space.

Because A_{31} is used to select the MCP for data transfers, it is not possible for a program running on the 386 DX CPU to address the MCP with an I/O instruction. Only ESC instructions cause the 386 DX Microprocessor to communicate with the MCP. The 386 DX CPU BS16# input must be inactive during I/O cycles when A_{31} is active.

3.3.3 FUNCTION SELECT

The CMD0# and W/R# signals identify the four kinds of bus cycle: control or status register read, data read, opcode write, data write.

3.3.4 CPU/MCP Synchronization

The pin pairs BUSY#, PEREQ, and ERROR# are used for various aspects of synchronization between the CPU and the MCP.

BUSY# is used to synchronize instruction transfer from the 386 DX CPU to the MCP. When the MCP recognizes an ESC instruction, it asserts BUSY#. For most ESC instructions, the 386 DX CPU waits for the MCP to deassert BUSY# before sending the new opcode.

The MCP uses the PEREQ pin of the 386 DX CPU to signal that the MCP is ready for data transfer to or from its data FIFO. The MCP does not directly access memory; rather, the 386 DX Microprocessor provides memory access services for the MCP. Thus, memory access on behalf of the MCP always obeys the rules applicable to the mode of the 386 DX CPU, whether the 386 DX CPU be in real-address mode or protected mode.

Once the 386 DX CPU initiates an MCP instruction that has operands, the 386 DX CPU waits for PEREQ signals that indicate when the MCP is ready for operand transfer. Once all operands have been transferred (or if the instruction has no operands) the 386 DX CPU continues program execution while the MCP executes the ESC instruction.

In 8086/8087 systems, WAIT instructions may be required to achieve synchronization of both commands and operands. In 80286/80287, 386 DX Microprocessor and 387 DX Math Coprocessor systems, WAIT instructions are required only for operand synchronization; namely, after MCP stores to memory (except FSTSW and FSTCW) or loads from memory. Used this way, WAIT ensures that the value has already been written or read by the MCP before the CPU reads or changes the value.

Once it has started to execute a numerics instruction and has transferred the operands from the 386 DX CPU, the MCP can process the instruction in parallel with and independent of the host CPU. When the MCP detects an exception, it asserts the ERROR# signal, which causes a 386 DX CPU interrupt.

3.3.5 SYNCHRONOUS OR ASYNCHRONOUS MODES

The internal logic of the 387 DX MCP (the FPU) can either operate directly from the CPU clock (synchronous mode) or from a separate clock (asynchronous mode). The two configurations are distinguished by the CKM pin. In either case, the bus control logic (BCL) of the MCP is synchronized with the CPU clock. Use of asynchronous mode allows the 386 DX CPU and the FPU section of the MCP to run at different speeds. In this case, the ratio of the frequency of NUMCLK2 to the frequency of CPUCLK2 must lie within the range 10:16 to 14:10. Use of synchronous mode eliminates one clock generator from the board design.

3.3.6 AUTOMATIC BUS CYCLE TERMINATION

In configurations where no extra wait states are required, READYO# can be used to drive the 386 DX CPU READY# input. If this pin is used, it should be connected to the logic that ORs all READY outputs from peripherals on the 386 DX CPU bus. READYO# is asserted by the MCP only during I/O cycles that select the MCP. Refer to section 3.4 "Bus Operation" for details.

3.4 Bus Operation

With respect to the bus interface, the 387 DX MCP is fully synchronous with the 386 DX Microprocessor. Both operate at the same rate, because each generates its internal CLK signal by dividing CPUCLK2 by two.

The 386 DX CPU initiates a new bus cycle by activating ADS#. The MCP recognizes a bus cycle, if, during the cycle in which ADS# is activated, STEN, NPS1#, and NPS2 are all activated. Proper operation is achieved if NPS1# is connected to the M/IO# output of the 386 DX CPU, and NPS2 to the A31 output. The 386 DX CPU's A31 output is guaranteed to be inactive in all bus cycles that do not address the MCP (i.e. I/O cycles to other devices, interrupt acknowledge, and reserved types of bus cycles). System logic must not signal a 16-bit bus cycle via the 386 DX CPU BS16# input during I/O cycles when A31 is active.

During the CLK period in which ADS# is activated, the MCP also examines the W/R# input signal to determine whether the cycle is a read or a write cycle and examines the CMD0# input to determine whether an opcode, operand, or control/status register transfer is to occur.

The 387 DX MCP supports both pipelined and nonpipelined bus cycles. A nonpipelined cycle is one for which the 386 DX CPU asserts ADS# when no other MCP bus cycle is in progress. A pipelined bus cycle is one for which the 386 DX CPU asserts ADS# and provides valid next-address and control signals as soon as in the second CLK period after the ADS# assertion for the previous 386 DX CPU bus cycle. Pipelining increases the availability of the bus by at least one CLK period. The MCP supports pipelined bus cycles in order to optimize address pipelining by the 386 DX CPU for memory cycles.

Bus operation is described in terms of an abstract *state machine*. Figure 3.4 illustrates the states and state transitions for MCP bus cycles:

- T_I is the idle state. This is the state of the bus logic after RESET, the state to which bus logic returns after evey nonpipelined bus cycle, and the state to which bus logic returns after a series of pipelined cycles.
- T_{RS} is the READY# sensitive state. Different types of bus cycle may require a minimum of one or two successive T_{RS} states. The bus logic remains in T_{RS} state until READY# is sensed, at which point the bus cycle terminates. Any number of wait states may be implemented by delaying READY#, thereby causing additional successive T_{RS} states.
- T_P is the first state for every pipelined bus cycle.

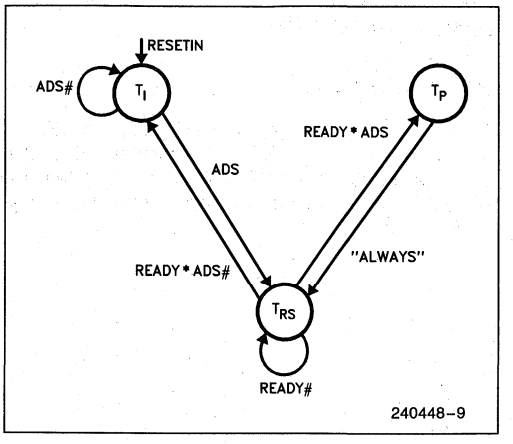


Figure 3.4. Bus State Diagram

The READYO# output of the 387 DX MCP indicates when a bus cycle for the MCP may be terminated if no extra wait states are required. For all write cycles (except those for the instructions FLDENV and FRSTOR), READYO# is always asserted in the first T_{RS} state, regardless of the number of wait states. For all read cycles and write cycles for FLDENV and FRSTOR, READYO# is always asserted in the second T_{RS} state, regardless of the number of wait states. These rules apply to both pipelined and nonpipelined cycles. Systems designers must use READYO# in one of the following ways:

1. Connect it (directly or through logic that ORs READY signals from other devices) to the READY# inputs of the 386 DX CPU and 387 DX MCP.
2. Use it as one input to a wait-state generator.

The following sections illustrate different types of MCP bus cycles.

Because different instructions have different amounts of overhead before, between, and after operand transfer cycles, it is not possible to represent in a few diagrams all of the combinations of successive operand transfer cycles. The following bus-cycle diagrams show memory cycles between MCP operand-transfer cycles. Note however that, during the instructions FLDENV, FSTENV, FSAVE, and FRSTOR, some consecutive accesses to the MCP do not have intervening memory accesses. For the timing relationship between operand transfer cycles and opcode write or other overhead activities, see Figure 3.8.

3.4.1 NONPIPELINED BUS CYCLES

Figure 3.5 illustrates bus activity for consecutive nonpipelined bus cycles.

3.4.1.1 Write Cycle

At the second clock of the bus cycle, the 387 DX MCP enters the T_{RS} (READY#-sensitive) state. During this state, the 387 DX MCP samples the READY# input and stays in this state as long as READY# is inactive.

In write cycles, the MCP drives the READYO# signal for one CLK period beginning with the second CLK of the bus cycle; therefore, the fastest write cycle takes two CLK cycles (see cycle 2 of Figure 3.5). For the instructions FLDENV and FRSTOR, however, the MCP forces a wait state by delaying the activation of READYO# to the second T_{RS} cycle (not shown in Figure 3.5).

When READY# is asserted the MCP returns to the idle state, in which ADS# could be asserted again by the 386 DX Microprocessor for the next cycle.

3.4.1.2 Read Cycle

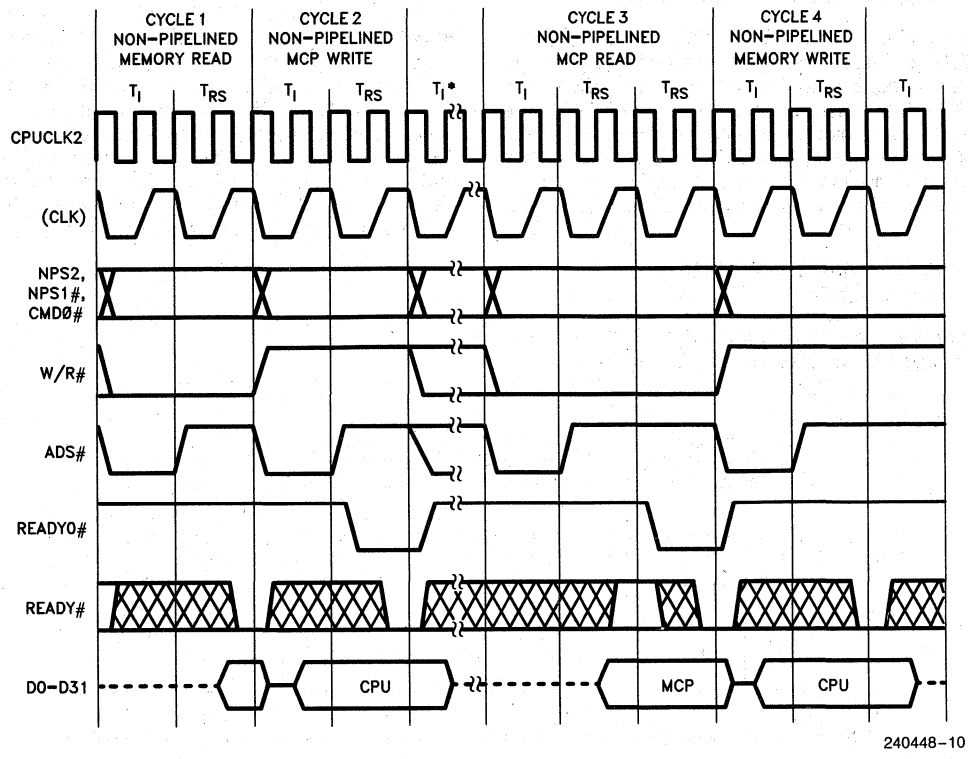
At the second clock of the bus cycle, the MCP enters the T_{RS} state. See Figure 3.5. In this state, the MCP samples the READY# input and stays in this state as long as READY# is inactive.

At the rising edge of CLK in the second clock period of the cycle, the MCP starts to drive the D31–D0 outputs and continues to drive them as long as it stays in T_{RS} state.

In read cycles that address the MCP, at least one wait state must be inserted to insure that the 386 DX CPU latches the correct data. Since the MCP starts driving the system data bus only at the rising edge of CLK in the second clock period of the bus cycle, not enough time is left for the data signals to propagate and be latched by the 386 DX CPU at the falling edge of the same clock period. The MCP drives the READYO# signal for one CLK period in the third CLK of the bus cycle. Therefore, if the READYO# output is used to drive the 386 DX CPU READY# input, one wait state is inserted automatically.

Because one wait state is required for MCP reads, the minimum is three CLK cycles per read, as cycle 3 of Figure 3.5 shows.

When READY# is asserted the MCP returns to the idle state, in which ADS# could be asserted again by the 386 DX CPU for the next cycle. The transition from T_{RS} state to idle state causes the MCP to put the tristate D31–D0 outputs into the floating state, allowing another device to drive the system data bus.



Cycles 1 & 2 represent part of the operand transfer cycle for instructions involving either 4-byte or 8-byte operand loads.
Cycles 3 & 4 represent part of the operand transfer cycle for a store operation.
*Cycles 1 & 2 could repeat here or T_I states for various non-operand transfer cycles and overhead.

Figure 3.5. Nonpipelined Read and Write Cycles

3.4.2 PIPELINED BUS CYCLES

Because all the activities of the 387 DX MCP bus interface occur either during the T_{RS} state or during the transitions to or from that state, the only difference between a pipelined and a nonpipelined cycle is the manner of changing from one state to another. The exact activities in each state are detailed in the previous section "Nonpipelined Bus Cycles".

When the 386 DX CPU asserts ADS# before the end of a bus cycle, both ADS# and READY# are active during a T_{RS} state. This condition causes the MCP to change to a different state named T_P. The MCP activities in the transition from a T_{RS} state to a T_P state are exactly the same as those in the transition from a T_{RS} state to a T_I state in nonpipelined cycles.

T_P state is metastable; therefore, one clock period later the MCP returns to T_{RS} state. In consecutive pipelined cycles, the MCP bus logic uses only T_{RS} and T_P states.

Figure 3.6 shows the fastest transition into and out of the pipelined bus cycles. Cycle 1 in this figure represents a nonpipelined cycle. (Nonpipelined write cycles with only one T_{RS} state (i.e. no wait states) are always followed by another nonpipelined cycle, because READY# is asserted before the earliest possible assertion of ADS# for the next cycle.)

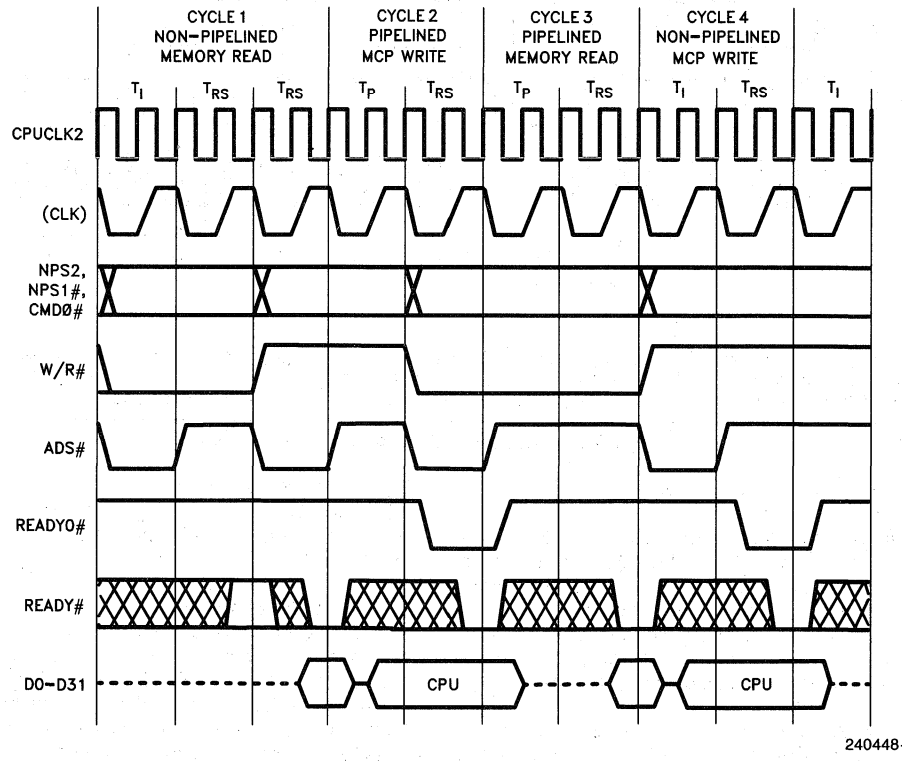
Figure 3.7 shows the pipelined write and read cycles with one additional T_{RS} states beyond the minimum required. To delay the assertion of READY# requires external logic.

3.4.3 BUS CYCLES OF MIXED TYPE

When the 387 DX MCP bus logic is in the T_{RS} state, it distinguishes between nonpipelined and pipelined cycles according to the behavior of ADS# and READY#. In a nonpipelined cycle, only READY# is activated, and the transition is from T_{RS} to idle state. In a pipelined cycle, both READY# and ADS# are active and the transition is first from T_{RS} state to T_P state then, after one clock period, back to T_{RS} state.

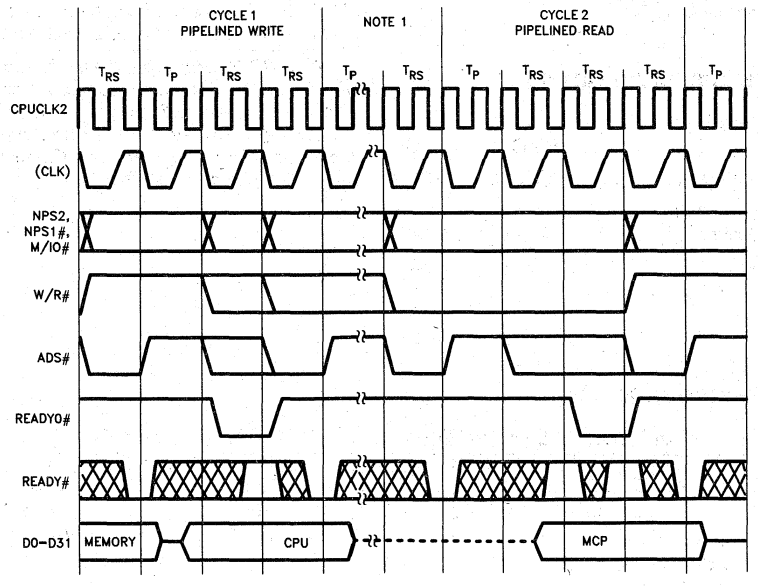
3.4.4 BUSY# AND PEREQ TIMING RELATIONSHIP

Figure 3.8 shows the activation of BUSY# at the beginning of instruction execution and its deactivation after execution of the instruction is complete. When possible, the 387 DX MCP may deactivate BUSY# prior to the completion of the current instruction allowing the CPU to transfer the next instruction's opcode and operands. PEREQ is activated in this interval. If ERROR# (not shown in the diagram) is ever asserted, it would occur at least six CPUCLK2 periods after the deactivation of PEREQ and at least six CPUCLK2 periods before the deactivation of BUSY#. Figure 3.8 shows also that STEN is activated at the beginning of a bus cycle.



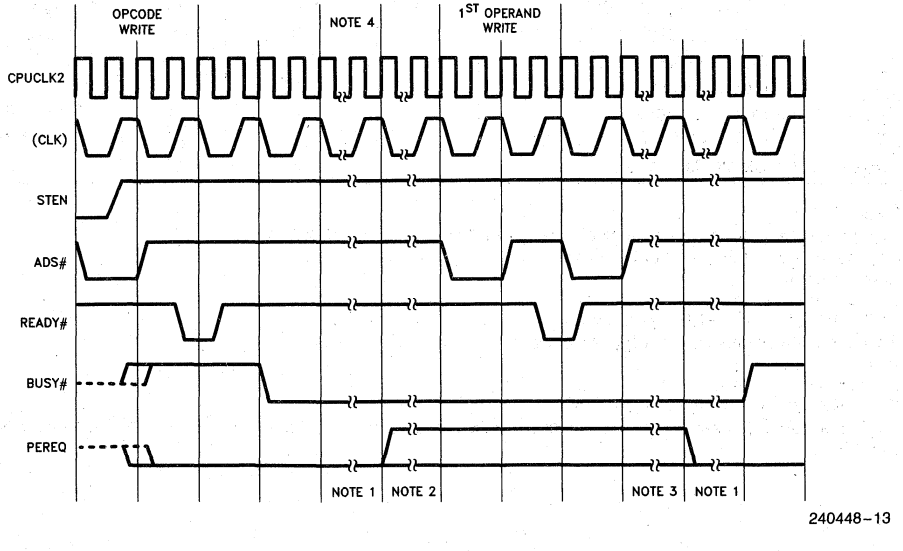
Cycle 1–Cycle 4 represent the operand transfer cycle for an instruction involving a transfer of two 32-bit loads in total. The opcode write cycles and other overhead are not shown.
Note that the next cycle will be a pipelined cycle if both READY# and ADS# are sampled active at the end of a T_{RS} state of the current cycle.

Figure 3.6. Fastest Transitions to and from Pipelined Cycles



NOTE:
1. Cycles between operand write to the MCP and storing result.

Figure 3.7. Pipelined Cycles with Wait States



NOTES:
1. Instruction dependent.
2. PEREQ is an asynchronous input to the 386™ DX Microprocessor; it may not be asserted (instruction dependent).
3. More operand transfers.
4. Memory read (operand) cycle is not shown.

Figure 3.8. STEN, BUSY# and PEREQ Timing Relationship

4.0 ELECTRICAL DATA

NOTICE: This is a production data sheet. The specifications are subject to change without notice.

4.1 Absolute Maximum Ratings*

Case Temperature T_C
Under Bias ..... −65°C to +110°C
Storage Temperature ..... −65°C to +150°C
Voltage on Any Pin with
Respect to Ground ..... −0.5 to V_{CC} +0.5V
Power Dissipation..... 1.5W

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

4.2 DC Characteristics

Table 4.1. DC Specifications T_C = 0° to 85°C, V_{CC} = 5V ± 5%

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input LO Voltage	−0.3	+0.8	V	(Note 1)
V_{IH}	Input HI Voltage	2.0	V_{CC} + 0.3	V	(Note 1)
V_{CL}	CPUCLK2 Input LO Voltage	−0.3	+0.8	V	
V_{CH}	CPUCLK2 Input Hi Voltage	3.7	V_{CC} +0.3	V	
V_{OL}	Output LO Voltage		0.45	V	(Note 2)
V_{OH}	Output HI Voltage	2.4		V	(Note 3)
I_{CC}	Supply Current				
	NUMCLK2 = 32 MHz(4)		160	mA	I_{CC} typ. = 95 mA
	NUMCLK2 = 40 MHz(4)		180	mA	I_{CC} typ. = 105 mA
	NUMCLK2 = 50 MHz(4)		210	mA	I_{CC} typ. = 125 mA
	NUMCLK2 = 66.6 MHz(4)		250	mA	I_{CC} typ. = 150 mA
I_{LI}	Input Leakage Current		±15	μA	$0V \le V_{IN} \le V_{CC}$
I_{LO}	I/O Leakage Current		±15	μA	$0.45V \le V_O \le V_{CC}$
C_{IN}	Input Capacitance		10	pF	fc = 1 MHz
C_O	I/O or Output Capacitance		12	pF	fc = 1 MHz
C_{CLK}	Clock Capacitance		15	pF	fc = 1 MHz

NOTES:
1. This parameter is for all inputs, including NUMCLK2 but excluding CPUCLK2.
2. This parameter is measured at I_{OL} as follows:
   data = 4.0 mA
   READYO# = 2.5 mA
   ERROR#, BUSY#, PEREQ = 2.5 mA
3. This parameter is measured at I_{OH} as follows:
   data = 1.0 mA
   READYO# = 0.6 mA
   ERROR#, BUSY#, PEREQ = 0.6 mA
4. I_{CC} is measured at steady state, maximum capacitive loading on the outputs, CPUCLK2 at the same frequency as NUMCLK2.

4.3 AC Characteristics

Table 4.2a. Combinations of Bus Interface and Execution Speeds

Functional Block	80387DX-16	80387DX-20	80387DX-25	80387DX-33
Bus Interface Unit (MHz)	16	20	25	33
Execution Unit (MHz)	16	20	25	33

Table 4.2b. Timing Requirements of the Execution Unit
T_C = 0°C to +85°C, V_{CC} = 5V ±5%

Pin	Symbol	Parameter	16 MHz		20 MHz		25 MHz		33 MHz		Test Conditions	Figure Reference
			Min (ns)	Max (ns)	Min (ns)	Max (ns)	Min (ns)	Max (ns)	Min (ns)	Max (ns)		
NUMCLK2	t1	Period	31.25	125	25	125	20	125	**15**	**125**	2.0V	4.1
NUMCLK2	t2a	High Time	9		8		7		**6.25**		2.0V	
NUMCLK2	t2b	High Time	5		5		4		**4.5**		3.7V	
NUMCLK2	t3a	Low Time	9		8		7		**6.25**		2.0V	
NUMCLK2	t3b	Low Time	7		6		5		**4.5**		0.8V	
NUMCLK2	t4	Fall Time		8		8		7		**6**	3.7V to 0.8V	
NUMCLK2	t5	Rise Time		8		8		7		**6**	0.8V to 3.7V	

Table 4.2c. Timing Requirements of the Bus Interface Unit
T_C = 0°C to +85°C, V_{CC} = 5V ±5%
(All measurements made at 1.5V and C_L = 50 pF unless otherwise specified)

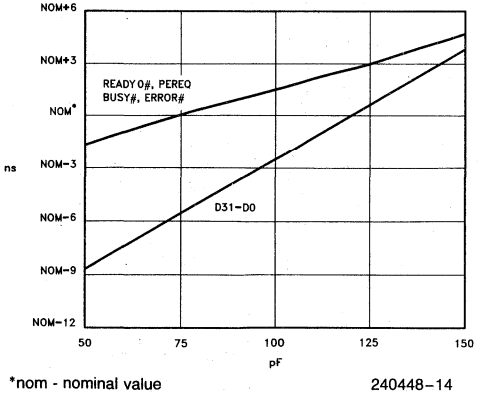
Pin	Symbol	Parameter	16 MHz		20 MHz		25 MHz		33 MHz		Test Conditions	Figure Reference
			Min (ns)	Max (ns)	Min (ns)	Max (ns)	Min (ns)	Max (ns)	Min (ns)	Max (ns)		
CPUCLK2	t1	Period	31.25	125	25	125	20	125	**15**	**125**	2.0V	4.1
CPUCLK2	t2a	High Time	9		8		7		**6.25**		2.0V	
CPUCLK2	t2b	High Time	5		5		4		**4.5**		3.7V	
CPUCLK2	t3a	Low Time	9		8		7		**6.25**		2.0V	
CPUCLK2	t3b	Low Time	7		6		5		**4.5**		0.8V	
CPUCLK2	t4	Fall Time		8		8		7		**4**	3.7V to 0.8V	
CPUCLK2	t5	Rise Time		8		8		7		**4**	0.8V to 3.7V	
CPUCLK2/ NUMCLK2		Ratio	10/16	14/10	10/16	14/10	10/16	14/10	**10/16**	**14/10**		
READYO#	t7	Out Delay	3	34	3	31	3	24	**3**	**17**	C_L = 75 pF†	4.2
READYO# (1)	t7	Out Delay	3	31	3	27	3	21	**3**	**15**	C_L = 25 pF	
PEREQ	t7	Out Delay	5	34	5	34	4	33	**4**	**25**	C_L = 75 pF†	
BUSY#	t7	Out Delay	5	34	5	29	4	29	**4**	**21**	C_L = 75 pF†	
BUSY# (1)	t7	Out Delay	N/A	N/A	N/A	N/A	4	27	**4**	**19**	C_L = 25 pF	
ERROR#	t7	Out Delay	5	34	5	34	4	33	**4**	**25**	C_L = 75 pF†	
D31-D0	t8	Out Delay	1	54	1	54	0	50	**0**	**37**	C_L = 120 pF†	4.3
D31-D0	t10	Setup Time	11		11		11		**8**			
D31-D0	t11	Hold Time	11		11		11		**8**			
D31-D0 (2)	t12*	Float Time	6	33	6	27	5	24	**3**	**19**	C_L = 120 pF†	
PEREQ (2)	t13*	Float Time	1	60	1	50	1	40	**1**	**30**	C_L = 75 pF†	4.5
BUSY# (2)	t13*	Float Time	1	60	1	50	1	40	**1**	**30**	C_L = 75 pF†	
ERROR# (2)	t13*	Float Time	1	60	1	50	1	40	**1**	**30**	C_L = 75 pF†	
READYO# (2)	t13*	Float Time	1	60	1	50	1	40	**1**	**30**	C_L = 75 pF†	

*Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not tested.
†For 25 MHz and 33 MHz, C_L = 50 pF

Table 4.2c. Timing Requirements of the Bus Interface Unit (Continued)
T_C = 0°C to +85°C, V_{CC} = 5V ±5%
(All measurements made at 1.5V and C_L = 50 pF unless otherwise specified)

Pin	Symbol	Parameter	16 MHz		20 MHz		25 MHz		33 MHz		Figure Reference
			Min (ns)	Max (ns)	Min (ns)	Max (ns)	Min (ns)	Max (ns)	Max (ns)	Min (ns)	
ADS#	t14	Setup Time	25		20		15		13		4.3
ADS#	t15	Hold Time	5		5		4		4		
W/R#	t14	Setup Time	25		20		15		13		
W/R#	t15	Hold Time	5		5		4		4		
READY#	t16	Setup Time	20		11		8		7		
READY#	t17	Hold Time	4		4		4		4		
CMD0#	t16	Setup Time	20		18		15		13		
CMD0#	t17	Hold Time	2		2		4		4		
NPS1# NPS2	t16	Setup Time	20		18		15		13		
NPS1# NPS2	t17	Hold Time	2		2		4		4		
STEN	t16	Setup Time	20		20		14		13		
STEN	t17	Hold Time	2		2		2		2		
RESETIN	t18	Setup Time	13		12		10		5		4.4
RESETIN	t19	Hold Time	4		4		3		3		

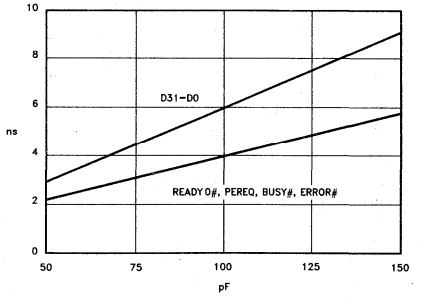
NOTES:
1. Not tested at 25 pF.
2. Float delay is not tested. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude.



*nom - nominal value 240448-14

NOTE:
This graph will not be linear outside of the C_L range shown.

Figure 4.0a. Typical Output Valid Delay vs Load Capacitance at Max Operating Temperature



240448-15

NOTE:
This graph will not be linear outside of the C_L range shown.

Figure 4.0b. Typical Output Rise Time vs Load Capacitance at Max Operating Temperature

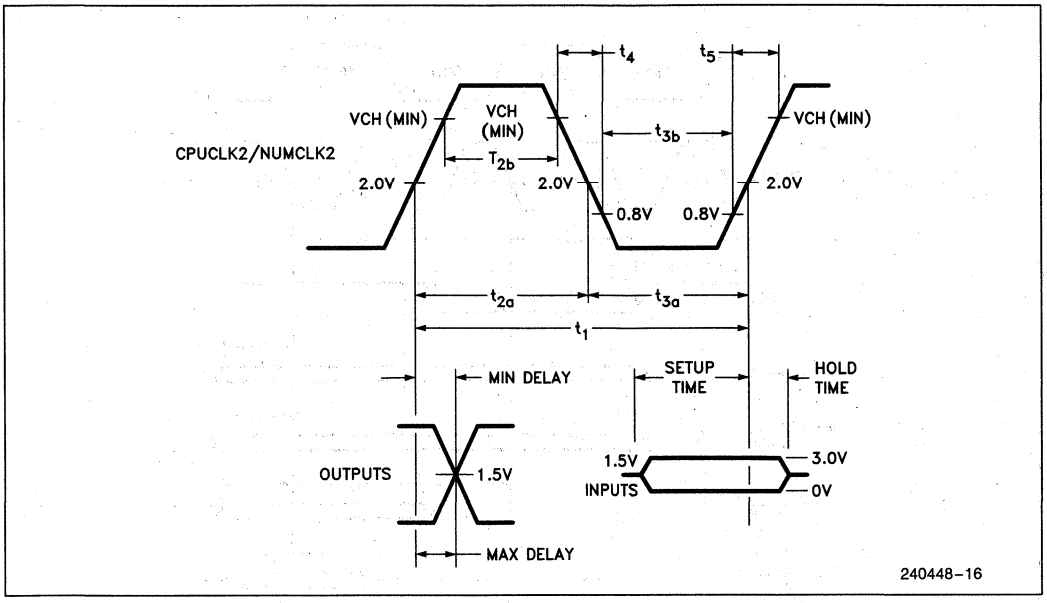


Figure 4.1. CPUCLK2/NUMCLK2 Waveform and Measurement Points for Input/Output A.C. Specifications

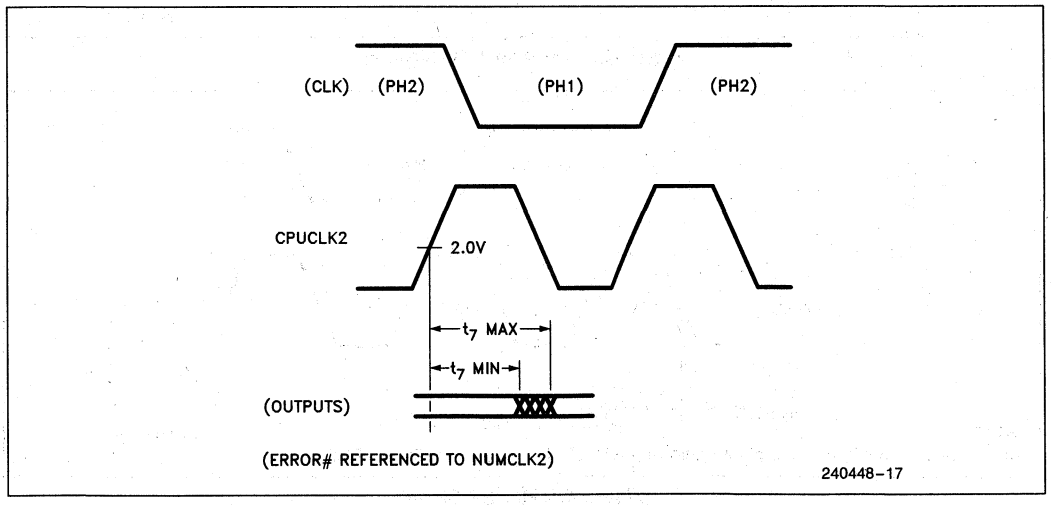


Figure 4.2. Output Signals

5

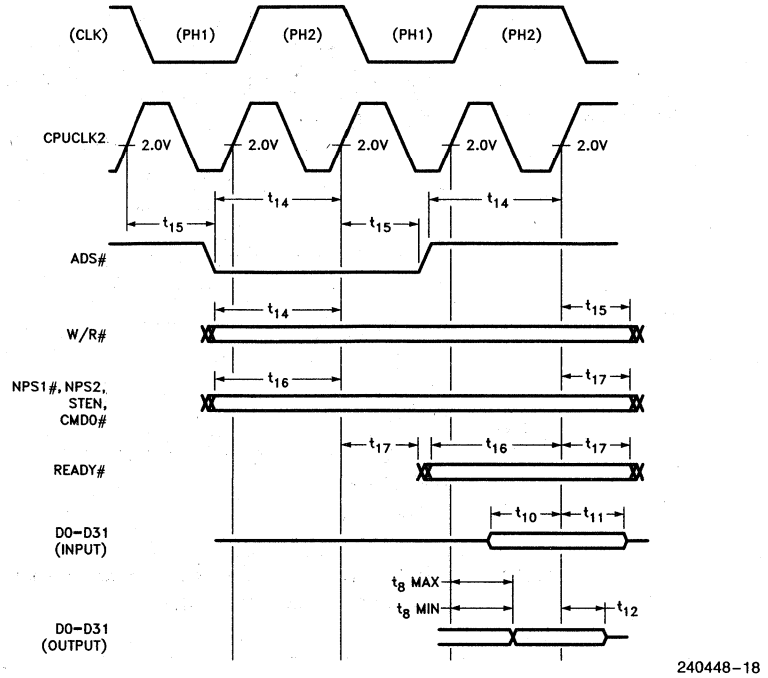
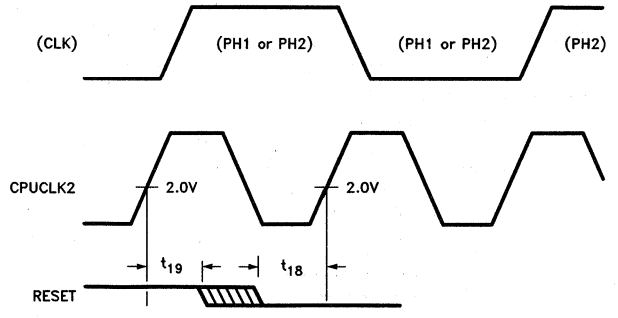


Figure 4.3. Input and I/O Signals



NOTE:
The second internal processor phase following RESET high to low transition is PH2.

Figure 4.4. RESET Signal

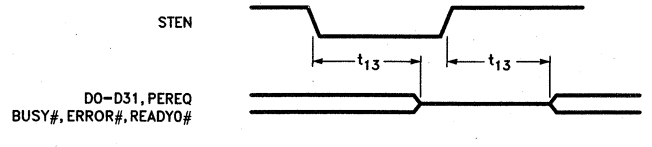
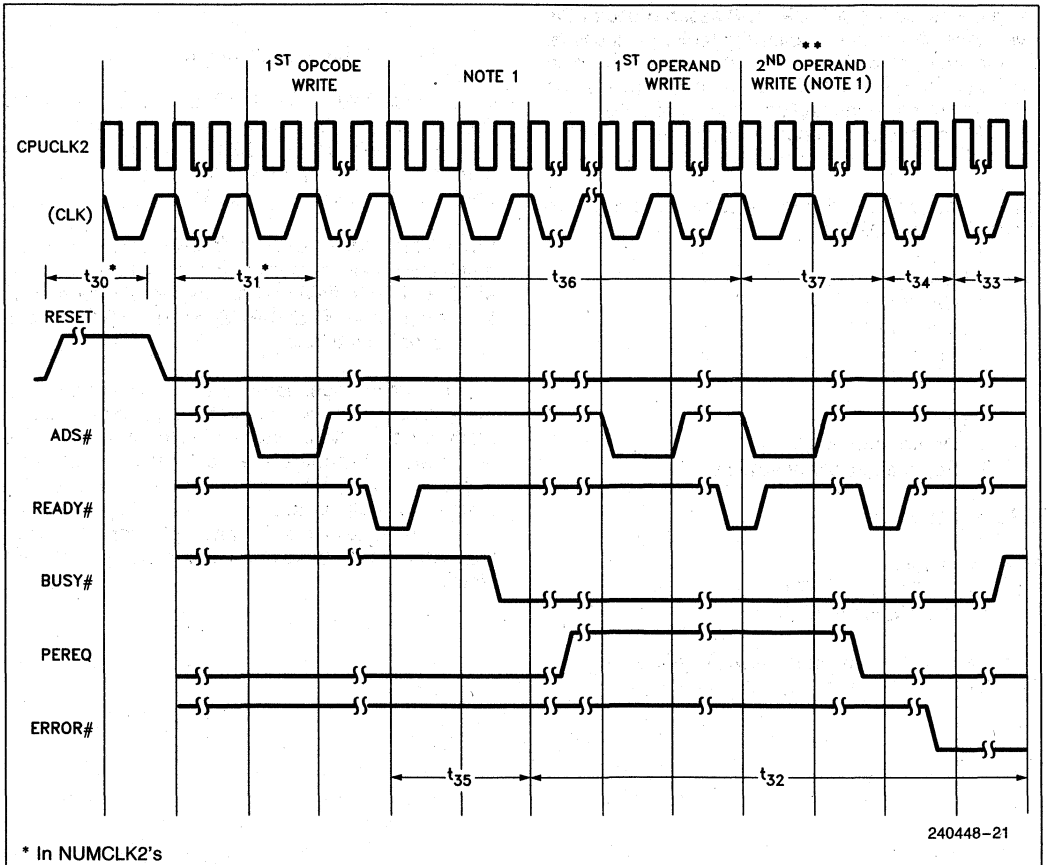


Figure 4.5. Float from STEN

Table 4.3. Other Parameters

Pin	Symbol	Parameter	Min	Max	Units
RESETIN	t30	Duration	40		NUMCLK2
RESETIN	t31	RESETIN Inactive to 1st Opcode Write	50		NUMCLK2
BUSY#	t32	Duration	6		CPUCLK2
BUSY#, ERROR#	t33	ERROR# (In) Active to BUSY# Inactive	6		CPUCLK2
PEREQ, ERROR#	t34	PEREQ Inactive to ERROR# Active	6		CPUCLK2
READY#, BUSY#	t35	READY# Active to BUSY# Active	4	4	CPUCLK2
READY#	t36	Minimum Time from Opcode Write to Opcode/Operand Write	6		CPUCLK2
READY#	t37	Minimum Time from Operand Write to Operand Write	8		CPUCLK2



* In NUMCLK2's
** or last operand

NOTE:
1. Memory read (operand) cycle is not shown.

Figure 4.6. Other Parameters

	Instruction								Optional Fields	
	First Byte				Second Byte					
1	11011	OPA		1	MOD	1	OPB	R/M	SIB	DISP
2	11011	MF		OPA	MOD		OPB	R/M	SIB	DISP
3	11011	d	P	OPA	1	1	OPB	ST(i)		
4	11011	0	0	1	1	1	1	OP		
5	11011	0	1	1	1	1	1	OP		
	15–11	10	9	8	7	6	5	4 3 2 1 0		

5.0 387™ DX MCP EXTENSIONS TO THE 386™ DX CPU INSTRUCTION SET

Instructions for the 387 DX MCP assume one of the five forms shown in the following table. In all cases, instructions are at least two bytes long and begin with the bit pattern 11011B, which identifies the ESCAPE class of instruction. Instructions that refer to memory operands specify addresses using the 386 DX CPU addressing modes.

OP = Instruction opcode, possible split into two fields OPA and OPB

MF = Memory Format
- 00—32-bit real
- 01—32-bit integer
- 10—64-bit real
- 11—16-bit integer

P = Pop
- 0—Do not pop stack
- 1—Pop stack after operation

ESC = 11011

d = Destination
- 0—Destination is ST(0)
- 1—Destination is ST(i)

R XOR d = 0—Destination (op) Source
R XOR d = 1—Source (op) Destination

ST(i) = Register stack element *i*
- 000 = Stack top
- 001 = Second stack element
- •
- •
- •
- 111 = Eighth stack element

MOD (Mode field) and R/M (Register/Memory specifier) have the same interpretation as the corresponding fields of the 386 DX Microprocessor instructions (refer to *386™ DX Microprocessor Programmer's Reference Manual*).

SIB (Scale Index Base) byte and DISP (displacement) are optionally present in instructions that have MOD and R/M fields. Their presence depends on the values of MOD and R/M, as for 386 DX Microprocessor instructions.

The instruction summaries that follow assume that the instruction has been prefetched, decoded, and is ready for execution; that bus cycles do not require wait states; that there are no local bus HOLD request delaying processor access to the bus; and that no exceptions are detected during instruction execution. If the instruction has MOD and R/M fields that call for both base and index registers, add one clock.

387™ DX MCP Extensions to the 386™ DX CPU Instruction Set

Instruction	Encoding			Clock Count Range			
	Byte 0	Byte 1	Optional Bytes 2–6	32-Bit Real	32-Bit Integer	64-Bit Real	16-Bit Integer
DATA TRANSFER							
FLD = Load[a]							
Integer/real memory to ST(0)	ESC MF 1	MOD 000 R/M	SIB/DISP	9–18	26–42	16–23	42–53
Long integer memory to ST(0)	ESC 111	MOD 101 R/M	SIB/DISP		26–54		
Extended real memory to ST(0)	ESC 011	MOD 101 R/M	SIB/DISP		12–43		
BCD memory to ST(0)	ESC 111	MOD 100 R/M	SIB/DISP		45–97		
ST(i) to ST(0)	ESC 001	11000 ST(i)			7–12		
FST = Store							
ST(0) to integer/real memory	ESC MF 1	MOD 010 R/M	SIB/DISP	25–43	57–76	32–44	58–76
ST(0) to ST(i)	ESC 101	11010 ST(i)			7–11		
FSTP = Store and Pop							
ST(0) to integer/real memory	ESC MF 1	MOD 011 R/M	SIB/DISP	25–43	57–76	32–44	58–76
ST(0) to long integer memory	ESC 111	MOD 111 R/M	SIB/DISP		60–82		
ST(0) to extended real	ESC 011	MOD 111 R/M	SIB/DISP		46–52		
ST(0) to BCD memory	ESC 111	MOD 110 R/M	SIB/DISP		112–190		
ST(0) to ST(i)	ESC 101	11011 ST (i)			7–11		
FXCH = Exchange							
ST(i) and ST(0)	ESC 001	11001 ST(i)			10–17		
COMPARISON							
FCOM = Compare							
Integer/real memory to ST(0)	ESC MF 0	MOD 010 R/M	SIB/DISP	13–25	34–52	14–27	39–62
ST(i) to ST(0)	ESC 000	11010 ST(i)			13–21		
FCOMP = Compare and pop							
Integer/real memory to ST	ESC MF 0	MOD 011 R/M	SIB/DISP	13–25	34–52	14–27	39–62
ST(i) to ST(0)	ESC 000	11011 ST(i)			13–21		
FCOMPP = Compare and pop twice							
ST(1) to ST(0)	ESC 110	1101 1001			13–21		
FTST = Test ST(0)	ESC 001	1110 0100			17–25		
FUCOM = Unordered compare	ESC 101	11100 ST(i)			13–21		
FUCOMP = Unordered compare and pop	ESC 101	11101 ST(i)			13–21		
FUCOMPP = Unordered compare and pop twice	ESC 010	1110 1001			13–21		
FXAM = Examine ST(0)	ESC 001	11100101			24–37		
CONSTANTS							
FLDZ = Load +0.0 into ST(0)	ESC 001	1110 1110			10–17		
FLD1 = Load +1.0 into ST(0)	ESC 001	1110 1000			15–22		
FLDPI = Load pi into ST(0)	ESC 001	1110 1011			26–36		
FLDL2T = Load $\log_2(10)$ into ST(0)	ESC 001	1110 1001			26–36		

Shaded areas indicate instructions not available in 8087/80287.

NOTE:
a. When loading single- or double-precision zero from memory, add 5 clocks.

387™ DX MCP Extensions to the 386™ DX CPU Instruction Set (Continued)

Instruction	Encoding			Clock Count Range			
	Byte 0	Byte 1	Optional Bytes 2–6	32-Bit Real	32-Bit Integer	64-Bit Real	16-Bit Integer
CONSTANTS (Continued)							
FLDL2E = Load $\log_2(e)$ into ST(0)	ESC 001	1110 1010		26–36			
FLDLG2 = Load $\log_{10}(2)$ into ST(0)	ESC 001	1110 1100		25–35			
FLDLN2 = Load $\log_e(2)$ into ST(0)	ESC 001	1110 1101		26–38			
ARITHMETIC							
FADD = Add							
Integer/real memory with ST(0)	ESC MF 0	MOD 000 R/M	SIB/DISP	12–29	34–56	15–34	38–64
ST(i) and ST(0)	ESC d P 0	11000 ST(i)		12–26[b]			
FSUB = Subtract							
Integer/real memory with ST(0)	ESC MF 0	MOD 10 R R/M	SIB/DISP	12–29	34–56	15–34	38–64[c]
ST(i) and ST(0)	ESC d P 0	1110 R R/M		12–26[d]			
FMUL = Multiply							
Integer/real memory with ST(0)	ESC MF 0	MOD 001 R/M	SIB/DISP	19–32	43–71	23–53	46–74
ST(i) and ST(0)	ESC d P 0	1100 1 R/M		17–50[e]			
FDIV = Divide							
Integer/real memory with ST(0)	ESC MF 0	MOD 11 R R/M	SIB/DISP	77-85	101–114[f]	81–91	105–124[g]
ST(i) and ST(0)	ESC d P 0	1111 R R/M		77-80[h]			
FSQRT[i] = Square root	ESC 001	1111 1010		97–111			
FSCALE = Scale ST(0) by ST(1)	ESC 001	1111 1101		44–82			
FPREM = Partial remainder	ESC 001	1111 1000		56–140			
FPREM1 = Partial remainder (IEEE)	ESC 001	1111 0101		81–168			
FRNDINT = Round ST(0) to integer	ESC 001	1111 1100		41–62			
FXTRACT = Extract components of ST(0)	ESC 001	1111 0100		42–63			
FABS = Absolute value of ST(0)	ESC 001	1110 0001		14–21			
FCHS = Change sign of ST(0)	ESC 001	1110 0000		17–24			

Shaded areas indicate instructions not available in 8087/80287.

NOTES:
b. Add 3 clocks to the range when d = 1.
c. Add 1 clock to **each** range when R = 1.
d. Add 3 clocks to the range when d = 0.
e. typical = 52 (When d = 0, 46–54, typical = 49).
f. Add 1 clock to the range when R = 1.
g. 135–141 when R = 1.
h. Add 3 clocks to the range when d = 1.
i. $-0 \leq ST(0) \leq +\infty$.

387™ DX MCP Extensions to the 386™ DX CPU Instruction Set (Continued)

Instruction	Encoding			Clock Count Range
	Byte 0	Byte 1	Optional Bytes 2–6	
TRANSCENDENTAL				
FCOS[k] = Cosine of ST(0)	ESC 001	1111 1111		122–680
FPTAN[k] = Partial tangent of ST(0)	ESC 001	1111 0010		162–430[j]
FPATAN = Partial arctangent	ESC 001	1111 0011		250–420
FSIN[k] = Sine of ST(0)	ESC 001	1111 1110		121–680
FSINCOS[k] = Sine and cosine of ST(0)	ESC 001	1111 1011		150–650
F2XM1[l] = $2^{ST(0)} - 1$	ESC 001	1111 0000		167–410
FYL2X[m] = ST(1) * $\log_2(ST(0))$	ESC 001	1111 0001		99–436
FYL2XP1[n] = ST(1) * $\log_2(ST(0) + 1.0)$	ESC 001	1111 1001		210–447
PROCESSOR CONTROL				
FINIT = Initialize MCP	ESC 011	1110 0011		33
FSTSW AX = Store status word	ESC 111	1110 0000		13
FLDCW = Load control word	ESC 001	MOD 101 R/M	SIB/DISP	19
FSTCW = Store control word	ESC 101	MOD 111 R/M	SIB/DISP	15
FSTSW = Store status word	ESC 101	MOD 111 R/M	SIB/DISP	15
FCLEX = Clear exceptions	ESC 011	1110 0010		11
FSTENV = Store environment	ESC 001	MOD 110 R/M	SIB/DISP	103–104
FLDENV = Load environment	ESC 001	MOD 100 R/M	SIB/DISP	71
FSAVE = Save state	ESC 101	MOD 110 R/M	SIB/DISP	375–376
FRSTOR = Restore state	ESC 101	MOD 100 R/M	SIB/DISP	308
FINCSTP = Increment stack pointer	ESC 001	1111 0111		21
FDECSTP = Decrement stack pointer	ESC 001	1111 0110		22
FFREE = Free ST(i)	ESC 101	1100 0 ST(i)		18
FNOP = No operations	ESC 001	1101 0000		12

Shaded areas indicate instructions not available in 8087/80287.

NOTES:
j. These timings hold for operands in the range $|x| < \pi/4$. For operands not in this range, up to 76 additional clocks may be needed to reduce the operand.
k. $0 \leq |ST(0)| < 2^{63}$.
l. $-1.0 \leq ST(0) \leq 1.0$.
m. $0 \leq ST(0) < \infty$, $-\infty < ST(1) < +\infty$.
n. $0 \leq |ST(0)| < (2 - SQRT(2))/2$, $-\infty < ST(1) < +\infty$.

5

APPENDIX A
COMPATIBILITY BETWEEN THE 80287 AND THE 8087

The 80286/80287 operating in Real-Address mode will execute 8086/8087 programs without major modification. However, because of differences in the handling of numeric exceptions by the 80287 MCP and the 8087 MCP, exception-handling routines *may* need to be changed.

This appendix summarizes the differences between the 80287 MCP and the 8087 MCP, and provides details showing how 8086/8087 programs can be ported to the 80286/80287.

1. The MCP signals exceptions through a dedicated ERROR# line to the 80286. The MCP error signal does not pass through an interrupt controller (the 8087 INT signal does). Therefore, any interrupt-controller-oriented instructions in numeric exception handlers for the 8086/8087 should be deleted.
2. The 8087 instructions FENI/FNENI and FDISI/FNDISI perform no useful function in the 80287. If the 80287 encounters one of these opcodes in its instruction stream, the instruction will effectively be ignored—none of the 80287 internal states will be updated. While 8086/8087 containing these instructions may be executed on the 80286/80287, it is unlikely that the exception-handling routines containing these instructions will be completely portable to the 80287.
3. Interrupt vector 16 must point to the numeric exception handling routine.
4. The ESC instruction address saved in the 80287 includes any leading prefixes before the ESC opcode. The corresponding address saved in the 8087 does not include leading prefixes.
5. In Protected-Address mode, the format of the 80287's saved instruction and address pointers is different than for the 8087. The instruction opcode is not saved in Protected mode—exception handlers will have to retrieve the opcode from memory if needed.
6. Interrupt 7 will occur in the 80286 when executing ESC instructions with either TS (task switched) or EM (emulation) of the 80286 MSW set (TS = 1 or EM = 1). If TS is set, then a WAIT instruction will also cause interrupt 7. An exception handler should be included in 80286/80287 code to handle these situations.
7. Interrupt 9 will occur if the second or subsequent words of a floating-point operand fall outside a segment's size. Interrupt 13 will occur if the starting address of a numeric operand falls outside a segment's size. An exception handler should be included in 80286/80287 code to report these programming errors.
8. Except for the processor control instructions, all of the 80287 numeric instructions are automatically synchronized by the 80286 CPU—the 80286 automatically tests the BUSY# line from the 80287 to ensure that the 80287 has completed its previous instruction before executing the next ESC instruction. No explicit WAIT instructions are required to assure this synchronization. For the 8087 used with 8086 and 8088 processors, explicit WAITs are required before each numeric instruction to ensure synchronization. Although 8086/8087 programs having explicit WAIT instructions will execute perfectly on the 80286/80287 without reassembly, these WAIT instructions are unnecessary.
9. Since the 80287 does not require WAIT instructions before each numeric instruction, the ASM286 assembler does not automatically generate these WAIT instructions. The ASM86 assembler, however, automatically precedes every ESC instruction with a WAIT instruction. Although numeric routines generated using the ASM86 assembler will generally execute correctly on the 80286/80287, reassembly using ASM286 may result in a more compact code image.

   The processor control instructions for the 80287 may be coded using either a WAIT or No-WAIT form of mnemonic. The WAIT forms of these instructions cause ASM286 to precede the ESC instruction with a CPU WAIT instruction, in the identical manner as does ASM86.

DATA SHEET REVISION REVIEW

The following list represents the key differences between this and the -002 versions of the 387™ Math Coprocessor Data Sheet. Please review this summary carefully.

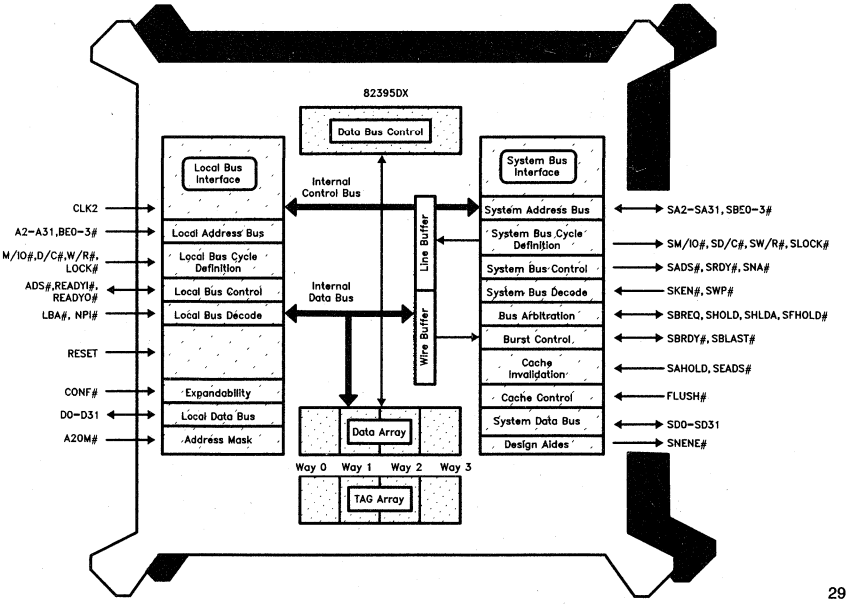
1. Updated I_{CC} max and typical specs to reflect CHMOS IV process.
2. Updated instruction clock counts.
3. Change pins K3, L9 back to tie high insted of V_{CC}.
4. Corrected typographical errors in A.C. Characteristics table. Affected pins were NUMCLK2 Rise Time test conditions, READYO# Min Out Delay at 16 MHz, and Max Data Out Delay at 25 MHz.

82395DX
HIGH PERFORMANCE 386™ SMART CACHE

- **Optimized 386 DX Microprocessor Companion**
- **Integrated 16KB Data RAM**
- **4 Way SET Associative with Pseudo LRU Algorithm**
- **Write Buffer Architecture**
- **Integrated 4 Double Word Write Buffer**
- **16 Byte Line Size**
- **Integrated 387™ Math Coprocessor and Weitek 3167 Floating Point Coprocessor Decode Logic**
- **Concurrent Line Buffer Cacheing**
- **Multiprocessor Support**
- **Expandable - up to 64KB**
- **Supports i486™ Microprocessor-like Burst**
- **Dual Bus Architecture — Snooping Maintains Cache Coherency**
- **20, 25 and 33MHz Clock**
- **196 lead PQFP package**

The 82395DX High Performance 386 Smart Cache is a low cost, high integration, 32-Bit peripheral for Intel's 386™ DX Microprocessor. It stores a copy of frequently accessed code or data from main memory to on chip data RAM that can be accessed in zero wait states. The 82395DX enables the 386 DX Microprocessor to run at near its full potential by reducing the average number of wait states seen by the CPU to nearly zero. The dual bus architecture allows another bus master to access the System Bus while the 386 DX Microprocessor can operate out of the 82395DX's cache on the Local Bus. The 82395DX has a snooping mechanism which maintains cache coherency during these cycles.

The 82395DX is completely software transparent, protecting the integrity of system software. High performance, low cost and board space saving are achieved due to the high integration and new write buffer architecture.



290382-1

386™ Smart Cache

387™ DX, 386™ DX, and i486™ are trademarks of Intel Corporation

October 1990
Order Number: 290382-001

82395DX
HIGH PERFORMANCE 386™ SMART CACHE

CONTENTS

PAGE

0.0 DESIGNER SUMMARY 5-471
0.1 Pin Out 5-471
0.2 Quick Pin Reference 5-473

1.0 82395DX Functional Overview 5-478
1.1 Introduction 5-478
1.2 Features 5-479
1.2.1 82385-Like Features 5-479
1.2.2 New Features 5-479

2.0 82395DX CACHE SYSTEM DESCRIPTION 5-480
2.1 82395DX Cache Organization 5-480
2.1.1 82395DX Cache Structure and Terminology 5-482
2.2 Pseudo LRU Algorithm 5-484
2.3 Four Way Set Associative Cache Organization 5-486
2.3.1 Cache Read Hits 5-486
2.3.2 Cache Read Misses 5-486
2.3.3 Other Operations That Affect the Cache and Cache Directory 5-486
2.4 Concurrent Line Buffer Cacheing 5-486
2.5 Cache Control 5-487
2.6 Cache Invalidation 5-487
2.7 Cache Flushing 5-487
2.8 Cache Directory Accesses and Arbitration 5-487
2.9 Cache Memory Description 5-489

3.0 PIN DESCRIPTION 5-491
3.1 Local Bus Interface Pins 5-491
3.1.1 386 DX Microprocessor/ 82395DX Clock (CLK2 I) 5-491
3.1.2 Local Address Bus 5-491
3.1.2.1 Local Bus Address Lines (A2–A31 I) 5-491
3.1.2.2 Local Bus Byte Enables (BE3#–BE0# I) 5-491
3.1.3 Local Bus Cycle Definition .. 5-491

CONTENTS

PAGE

3.1.3.1 Local Bus Cycle Definition Signals (W/R#, D/C#, M/IO# I) 5-491
3.1.3.2 Local Bus Lock (LOCK# I) 5-491
3.1.4 Local Bus Control 5-491
3.1.4.1 Address Status (ADS# I) 5-491
3.1.4.2 Local Bus Ready (READYI# I) 5-492
3.1.4.3 Local Bus Ready Output (READYO# I/O) 5-492
3.1.5 Reset (RESET I) 5-492
3.1.6 Configuration (CONF# I) 5-492
3.1.7 Local Data Bus 5-492
3.1.7.1 Local Bus Data Lines (D0–D31 I/O) 5-492
3.1.8 Local Bus Decode Pins 5-492
3.1.8.1 Local Bus Access Indication (LBA# I) 5-492
3.1.8.2 No Post Input (NPI# I) 5-493
3.1.9 Address Mask 5-494
3.1.9.1 Address Bit 20 Mask (A20M#I) 5-494
3.2 System Bus Interface Pins 5-494
3.2.1 System Address Bus 5-494
3.2.1.1 System Bus Address Lines (SA2–SA31 I/O) 5-494
3.2.1.2 System Bus Byte Enables (SBE3#–SBE0# 0) 5-494
3.2.2 System Bus Cycle Definition 5-494
3.2.2.1 System Bus Cycle Definition (SW/R#, SD/C# 0, SM/IO#) 5-494
3.2.2.2 System Bus Lock (SLOCK# O) 5-494
3.2.3 System Bus Control 5-494
3.2.3.1 System Bus Address Status (SADS# O) 5-494
3.2.3.2 System Bus Ready (SRDY# I) 5-494

5

CONTENTS

PAGE

3.2.3.3 System Bus Next Address (SNA# I) 5-495
3.2.4 Bus Arbitration 5-495
3.2.4.1 System Bus Request (SBREQ O) 5-495
3.2.4.2 System Bus Hold Request (SHOLD I) 5-495
3.2.4.3 System Bus Hold Acknowledge (SHLDA O) 5-495
3.2.4.4 System Bus Fast Hold Request (SFHOLD# I) 5-495
3.2.5 Burst Control 5-495
3.2.5.1 System Bus Burst Ready (SBRDY# I) 5-495
3.2.5.2 System Bus Burst Last Cycle Indicator (SBLAST# O) 5-495
3.2.6 Cache Invalidation 5-496
3.2.6.1 System Bus Address Hold (SAHOLD I) 5-496
3.2.6.2 System Bus External Address Strobe (SEADS# I) 5-496
3.2.7 Cache Control 5-496
3.2.7.1 Flush (FLUSH# I) 5-496
3.2.8 System Data Bus 5-496
3.2.8.1 System Bus Data Lines (SD0–SD31 I/O) 5-496
3.2.9 System Bus Decode Pins ... 5-496
3.2.9.1 System Cacheability Indication (SKEN# I) 5-496
3.2.9.2 System Write Protect Indication (SWP# I) 5-496
3.2.10 Design Aides 5-496
3.2.10.1 System Bus Next Near Indication (SNENE# O) 5-496
3.3 Pinout Summary Tables 5-497

4.0 BASIC FUNCTIONAL DESCRIPTION 5-498
4.1 Cacheable Accesses 5-499
4.1.1 Cacheable Read Hit Accesses 5-499
4.1.2 Cacheable Read Miss Accesses 5-499
4.1.2.1 Burst Bus 5-500
4.1.3 Cache Write Accesses 5-500

CONTENTS

PAGE

4.2 Noncacheable System Bus Accesses 5-500
4.3 Local and System Bus Concurrency 5-501
4.4 Disabling the 82395DX 5-504
4.5 System Description and Device Selection 5-504
4.6 Auto Configuration 5-504
4.7 Address Mapping 5-506
4.8 Multi 82395DX Operation Description 5-506
4.9 Signal Driving in Multi 82395DX Environment 5-506
4.9.1 Local Bus Signals 5-506
4.9.2 System Bus Signals 5-506
4.10 SHOLD/SHLDA/SBREQ Arbitration Mechanism 5-507
4.11 System Description 5-507

5.0 PROCESSOR INTERFACE 5-509
5.1 Hardware Interface 5-509
5.2 Nonpipelined Local Bus 5-509
5.3 Local Bus Response to Hit Cycles 5-509
5.4 Local Bus Response to Miss Cycles 5-509
5.5 Local Bus Control Signals - ADS#, READYI# 5-509
5.6 82395's Response to the 386 DX Microprocessor Cycles 5-510
5.6.1 Locked Cycles 5-510
5.6.2 I/O, HALT/SHUTDOWN 5-510
5.6.3 LBA# Cycles 5-510
5.6.4 NPI# Cycles 5-510
5.6.5 LBA#/NPI# Timing 5-510
5.7 82395DX READYO# Generation 5-511
5.8 A20 Mask Signal 5-512
5.9 82395DX Cycle Overview 5-512

6.0 SYSTEM BUS INTERFACE 5-515
6.1 System Bus Cycle Types 5-517
6.1.1 Buffered Write Cycle 5-517
6.1.2 Non Buffered Write Cycle ... 5-518
6.1.3 Write Protected Cycles 5-519

CONTENTS

PAGE

6.1.4 Non Cacheable Read Cycle 5-519
6.1.5 Cacheable Read Miss Cycles 5-520
6.1.5.1 Aborted Line Fill (ALF) Cycles 5-520
6.1.5.2 Line Fill Cycles 5-522
6.2 82395DX Latency in System Bus Accesses 5-526
6.3 SHLDA Latency 5-526
6.4 Cache Consistency Support 5-526
6.5 Bus Deadlock Resolution Support 5-528
6.6 Arbitration Mechanism 5-528
6.7 Next Near Cycles 5-529
6.8 Write Buffer 5-530

7.0 TESTABILITY FEATURES 5-530
7.1 SRAM Test Mode 5-530
7.2 Tristate Output Test Mode 5-533

8.0 MECHANICAL DATA 5-534
8.1 Introduction 5-534
8.2 Pin Assignment 5-534
8.3 Package Dimensions and Mounting 5-534
8.4 Package Thermal Specification 5-534

9.0 ELECTRICAL DATA 5-538
9.1 Power and Grounding 5-538
9.1.1 Power Decoupling Recommendations 5-538
9.1.2 Resistor Recommendations 5-538
9.2 Absolute Maximum Ratings 5-539
9.3 DC Specifications 5-539
9.4 AC Characteristics 5-540
9.4.1 Timing Considerations for Cache Extensions 5-540
9.4.2 AC Characteristic Tables 5-541

APPENDIX A TERMS 5-546

TABLES

Table 0.1 82395DX 196-Lead PQFP Package Pin Description 5-472

CONTENTS

PAGE

Table 2.1 82395DX Cache Organization 5-481
Table 3.1 Input Pins 5-497
Table 3.2 Output Pins 5-498
Table 3.3 Input-Output Pins 5-498
Table 4.1 386 DX Microprocessor Bus Cycle Definition with Cacheability 5-500
Table 4.2 Address Mapping for 1-4 82395DX Systems 5-506
Table 4.3 Local Bus Signal Connections in Multi-82395DX Systems 5-507
Table 4.4 System Bus Signal Connections in Multi-82395DX Systems 5-508
Table 5.1 386 DX Microprocessor Bus Cycle Definition 5-512
Table 5.2 Activity by Functional Groupings 5-513
Table 5.3 Activity in Line Buffer Hit Cycles 5-514
Table 5.4 Activity in the Line Buffer During ALF Cycles 5-514
Table 5.5 Activity in Test Cycles 5-515
Table 6.1 Line Fill Address Order 5-523
Table 7.1 SRAM Memory Map 5-531
Table 7.2 Cache Address Allocation 5-532
Table 7.3 TAGRAM Address Allocation 5-532
Table 8.1 Symbol List and Dimensions for 196 Lead Plastic Quad Flat Pack Package 5-536
Table 9.1 Pullup Resistor Recommendations 5-539
Table 9.2 DC Specifications 5-539
Table 9.3 Local Bus Signal AC Parameters 5-541
Table 9.4 System Bus Signal AC Parameters 5-542

FIGURES

Figure 0.1 82395DX 196 Lead PQFP Package Pin Orientation 5-471
Figure 1.1 System Block Diagram 5-478
Figure 2.1 82395DX Cache Organization 5-481
Figure 2.2 82395DX Cache Directory Organization 5-482

5

CONTENTS

PAGE

Figure 2.3 82395DX Cache Hit Logic ... 5-483
Figure 2.4 Pseudo LRU Decision Tree .. 5-484
Figure 2.5 Four Way SET Associative Cache Organization 5-485
Figure 2.6 Interposing in the Cache Directory 5-488
Figure 2.7 Cache Directory and Cache Accesses 5-490
Figure 3.1 CLK2 and Internal Clock 5-493
Figure 3.2 RESET/Internal Phase Relationship 5-493
Figure 3.3 Sampling LBA# During RESET 5-493
Figure 4.1 Read Hit Cycles During a Line Fill 5-501
Figure 4.2 Cache Read Hit Cycles while Executing a Buffered Write on the System Bus 5-502
Figure 4.3 Buffered Write Cycles During a Line Fill 5-502
Figure 4.4 SWP# and SKEN# Timing .. 5-503
Figure 4.5 Self-Configuration of Four 82395DXs 5-505
Figure 4.6 System Description 5-508
Figure 5.1 Valid Time of LBA# and NPI# 5-511
Figure 5.2 Externally Delayed READY .. 5-511
Figure 5.3 A20 Mask Logic 5-512
Figure 5.4 Valid Time of A20M# 5-512
Figure 6.1 SB State Machine 5-516
Figure 6.2 Single Buffered Write Cycle 5-517
Figure 6.3 Multiple Buffered Write Cycles During System Bus HOLD ... 5-517
Figure 6.4 I/O Write Cycle 5-518
Figure 6.5 LOCK#ed Read Modify Write Cycle 5-518
Figure 6.6 I/O Read Cycle 5-519
Figure 6.7 LOCK#ed INTA Cycle 5-519
Figure 6.8 Aborted Line Fill Cycle 5-520
Figure 6.9 Line Fill Without Burst or Pipeline 5-521
Figure 6.9A Burst Mode Line Fill Followed by a Line Buffer Hit Cycle 5-521

CONTENTS

PAGE

Figure 6.10 Pipelined Line Fill 5-523
Figure 6.11 Fastest Burst Cycle (one clock burst) 5-524
Figure 6.12 Burst Read (two clock burst) 5-524
Figure 6.13 Interrupted Burst Read (two clock burst) 5-525
Figure 6.14 SAHOLD Behavior in Pipelined Cycles 5-527
Figure 6.15 Multiple 82395DX Bus Arbitration Scheme 5-528
Figure 6.16 SHOLD/SHLDA/SBREQ Mechanism 5-529
Figure 7.1 SRAM Mode Read Cycle ... 5-532
Figure 7.2 SRAM Mode Write Cycle ... 5-532
Figure 7.3 Entering the Tristate Test Mode 5-533
Figure 8.1 Principal Dimensions and Datums 5-535
Figure 8.2 Typical Lead 5-536
Figure 8.3 Detail C 5-536
Figure 8.4 Junction to Ambient Thermal Resistance vs Power 5-537
Figure 8.5 Junction to Case Thermal Resistance vs Power 5-537
Figure 8.6 Junction to Ambient Thermal Resistance vs Air Flow Rate 5-538
Figure 9.1 Drive Levels and Measurement Points for AC Specifications 5-540
Figure 9.2 AC Timing Waveforms - Local Bus Input Setup and Hold Timing 5-543
Figure 9.3 AC Timing Waveforms - System Bus Input Setup and Hold Timing 5-543
Figure 9.4 AC Timing Waveforms - Output Valid Delays 5-544
Figure 9.5 AC Timing Waveforms - Output Float Delays 5-544
Figure 9.6 Typical Output Valid Delay vs Load Capacitance at Maximum Operating Temperature (C_L = 50 pF) 5-545

0.0 DESIGNER SUMMARY

0.1 Pin Out

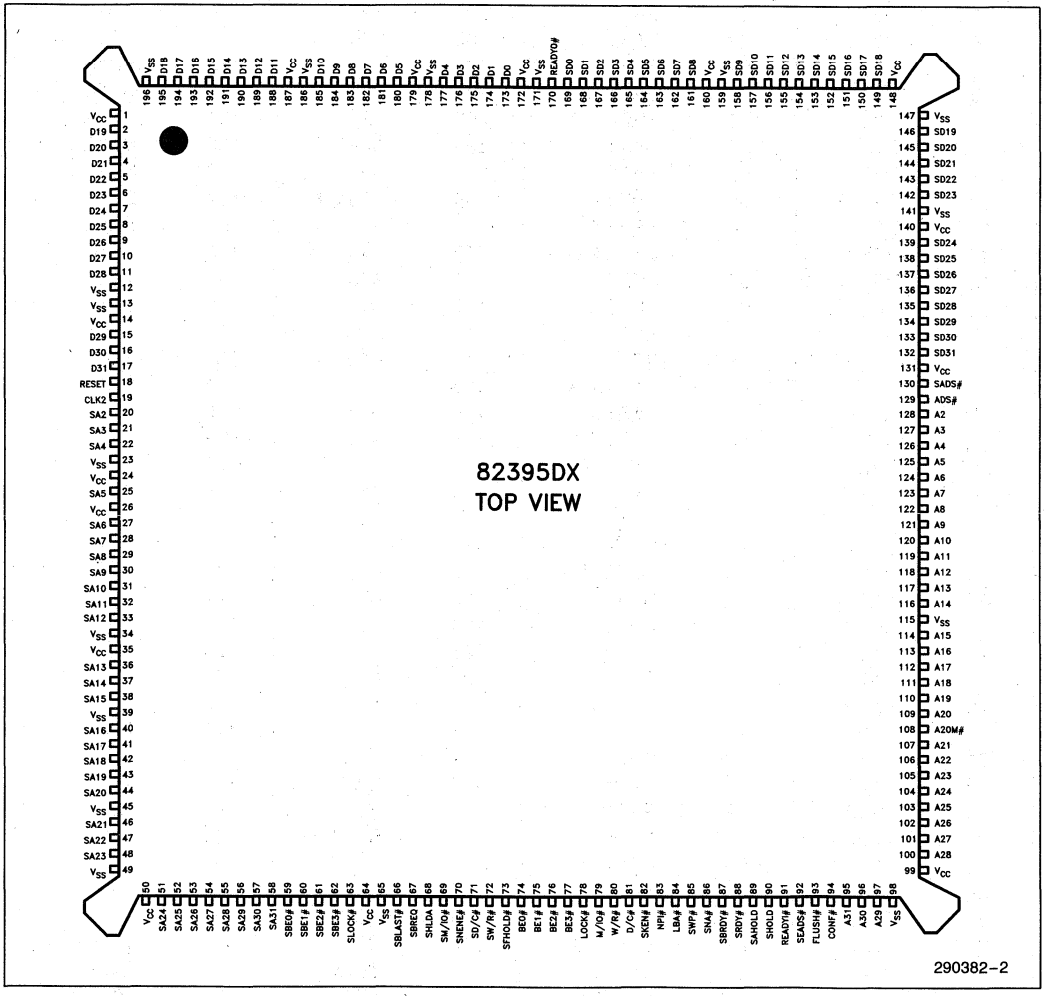


Figure 0.1 - 82385DX 196 Lead PQFP Package Pin Orientation

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	VCC	50	VCC	99	VCC	148	VCC
2	D19	51	SA24	100	A28	149	SD18
3	D20	52	SA25	101	A27	150	SD17
4	D21	53	SA26	102	A26	151	SD16
5	D22	54	SA27	103	A25	152	SD15
6	D23	55	SA28	104	A24	153	SD14
7	D24	56	SA29	105	A23	154	SD13
8	D25	57	SA30	106	A22	155	SD12
9	D26	58	SA31	107	A21	156	SD11
10	D27	59	SBE0#	108	A20M#	157	SD10
11	D28	60	SBE1#	109	A20	158	SD9
12	VSS	61	SBE2#	110	A19	159	VSS
13	VSS	62	SBE3#	111	A18	160	VCC
14	VCC	63	SLOCK#	112	A17	161	SD8
15	D29	64	VCC	113	A16	162	SD7
16	D30	65	VSS	114	A15	163	SD6
17	D31	66	SBLAST#	115	VSS	164	SD5
18	RESET	67	SBREQ	116	A14	165	SD4
19	CLK2	68	SHLDA	117	A13	166	SD3
20	SA2	69	SM/IO#	118	A12	167	SD2
21	SA3	70	SNENE#	119	A11	168	SD1
22	SA4	71	SD/C#	120	A10	169	SD0
23	VSS	72	SW/R#	121	A9	170	READY0#
24	VCC	73	SFHOLD#	122	A8	171	VSS
25	SA5	74	BE0#	123	A7	172	VCC
26	VCC	75	BE1#	124	A6	173	D0
27	SA6	76	BE2#	125	A5	174	D1
28	SA7	77	BE3#	126	A4	175	D2
29	SA8	78	LOCK#	127	A3	176	D3
30	SA9	79	M/IO#	128	A2	177	D4
31	SA10	80	W/R#	129	ADS#	178	VSS
32	SA11	81	D/C#	130	SADS#	179	VCC
33	SA12	82	SKEN#	131	VCC	180	D5
34	VSS	83	NPI#	132	SD31	181	D6
35	VCC	84	LBA#	133	SD30	182	D7
36	SA13	85	SWP#	134	SD29	183	D8
37	SA14	86	SNA#	135	SD28	184	D9
38	SA15	87	SBRDY#	136	SD27	185	D10
39	VSS	88	SRDY#	137	SD26	186	VSS
40	SA16	89	SAHOLD	138	SD25	187	VCC
41	SA17	90	SHOLD	139	SD24	188	D11
42	SA18	91	READYI#	140	VCC	189	D12
43	SA19	92	SEADS#	141	VSS	190	D13
44	SA20	93	FLUSH#	142	SD23	191	D14
45	VSS	94	CONF#	143	SD22	192	D15
46	SA21	95	A31	144	SD21	193	D16
47	SA22	96	A30	145	SD20	194	D17
48	SA23	97	A29	146	SD19	195	D18
49	VSS	98	VSS	147	VSS	196	VSS

Table 0.1 - 82395DX 196-Pin PQFP Pin Description

0.2 Quick Pin Reference

What follows is a brief pin description. For more details refer to chapter 3.

Symbol	Type	Function
CLK2	I	This signal provides the fundamental timing for the 82395DX. All external timing parameters are specified with respect to the rising edge of CLK2.
Local Address Bus		
A2–31	I	A2–31 are the Local Bus address lines. These signals along with the byte enable signals, define the physical area of memory or input/output space accessed.
BE0–3#	I	The byte enable signals are used to determine which bytes are accessed in partial cache write cycles. These signals are ignored for Cache Read Hit cycles. For all System Bus memory read cycles (except the last three cycle of a Line Fill), these signals are mirrored by the SBE0–3# signals.
Local Bus Cycle Definition		
W/R# D/C# M/IO#	I I I	The write/read, data/code and memory/input-output signals are the primary bus definition signals directly connected to the 386 DX Microprocessor. They become valid as the ADS# signal is sampled active. The bus definition signals are not driven by the 386 DX Microprocessor during bus hold and follow the timing of the address bus.
LOCK#	I	The Local Bus LOCK# signal indicates that the current bus cycle is LOCK#ed. LOCK#ed cycles are treated as non-cacheable cycles, except that LOCK#ed write hit cycles update the cache.
Local Bus Control		
ADS#	I	The address status pin, an output of the 386 DX Microprocessor, indicates that new and valid information is currently available on the Local Bus. The signals that are valid when ADS# is activated are: A2–31, BE0–3#, W/R#, D/C#, M/IO#, LOCK#, NPI# and LBA#
READYI#	I	This is the READY input signal seen by the Local Bus master. Typically it is a logical OR between the 82395DX generated READYO# and READY# signals generated by other Local Bus masters (optional). It is used by the 82395DX, along with the ADS# signal, to keep track of the 386 DX Microprocessor bus state.
READYO#	I/O	This is the Local Bus READY output that is used to terminate all types of 386 DX Microprocessor bus cycles, except for 386 DX Microprocessor Local Bus cycles which must be terminated by the Local Bus device being accessed. This signal is wired-OR with parallel 82395DX READYO# signals in a multi-82395DX system. The READYO# pin may serve as READY# for the 387 DX Math Coprocessor.
RESET		
RESET	I	The RESET signal forces the 82395DX to begin execution at a known state. The RESET falling edge is used by the 82395DX to set the phase of its internal clock identical to the 386 DX Microprocessors internal clock. RESET falling edge must satisfy the appropriate setup and hold times (T14, T15b) for proper chip operation. RESET must remain active for at least 1ms after the power supply and CLK2 input have reached their proper DC and AC specifications.
Configuration		
CONF#	I	The activity on the CONF# input during and after RESET allows the 82395DX to configure itself to operate in the specified address range. Refer to chapter 4 for 1, 2 or 4 82395DXs operation.

0.2 Quick Pin Reference (Continued)

Symbol	Type	Function
Local Data Bus		
D0–31	I/O	These are the Local Bus data lines of the 82395DX. They must be connected to the D0–31 pins of the 386 DX Microprocessor.
Local Bus Decode Pins		
LBA#	I	This is the Local Bus Access indication. It instructs the 82395DX that the cycle currently in progress is targeted to a Local Bus device. This results in the cycle being ignored by the 82395DX. The 387 DX Math Coprocessor is considered a Local Bus devcie but LBA# need not be generated. If LBA# is asserted at the falling edge of RESET accesses to Weitek 3167 Floating-Point Coprocessor address space are decoded as Local Bus cycles. Note that LBA# cycles have priority over all other cycle types.
NPI#	I	The No Post Input signal instructs the 82395DX that the write cycle currently in progress must not be posted in the write buffer. NPI# is sampled at the falling edge of CLK at the end of T1 (see figure 5.1).
Address Mask		
A20M#	I	Address bit 20 Mask when active, forces the A20 input as seen by the 82395DX to logic "0", regardless of the actual value on the A20 input pin. A20M# emulates the address wraparound at 1 MByte which occurs on the 8086. This pin is asynchronous but must meet setup and hold times (t47 and t48) to guarantee recognition in a specific clock. It must be asserted two clock cycles before ADS# is sampled active (see figure 5.3). It must be stable throughout Local Bus memory cycles.
System Address Bus		
SA2–3 SA4–31	O I/O	These are the System Bus address lines of the 82395DX. When driven by the 82395DX, these signals, along with the System Bus byte enables define the physical area of memory or input/output space being accessed. During bus HOLD or address HOLD, the I/O signals serve as inputs for the cache invalidation cycle.
SB0–3#	O	These are the Byte Enable signals for the System Bus. The 82395DX drives these pins identically to BE0–3# in all System Bus cycles except Line Fills. In Line Fills these signals are driven identically to BE0–3# for the first read cycle of the Line Fill. They are all driven active in the remaining cycles of the Line Fill.
System Bus Cycle Definition		
SW/R# SD/C# SM/IO#	O O O	The System Bus write/read, data/code and memory/input-output signals are the System Bus cycle definition pins. When the 82395DX is the System Bus master, it drives these signals identically to the 386 DX Microprocessor cycle definition encoding.
SLOCK#	O	The System Bus LOCK# signal indicates that the current cycle is LOCK#ed. The 82395DX has exclusive access to the System Bus across bus cycle boundries until this signal is negated. The 82395DX does not acknowledge a bus HOLD request while this signal is asserted. The 82395DX asserts SLOCK# when the System Bus is available and a LOCK#ed cycle was started on the Local Bus that requires System Bus service. SLOCK# is negated only after completion of all LOCK#ed System Bus cycles and negation of the LOCK# signal.

0.2 Quick Pin Reference (Continued)

Symbol	Type	Function
System Bus Control		
SADS#	O	The System Bus ADdress Status signal is used to indicate that new and valid information is currently being driven onto the System Bus. The signals that are valid when SADS# is driven low are: SA2–31, SBE0–3#, SW/R#, SD/C#, SM/IO# and SLOCK#
SRDY#	I	The System Bus ReaDY# signal indicates that the current System Bus cycle is complete. When SRDY# is sampled asserted it indicates one of two things. In response to a read request it indicates that the external system has presented valid data on the system data bus. In response to a write request it indicates that the external system has accepted the 82395DX's data. This signal is ignored when the System Bus is in STi, STH, ST1 or ST1P states. At the first read cycle of a Line Fill SRDY#, SBRDY# and SNA# determine if the Line Fill will proceed as a burst/non-burst, pipelined/non-pipelined Line Fill. Once a burst Line Fill has started, if SRDY# is returned in the 2nd or 3rd DW, the burst Line Fill will be interrupted and the cache will not be updated. The 1st DW will already have been transferred to the CPU. In the 4th DW of a Line Fill both SRDY# and SBRDY# have the same affect. They indicate the end of the Line Fill.
SNA#	I	The System Bus Next Address signal, when active, indicates that a pipelined address cycle will be executed. It is sampled by the 82395DX at the rising edge of CLK in ST2 and ST1P cycles. If this signal is sampled active then burst Line Fills are disabled. This signal is ignored once a burst Line Fill begins.
Bus Arbitration		
SBREQ	O	The System Bus REQuest signal is the internal cycle pending signal. This indicates to the outside world that internally the 82395DX has generated a bus request (due to the CPU's request that requires access to the System Bus). It is generated whether the 82395DX owns the bus or not and can be used to arbitrate among the various masters on the System Bus. If the bus is available and the cycle starts immediately this signal will not be activated for cache read miss cycles.
SHOLD	I	The System Bus HOLD request indicates that another master must have complete control of the entire System Bus. When SHOLD is sampled asserted the 82395DX completes the current System Bus cycle or sequence of LOCK#ed cycles, before driving SHLDA active. In the same clock that SHLDA went active all the System Bus outpus and I/O pins are floated (with the exception of SHLDA and SBREQ). The 82395DX stays in this state until SHOLD is negated. SHOLD is recognized during RESET.
SHLDA	O	The System Bus HOLD Acknowledge signal is driven active by the 82395DX in response to a hold request. It indicates that the 82395DX has given the bus to another System Bus master. It is driven active in the same clock that the 82395DX floats it's System Bus. When leaving a bus HOLD, SHLDA is driven inactive and the 82395DX resumes driving the bus in the same clock. The 82395DX is able to support CPU Local Bus activities during System Bus HOLD.

0.2 Quick Pin Reference (Continued)

Symbol	Type	Function
Bus Arbitration (Continued)		
SFHOLD#	I	The System Bus Fast HOLD Request signal indicates that another master needs immediate access to the System Bus. In response to SFHOLD# being sampled active, the 82395DX stops driving (in the next clock) the System Bus output and I/O pins (except SHLDA and SBREQ). Because the 82395DX always stops driving the System Bus in response to SFHOLD# active, no acknowledge is required. The System Bus output and I/O pins remain in the high impedance state until SFHOLD# is negated. It is the responsibility of the system designer to guarantee that bus cycles that are in progress when SFHOLD# is asserted are terminated correctly. This pin is recognized during RESET.
Burst Control		
SBRDY#	I	The System Bus Burst ReaDY signal performs the same function during a burst cycle that SRDY# does in a non-burst cycle. SBRDY# asserted indicates that the external system has presented valid data on the data pins in response to a burst Line Fill cycle. This signal is ignored when the System Bus is at STi, STH, ST1 or ST1P states. Note that in the fourth bus cycle of a Line Fill, SBRDY# and SRDY# have the same effect on the 82395DX. They indicate the end of the Line Fill. For all cycles other than burst Line Fills, SBRDY# and SRDY# have the same effect on the 82395DX.
SBLAST#	O	The System Bus Burst LAST cycle indicator signal indicates that the next time SBRDY# is returned the burst cycle is complete. It indicates to the external system that the next SBRDY# returned is treated as a normal SRDY# by the 82395DX. Another set of addresses will be driven with SADS# or the System Bus will go idle. SBLAST# is normally active. In a cache read miss cycle, which may proceed as a Line Fill, SBLAST# starts active. After determining whether or not the cycle is cacheable via SKEN#, SBLAST# is driven inactive. If it is a cacheable cycle, and SBRDY# terminates the first DW of the Line Fill, a burst Line Fill, SBLAST# will be driven active when the data is valid for the fourth DW of the Line Fill. If SRDY# terminates the first DW of the Line Fill, a non-burst Line Fill, SBLAST# is driven active in the cycle where SRDY# was sampled active.
Cache Invalidation		
SAHOLD	I	The System Bus Address HOLD request allows another bus master access to the address bus of the 82395DX. This is to indicate the address of an external cycle for performing an internal cache directory lookup and invalidation cycle. In response to this signal the 82395DX stops driving the System Bus address pins in the next cycle. No HOLD Acknowledge is required. Other System Bus signals can remain active during address hold. The 82395DX does not initiate another bus cycle during address hold. This pin is recognized during RESET.
SEADS#	I	The System Bus External ADress Strobe signal indicates that a valid external address has been driven onto the 82395DX System Bus address pins. This address will be used to perform an internal cache invalidation cycle. The maximum invalidation cycle rate is one every two clock cycles.

0.2 Quick Pin Reference (Continued)

Symbol	Type	Function
Cache Control		
FLUSH#	I	The FLUSH# pin, when sampled active for four clock cycles or more, causes the 82395DX to invalidate its entire TAG array. In addition, it is used to configure the 82395DX to enter various test modes. For details refer to chapter 7. This signal is asynchronous but must meet setup and hold times to guarantee recognition in any specific clock.
System Data Bus		
SD0–31	I/O	The System Bus Data lines of the 82395DX must be driven with appropriate setup and hold times for proper operation. These signals are driven by the 82395DX only during write cycles.
System Bus Decode Pins		
SKEN#	I	The System Cacheability ENable signal is used to determine if the current cycle running on the System Bus is cacheable or not. When the 82395DX generates a read cycle, SKEN# is sampled one clock before the first SBRDY# or SRDY# or one cycle before the first SNA# is sampled active (see chapter 6). If SKEN# is sampled active the cycle will be transformed into a Line Fill. Otherwise, the cache and cache directory will be unaffected. Note that SKEN# is ignored after the first cycle in a Line Fill. SKEN# is ignored for all System Bus cycles except for cache read miss cycles.
SWP#	I	The System Write Protect indicator signal is used to determine whether the current System Bus Line Fill cycle is write protected or not. In non-pipelined cycles, SWP# is sampled with the first SRDY# or SBRDY# of the Line Fill. In pipelined cycles, SWP# is sampled one clock phase after the first SNA# is sampled active (see figures 6.9-10). The Write Protect bit is sampled together with the TAG of each line in the 82395DX Cache Directory. In every cacheable write cycle the Write Protect bit is read. If active, the cycle will be a write protected cycle which is treated like a cacheable write miss cycle. It is buffered and it does not update the cache even if the addressed location is present in the cache.
Design Aides		
SNENE#	O	The System NExt NEar indicator signal indicates that the current System Bus memory cycle is to the same 2048 byte area as the previous memory cycle. Address lines A11–31 of the current System Bus memory cycle are identical to address lines A11–31 of the previous memory cycle. SNENE# can be used in an external DRAM system to run CAS# only cycles, thereby increasing the throughput of the memory system. SNENE# is valid for all memory cycles, and indicates that the current memory cycle is to the same 2048 byte area, even if there were idle or non-memory bus cycles since the last System Bus memory cycle. For the first cycle after the 82395DX has exited the HOLD state, or after SAHOLD was deactivated, this pin will be inactive.

1.0 82395DX FUNCTIONAL OVERVIEW

1.1 Introduction

The primary function of a cache is to provide local storage for frequently accessed memory locations. The cache intercepts memory references and handles them directly without transferring the request to the System Bus. This results in lower traffic on the System Bus and decreases latency on the local bus. This leads to improved performance for a processor on the Local Bus. By providing fast access to frequently used code and data, the cache is able to reduce the average memory access time of the 386 DX Microprocessor based system.

The 82395DX is a single chip cache subsystem specifically designed for use with the 386 DX Microprocessor. The 82395DX integrates 16KB cache, the Cache Directory and the Cache Control Logic onto one chip.

The 82395DX is expandable such that larger cache sizes are supported by cascading 82395DXs. In a single 82395DX system, the 82395DX can map 4 Giga bytes of main memory into a 16KB cache. In the maximum configuration of a four 82395DX system, the 4 Giga bytes of main memory are mapped into a 64KB cache. The cache is unified for code and data and is transparent to application software. The 82395DX provides a cache consistency mechanism which guarantees that the cache has the most recently updated version of the main memory. Consistency support has no performance impact on the 386 DX Microprocessor. Section 1.2 covers all the 82395DX features.

The 82395DX cache architecture is similar to the i486 Microprocessor's on-chip cache. The cache is four Way set associative with Pseudo LRU replacement algorithm. The line size is 16B and a full line is retrieved from the memory every cache miss. A TAG is associated with every 16B line.

The 82395DX architecture allows for cache read hit cycles to run on the Local Bus even when the System Bus is not available. 82395DX incorporates a new write buffer cache architecture, which allows the 386 DX Microprocessor to continue operation without waiting for write cycles to actually update the main memory.

A detailed description of the cache operation and parameters is included in chapter 2.

The 82395DX has an interface to two electrically isolated busses. The interface to the 386 DX Microprocessor bus is referred to as the Local Bus (LB) interface. The interface to the main memory and other system devices is referred to as the 82395DX System Bus (SB) interface. The SB interface emulates the 386 DX Microprocessor. The SB interface, as does the 386 DX Microprocessor, can be pipelined.

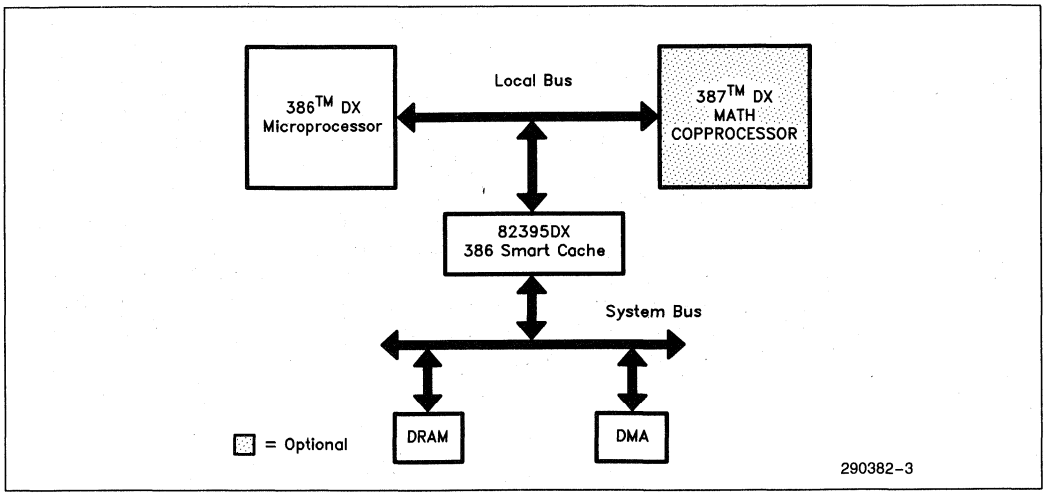


Figure 1.1 - System Block Diagram

In addition, it is enhanced by an optional burst mode for Line Fills. The burst mode provides faster line fills by allowing consecutive read cycles to be executed at a rate of up to one DW per clock cycle. Several bus masters (or several 82395DXs) can share the same System Bus and the arbitration is done via the SHOLD/SHLDA/SBREQ mechanism (similar to the i486 Microprocessor) along with SFHOLD#. Using these arbitration mechanisms, the 82395DX is able to support a multiprocessor system (multi 386 DX Microprocessor/82395DX systems sharing the same memory).

Cache consistency is maintained by the SAHOLD/SEADS# snooping mechanism, similar to the i486 microprocessor. The 82395DX is able to run a zero wait state 386 DX Microprocessor non-pipelined read cycle if the data exists in the cache. Memory write cycles can run with zero wait states if the write buffer is not full.

The 82395DX cache organization provides a higher hit rate than other standard configurations. The 82395DX, featuring the new high performance write buffer cache architecture, provides full concurrency between the electrically isolated Local Bus and System Bus. This allows the 82395DX to service read hit cycles on the Local Bus while running line fills or buffered write cycles on the System Bus. Moreover, the user has the option to expand his cache system up to 64KB.

1.2 Features

1.2.1 82385-LIKE FEATURES

- The 82395DX maps the entire physical address range of the 386 DX Microprocessor (4GB) into 16KB, 32KB, or 64KB cache (with one, two, or four 82395DXs respectively).
- Unified code and data cache.
- Cache attributes are handled by hardware. Thus the 82395DX is transparent to application software. This preserves the integrity of system software and protects the users software investment.
- Double Word, Word and Byte writes, Double Word reads.
- Zero wait states in read hits and in buffered write cycles. All 386 DX Microprocessor cycles are non- pipelined. (Note: The 386 DX Microprocessor must never be pipelined when used with the 82395DX - NA# must be tied to Vcc).
- A hardware cache FLUSH# option. The 82395DX will invalidate all the Tag Valid bits in the Cache Directory and clear the System Bus line buffer when FLUSH# is activated for a minimum of four CLK's. The line buffer is also FLUSH#ed.
- The 82395DX supports non-cacheable accesses. The 82395DX internally decodes the 387 DX Math Coprocessor accesses as Local Bus cycles.
- The system bus interface emulates a 386 DX Microprocessor interface.
- The 82395DX supports pipelined and non-pipelined system interface.
- Provides cache consistency (snooping): The 82395DX monitors the System Bus address via SEADS# and invalidates the cache address if the System Bus address matches a cached location.

1.2.2 NEW FEATURES

- 16KB on chip cache arranged in four banks, one bank for each way. In Read hit cycles, one DW is read. In a write hit cycle, any byte within the DW can be written. In cache fill cycle, the whole line (16B) is written. This large line size increases the hit rate over smaller line size caches.
- Cache architecture similar to the i486 Microprocessor cache: Four Way SET associative with Pseudo LRU replacement algorithm. Line size is 16B and a full line is retrieved from memory for every cache miss. Tag, Tag Valid Bit and Write Protect Bit are associated with every Line.
- New write buffer architecture with four DW deep write buffer provides zero wait state memory write cycles. I/O, Halt/Shutdown and LOCK#ed writes are not buffered.
- Concurrent Line Buffer Cacheing: The 82395DX has a line buffer that is used as additional memory. Before data gets written to the cache memory at the completion of a Line Fill it is stored in this buffer. Cache hit cycles to the line buffer can occur before the line is written to the cache.
- Expandable: two 82395DXs support 32KB cache memory, four 82395DXs support 64KB cache memory. This gives the user the option of configuring a system to meet their own performance requirements.
- In 387 DX Math Coprocessor accesses, the 82395DX drives the READYO# in one wait state if the READYI# was not driven in the previous clock.

  Note that the timing of the 82395's READYO# generation for 387 DX Math Coprocessor cycles is incompatible with 80287 timing.
- The 82395DX optionally decodes CPU accesses to Weitek 3167 Floating-Point Coprocessor address space (C0000000H–C1FFFFFFH) as Local Bus cycles. This option is enabled or disabled according to the LBA# pin value at the falling edge of RESET.

- An enhanced System Bus interface:
  a) Burst option is supported in line-fills similar to the i486 Microprocessor. SBRDY# (System Burst READY) is provided in addition to SRDY#. A burst is always a 16 byte cache update which is equivalent to four DW cycles. The i486 Microprocessor burst order is supported.
  b) System cacheability attribute is provided (SKEN#). SKEN# is used to determine whether the current cycle is cacheable. It is used to qualify Line Fill requests.
  c) SHOLD/SHLDA/SBREQ system bus arbitration mechanism is supported, the same as in the i486 Microprocessor. A Multi 386 DX/82395DX cluster can share the same System Bus via this mechanism.
  d) SNENE# output (Next Near) is provided to simplify the interface to DRAM controllers. DRAM page size of 2K is supported.
  e) Fast HOLD function (SFHOLD#) is provided. This function allows for multiprocessor support.
  f) Cache invalidation cycles supported via SEADS#. This is the mechanism used to provide cache coherency.
- Full Local Bus/System Bus concurrency is attained by:
  a) Servicing cache read hit cycles on the Local Bus while completing a Line Fill on the System Bus. The data requested by the 386 DX Microprocessor was provided over the local bus as the first part of the Line Fill.
  b) Servicing cache read hit cycles on the Local Bus while executing buffered write cycles on the system bus.
  c) Servicing cache read hit cycles on the Local Bus while another bus master is running (DMA, other 386 DX Microprocessor, 82395DX, i486 Microprocessor, etc . . .) on the System Bus.
  d) Buffering write cycles on the Local Bus while the system bus is executing other cycles.
- Write protected areas are supported by the SWP# input. This enables caching of ROM space or shadowed ROM space.
- No Post Input (NPI#) provided for disabling of write buffers per cycle. This option supports memory mapped I/O designs.
- A20M# input provided for emulation of 8086 address wrap-around.
- SRAM test mode, in which the TAGRAM and the cache RAM are treated as standard SRAM, is provided. A Tristate Output test mode is also provided for system debugging. In this mode the 82395DX is isolated from the other devices in the board by floating all its outputs.
- Single chip, 196 lead PQFP package, 1 micron CHMOS-IV technology.

2.0 82395DX CACHE SYSTEM DESCRIPTION

2.1 82395DX Cache Organization

The on chip cache memory is a unified code and data cache. The cache organization is 4 Way SET Associative and each Line is 16 bytes wide (see Figure 2.1). The 16K bytes of cache memory are logically organized as 4 4KB banks (4: 1 bank for each Way). Each bank contains 256 16B lines (256: 1 line for each SET).

The Cache Directory is used to determine whether the data in the cache memory is valid for the address being accessed. The Cache Directory contains 256 TAG's (each TAG is 22-bits wide) for each Way, for a total of 1K TAG's (See Figure 2.2). With each 20 bit TAG Address there is a TAG Valid Bit and a Write Protect bit. The Cache Directory also contains the LRU bits. The LRU bits are used to determine which Way to replace whenever the cache needs to be updated with a new line and all four ways contain data.

Table 2.1 lists the 82395DX cache organization.

Table 2.1 - 82395DX Cache Organization

Cache Element	82395DX Size/Qty	Comments
TAG	1K	Total number of TAGs
SET	256	Cache Directory Offset
LRU	256	3 bits per SET address
Way	4	4 TAG's per SET address
Line Size	16B	4 DW's
Sector Size	16B	4 DW's, one line per sector
Cache Size	16KB	Expandable to 64KB
Cache Directory	—	TAG address, TAG Valid Bit, and Write Protect Bit for each Way for each SET address (256 SET's × 4 Ways), and LRU bits.
TAG Valid Bit	1K	1 for each TAG in the cache directory, indicates valid data is in the cache memory.
Write Protect Bit	1K	1 for each TAG in the cache directory, indicates that the address is write protected.

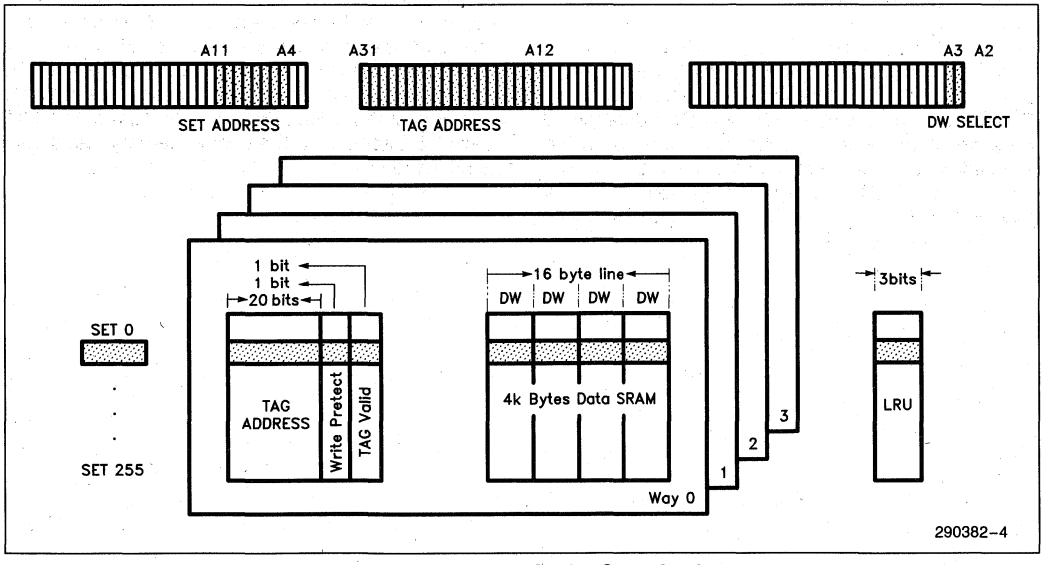


Figure 2.1 - 82395DX Cache Organization

5

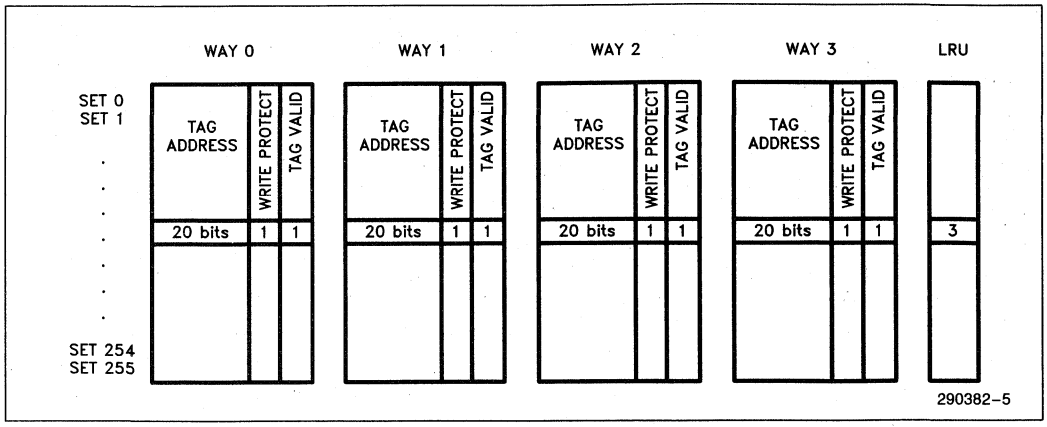


Figure 2.2 - 82395DX Cache Directory Organization

2.1.1 82395DX CACHE STRUCTURE AND TERMINOLOGY

A detailed description of the 82395DX cache parameters are defined here.

A **Line** is the basic unit of data transferred between the cache and main memory. In the 82395DX each Line is 16B. A Line is also known as a transfer block. The decision of a cache "hit or miss" is determined on a per Line basis. A cache hit results when the TAG address of the current address being accessed matches the TAG address in the Cache Directory (see Figure 2.3) and the TAG Valid bit is set. The 82395DX has 1K Lines.

A **TAG** is a storage element of the Cache Directory with which the hit/miss decision is made. The TAG consists of the TAG address (A31–A12), the TAG Valid bit and the Write Protect bit. Since many addresses map to a single line, the TAG is used to determine whether the data associated with the current address is present in the cache memory (a cache hit). This is done through a comparison of the TAG address bits of the current address and the contents of the Cache Directory, along with the TAG Valid bit. Each line in the cache memory has a TAG associated with it.

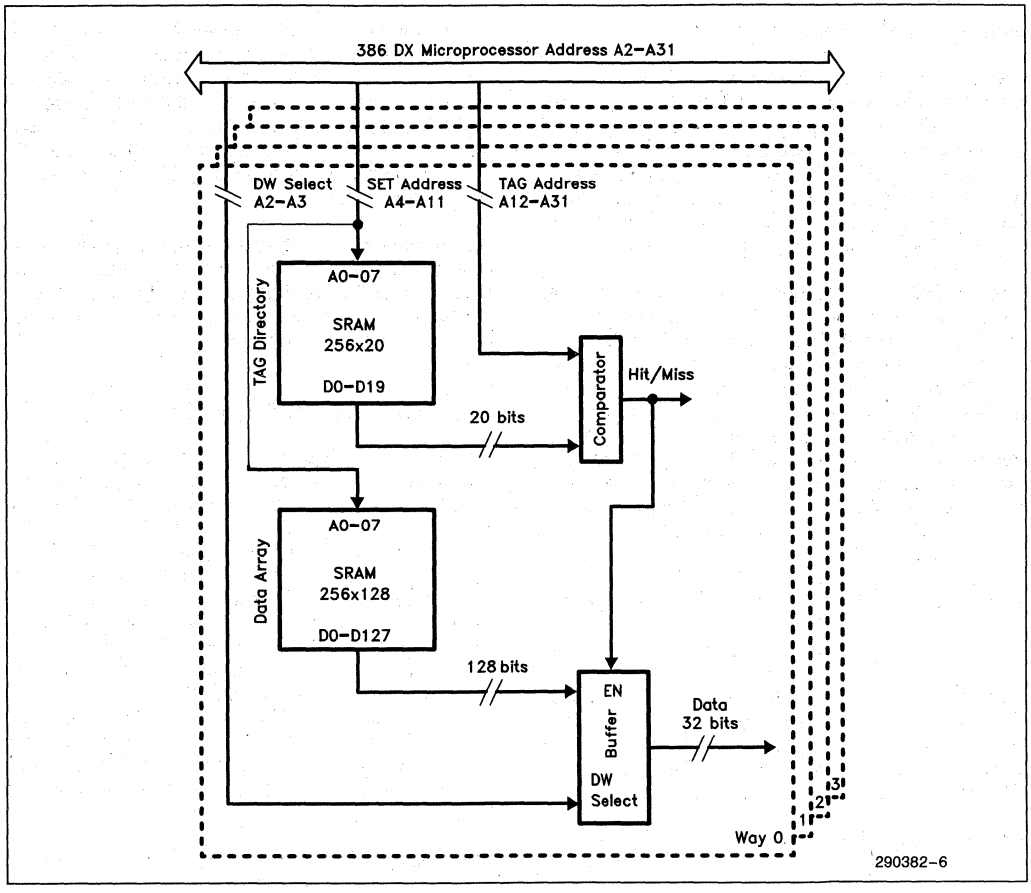


Figure 2.3 - 82395DX Cache Hit Logic

A **TAG Valid Bit** is associated with each TAG address in the Cache Directory. It determines if the data held in the cache memory for the particular TAG address is valid. It is used to determine whether the data in the cache is a match to data in main memory.

A **Write Protect Bit** is also associated with each TAG address in the Cache Directory. This field determines if the cache memory can be written to. It is set by the SWP# pin during Line Fill cycles (see chapter 6).

A **SET** address is a decoded portion of the Local Bus address that maps to 1 TAG address per Way in the Cache Directory. All the TAG's associated with a particular SET are simultaneously compared with the TAG field of the bus address to make the hit/miss decision. The 82395DX provides 256 SET addresses, each SET maps to four lines in the cache memory.

The term **Way** as in 4 Way SET Associative describes the degree of associativity of the cache system. Each Way provides TAG Address, TAG Valid bit, and Write Protect bit storage, 1 entry for each SET address. A simultaneous comparison of one TAG address from each Way with the bus address is done in order to make the hit/miss decision. The 82395DX is 4 Way SET Associative.

Other key 82395DX features include:

Cache Size - The 82395DX contains 16KB of cache memory. This can be expanded by connecting two or four 82395DX's in parallel to get up to 64KB of cache memory. Expanding the cache in this way results in an increased number of Tags with a constant number of lines per Tag. The cache is organized as four banks of 4KB. Each of the four banks corresponds to a particular Way.

Update Policy - The update policy deals with how main memory is updated when a cacheable write cycle is issued on the Local Bus. The 82395DX supports the write buffer policy, similar to the write through policy, which means that main memory is always updated in every write cycle. However, the cache is updated only when the write cycle hits the cache. Also, the 82395DX is able to cache write protected areas, e.g. ROMs, by preventing the cache update if the write cycle hits a write protected line. A write cycle to main memory is buffered as explained in chapter 6.

Replacement - When a new line is needed to update the cache, the Tag Valid bits are checked to see if any of the four ways are available. If they are all valid it is necessary to replace an old line that is already in the cache. In the 82395DX, the Pseudo LRU (least recently used) algorithm is adopted. The Pseudo LRU algorithm targets the least recently used line associated with the SET for replacement. (Pseudo LRU is described in section 2.2.).

Consistency - The 82395DX implements hooks for a consistency mechanism. This is to guarantee that in systems with multiple caches (and/or with multiple bus masters) all processor requests result in returning correct and consistent data. Whenever a system bus master performs memory accesses to data which also exists in the cache, the System Bus master can invalidate that entry in the 82395DX. This invalidation is done by using SEADS# (description in chapter 6).

The invalidation is performed by marking the TAG as invalid (the TAG Valid bit is cleared). Thus, the next time a Local Bus request is made to that location, the 82395DX accesses the main memory to get the most recent copy of the data.

2.2 Pseudo LRU Algorithm

When a line needs to be placed in the internal cache the 82395DX first checks to see if there is a non-valid line in the SET that can be replaced. The validity is checked by looking at the TAG Valid bit. The order that is used for this check is Way 0, Way 1, Way 2, and Way 3. If all four lines associated with the SET are valid, a pseudo Least Recently Used algorithm is used to determine which line will be replaced. If a non-valid line is found, that line is marked for replacement. All the TAG Valid bits are cleared when the 82395DX is RESET or when the cache is FLUSH#ed. Three bits, B0, B1, and B2, are defined for each of the 256 SETs. These bits are called the LRU bits and are stored in the cache directory. The LRU bits are updated for every access to the cache.

If the most recent access to the cache was to Way 0 or Way 1 then B0 is set to 1.

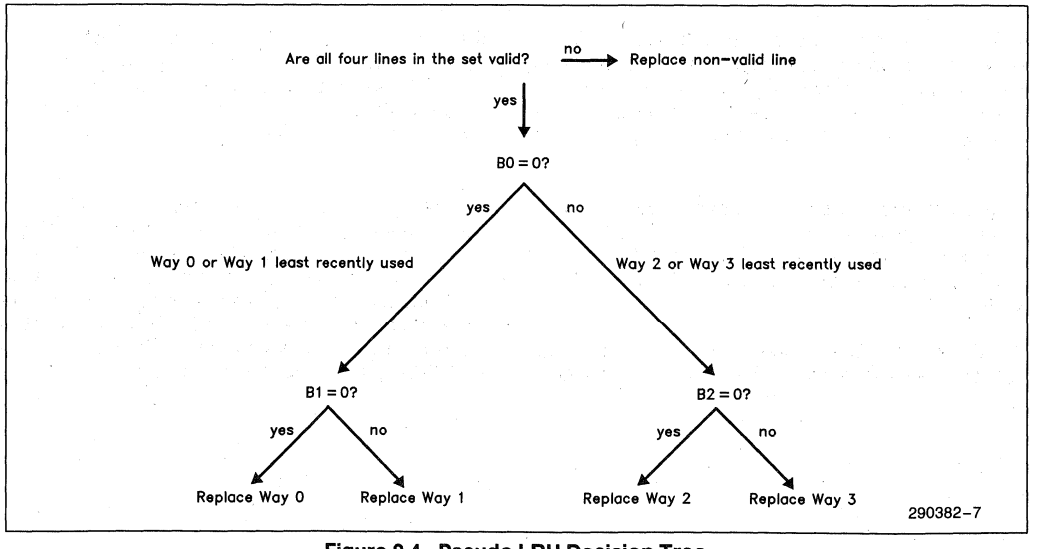


Figure 2.4 - Pseudo LRU Decision Tree

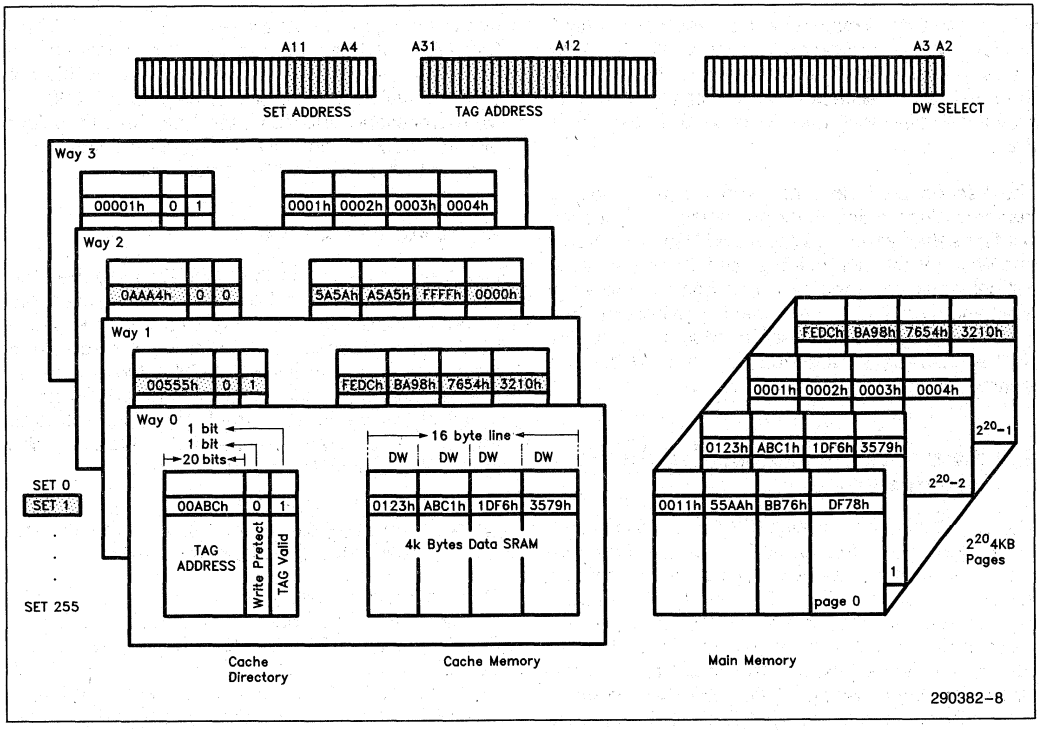


Figure 2.5 - Four Way Set Associative Cache Organization

5

B0 is set to 0 if the most recent access was to Way 2 or Way 3. If the most recent access to Way 0 or Way 1 was to Way 0, B1 is set to 1. Else B1 is set to 0. If the most recent access to Way 2 or Way 3 was to Way 2, B2 is set to 1. Else B2 is set to 0. See Table 2.2.

The Pseudo LRU algorithm works in the following manner. When a line must be replaced, the cache will first select which of Way 0 and Way 1 or Way 2 and Way 3 was least recently used. Then the cache will select which of the two lines was least recently used and mark it for replacement. The decision tree is shown in Figure 2.4. When the 82395DX is RESET or the cache is FLUSH#ed all the LRU bits are cleared along with the TAG Valid bits.

2.3 Four Way Set Associative Cache Organization

The 82395DX is a four Way SET Associative cache. Figure 2.5 shows the 82395DX's cache organization. For each of the 256 SET's there are four TAG's, one for each Way. The address currently being accessed is decoded into the SET and TAG addresses. If the access was to address 00555004h (SET=001,TAG=00555h), the four TAG's in the Cache Directory associated with SET 001 are simultaneously compared with the TAG of the address being accessed. The TAG Valid bits are also checked. If the TAG's match and the TAG Valid bit is set, the access is a hit to the Way where the hit was detected, in this example the hit occurred in Way 1. The data would be retrieved from Way 1 of the cache memory. If the next access was to address 0AAA4007h (SET=001, TAG=0AAA4h), the comparison would be done and a TAG match would be found in Way 2. However in this case the TAG Valid bit is cleared so the access is a miss and the data will be retrieved from main memory. The cache memory will also be updated. It is helpful to notice that the main memory is broken into pages by the TAG size. In this case with a 20-bit TAG address there are 2^{20} pages. The smaller the TAG size the fewer pages main memory is broken into. The SET breaks down these memory pages. The larger the SET size the more lines per page.

The following is a description of the interaction between the 386DX Microprocessor, the 82395DXs cache and Cache Directory.

2.3.1 CACHE READ HITS

When the 386 DX Microprocessor initiates a memory read cycle, the 82395DX uses the 8 bit SET address to select 1 of the 256 SET's in the Cache Directory. The four TAG's of this SET are simultaneously compared with address bits A12–A31. The four TAG Valid bits are checked. If any comparison produces a hit the corresponding bank of internal SRAM supplies the 32 bits of data to the 386 DX Microprocessor data bus based on the DW Select bits A2 and A3. The LRU bits are then updated according to the Pseudo LRU algorithm.

2.3.2 CACHE READ MISSES

Like the cache read hit the 82395DX uses the 8 bit SET address to select the 4 TAG's for comparison. If none of these match or if the TAG Valid bit associated with a matching TAG address is cleared the cycle is a miss and the 82395DX retrieves the requested data from main memory. A Line Fill is simultaneously started to read the line of data from system memory and write the line of data into the cache in the Way designated by the LRU bits.

2.3.3 OTHER OPERATIONS THAT AFFECT THE CACHE AND CACHE DIRECTORY

Other operations that affect the cache and Cache Directory include write hits, snoop hits, cache FLUSH#es and 82395DX RESETs. In write hits, the cache is updated along with main memory. The bank that detected the hit is the one that data is written to. The LRU bits are then adjusted according to the Pseudo LRU algorithm. When a cache invalidation cycle occurs (Snoop hit) the tag valid bit is cleared. RESETs and cache FLUSH#es clear all the TAG Valid bits.

2.4 Concurrent Line Buffer Cacheing

This feature of the 82395DX can be broken into two components, Concurrent Line Buffer and Line Buffer Cacheing.

A Concuurent Line Buffer indicates that the DW requested is returned to the 386 DX Microprocessor in the first cycle of a Line Fill. The Local Bus is then free to execute other cycles while the Line Fill is being completed on the System Bus.

Line Buffer Cacheing indicates that the 82395DX serves 386 DX Microprocessor cycles before it updates its Cache Directory. If the 386 DX Microprocessor cycle is to a line which resides in the cache memory, the 82395DX will serve that cycle as a regular cache hit cycle. The cache memory and cache directory are not updated until after the Line Fill is complete (see sections 2.8 and 2.9). The 82395DX keeps the address and data of the retrieved line in an internal buffer, the System Bus line buffer. Any 386 DX Microprocessor read cycle to the same line will be serviced from the line buffer. Until the cache memory and cache directory are updated, any

386 DX Microprocessor read cycle to a Doubleword, which has already been retrieved, will be serviced from the System Bus line buffer. On the other hand, any 386 DX Microprocessor write cycle to the same line will be done to the cache memory after updating the line in the cache. In this case, the write cycle is buffered and the READYO# is activated after updating the line in the cache. However, if the line is Write Protected, the write cycle will be handled as if it is a miss cycle.

A snooping cycle to a line which has not been updated in the cache will invalidate the SB Line Buffer and will prevent the cache update. Also, cache FLUSH will invalidate the buffer. More details about invalidation cycles can be found in chapter 6.

2.5 Cache Control

The cache can be controlled via the SWP# pin. By asserting this pin during the first DW in a Line Fill the 82395DX sets the write protect bit in the Cache Directory making the entry protected from writes.

2.6 Cache Invalidation

Cache invalidation cycles are activated using the SEADS# pin. SAHOLD or SHLDA asserted conditions the 82395DX's system address bus (SA4–SA31) to accept an input. The 82395DX floats its system address bus in the clock immediately after SAHOLD was asserted, or in the clock SHLDA is activated. No address hold acknowledge is required for SAHOLD. SEADS# asserted and the rising edge of CLK2 indicate that the address on the System Bus is valid. SEADS# is not conditioned by SAHOLD or SHLDA being asserted. The 82395DX will read the address and perform an internal cache invalidation cycle to the address indicated. The internal cache invalidation cycle is serviced 1 cycle after SEADS# was sampled active (or 2 cycles after SEADS# was sampled active if there is contention between the Cache Directory Snoop (CDS) cycle and a Cache Directory Lookup (CDL) cycle, see 2.8 and Figure 2.6). To actually invalidate the address the 82395DX clears the tag valid bit.

2.7 Cache Flushing

The user has an option of clearing the cache by activating the FLUSH# input. When sampling the FLUSH# input low for four clocks, the 82395DX resets all the tag valid bits and the LRU bits of the Cache Directory. Thus, all the banks of the cache are invalidated. Also, the SB Line Buffer is invalidated. The FLUSH# input must have at least eight CLK periods in order to be recognized. If FLUSH is activated for longer than four CLKs, the 82395DX will handle all accesses as misses and it will not update the Cache Directory (the Cache Directory will be FLUSH#ed as long as the FLUSH# input is low). The cache is also FLUSH#ed during RESET.

2.8 Cache Directory Accesses and Arbitration

There are five types of accesses to the cache directory. Each access is a one clock cycle:

1) Cache Directory Look-Up
2) Cache Directory Update
3) Cache Directory Snoop
4) Testability Accesses
5) Cache Directory FLUSH#

A description of each of these accesses follows:

1) **Cache Directory Look-up cycle (CDL):** A 386 DX Microprocessor access in which the hit/miss decision is made. The Cache Directory is accessed by the 386 DX Microprocessor address bus directly from the pins. CDL is executed whenever ADS# is activated, in both read and write cycles. The LRU bits are updated in every CDL hit cycle so the accessed "Way" becomes the most recently used. The LRU bits are read in every CDL miss cycle to indicate the "Way" to be updated in the Cache Directory Update cycle. Also, the WP bit is read.
2) **Cache Directory Update cycle (CDU):** A write cycle to the cache directory due to a previous miss. The CDU cycle can be caused by a TAG mismatch (either a Tag Address mismatch or a cleared TAG Valid bit). In both cases, the new TAG is written to the "Way" indicated by the LRU bits read by the previous CDL miss cycle. Also, the TAG Valid bit is turned on and the LRU algorithm is updated so the accessed "Way" becomes the most recently used. The WP bit is written according to the sampled SWP# input. The Cache Directory is accessed by the internally latched 386 DX Microprocessor address bus. Simultaneously with the CDU cycle, the cache memory is updated.
3) **Cache Directory Snooping cycle (CDS):** A Cache Directory look-up cycle initiated by the System Bus, in response to an access to a memory location that is shared with another system master, followed by a conditional invalidation of the TAG Valid bit. If the look-up cycle results in a hit, the corresponding TAG Valid bit in the Way which detected the HIT will be cleared. CDS cycles do not affect the LRU bits. The Cache Directory is accessed by the internally latched System Bus address.

4) **Testability accesses (CDT):** Cache Directory read and write cycles performed in SRAM test mode. During the TEST accesses, 25 bits of each entry (20 for the TAG, one for the TAG Valid BIT, one for the WP bit and 3 for the LRU bits) are read or written. No comparison is done. CDT cycles are used for debugging purposes so CDT cycles do not contend with other cycles.

5) **Cache Directory FLUSH cycle (CDF):** During RESET or as a result of a FLUSH# request generated by activating the FLUSH# input, all the TAG Valid bits and the LRU bits are cleared as well as the Line Buffer. CDF is a one clock cycle if FLUSH# is active for four clocks. If FLUSH# is activated longer, the CDF cycle is N−3 clocks, where N is the number of clocks FLUSH# is activated for. The actual clearing of the valid bits occurs seven clocks after the activation of FLUSH#. Two clocks are for internal synchronization and four for recognizing FLUSH# asserted. It has higher priority than all other cycles. CDF cycle may occur simultaneously with any other cycle but the result is always a FLUSH#ed Cache Directory.

The 82395DX performs the CDL cycle in T1 state. The CDU cycle, in general, is performed in the clock after the last SRDY# or SBRDY# of the Line Fill cycle and the CDS cycle one clock after sampling the SEADS# active (see more details on snooping cycles in chapter 6). Supporting concurrent activities on local and system busses causes CDL cycles to be requested in any clock during the execution with a maximum rate of a CDL cycle every other clock.

The following arbitration mechanism guarantees resolution of any possible contention between CDL, CDU and CDS cycles:

1. The priority order is CDL, CDS and CDU. CDL has the highest priority, CDU has the lowest.
2. In case of simultaneous CDL and CDS cycles, the CDS will be delayed by one clock. So, the maximum latency in executing the invalidation cycle is two clocks after sampling the SEADS# active. Since the maximum rate of each of the CDL and the CDS cycles is one every other clock, the 82395DX is able to interpose the CDL and CDS cycles such that both are serviced. Figure 2.6 clarifies the interposing in the Cache Directory between the 386 DX Microprocessor and the System Bus.
3. CDU cycle is executed in any clock after the last SRDY# or SBRDY# in which neither CDL nor CDS cycles are requested. The worst case is the case where immediately after the read miss, the 386 DX Microprocessor runs consecutive read hits while the System Bus is running invalidation cycles every other clock. In this case, the CDU cycle is postponed until a free clock is inserted, which may occur due to slower look-up rate (in case of read miss, non-cacheable read, etc...), or due to slower SEADS# rate.

   Since every CDU cycle is synchronized with the cache update (CU - writing the retrieved line into the cache), a possible contention on the cache can occur between a cache update cycle and a cache write cycle (CW - cache is written due to a write hit cycle). In this case, the CW cycle is executed, and the CDU and CU cycles are delayed.

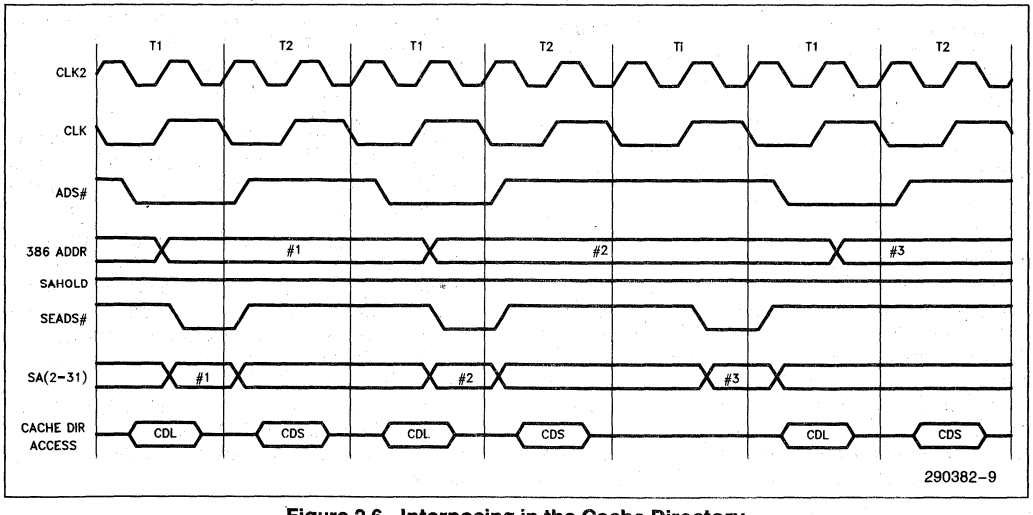


Figure 2.6 - Interposing in the Cache Directory

2.9 Cache Memory Description

The 82395DX cache memory is constructed of four banks, each bank is 1K double words (4KB) and represents a "Way". For example, if the read cycle is to Way 0, bank 0 will be read. The basic cache element is a Line. The cache is able to write a full line or any byte within the line. Reads are done by DW only.

There are four types of accesses to the cache data memory. Each access is a one clock cycle:

1) Cache Read cycle
2) Cache Write cycle
3) Cache Update cycle
4) Testability Access

A description of each type of access follows:

1) **Cache Read cycle (CR):** CR cycle occurs simultaneously with Cache Directory look-up (CDL) cycle if the cycle is a read. In case of a hit, the cache bank in which the hit was detected is read. In CR cycle, the A2–3 address lines select the requested DW within the line.
2) **Cache Write cycle (CW):** CW cycle occurs one clock after the Cache Directory look-up cycle (CDL) if the cycle is a write hit and the WP bit is not set. The cache bank in which the hit was detected is updated. In CW cycle, the A2–3 address lines and the four BE# lines select the required bytes within the line to be written. For all write hit cycles, READYO# is returned simultaneously with the CW cycle unless the write buffer is full. When the write buffer is full the first cycle buffered must be completed on the system bus before READYO# can be asserted.
3) **Cache Update cycle (CU):** CU cycle occurs simultaneously with every Cache Directory update cycle (CDU). The full line is written.
4) **Testability accesses (CT):** cache read and write cycles performed by the 82395DX TEST machine. During the TEST accesses, the cache memory acts as a standard RAM. CT cycles are used for debugging purposes so CT cycles do not contend with other cycles.

The Cache Directory arbitration rules guarantee that contention will not occur in the cache accesses. This is since CR is synchronized with the CDL cycle, CU is synchronized with CDU cycle, CW cannot occur simultaneously with CR cycles (ADS# not activated while READYO# is returned since 386 DX Microprocessor is not pipelined) and finally the possible contention of CW and CU is resolved. See figure 2.7 for an example of Cache Directory and cache memory accesses during a typical cycle execution.

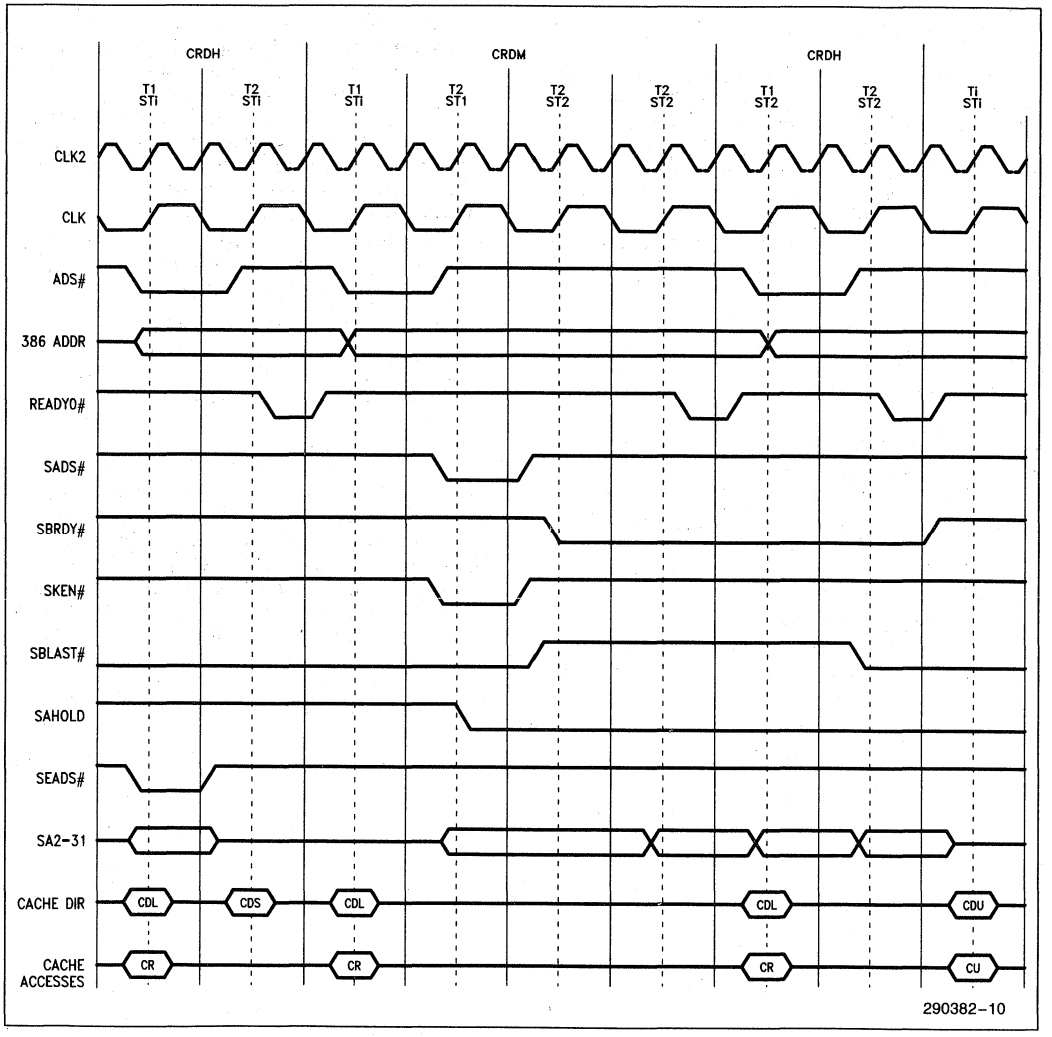


Figure 2.7 - Cache Directory and Cache Accesses

3.0 PIN DESCRIPTION

The 82395DX pins may be divided into 4 groups:

1. Local Bus interface pins
2. System Bus interface pins
3. Local Bus decode pins
4. System Bus decode pins

Some notes regarding these groups of pins follow:

1. All Pins - All input and I/O pins (when used as inputs) must be synchronous to CLK2, to guarantee proper operation. Exceptions are the RESET pin, where only the falling edge needs to be synchronous to CLK2, and A20M# and FLUSH# pin, which are asynchronous.
2. Local Bus Interface Pins - All Local Bus interface pins that have a corresponding 386DX Microprocessor signal (A2–31, W/R#, D/C#, M/IO#, LOCK#, and D0–31) must be connected directly to the corresponding 386 DX Microprocessor pins.
3. System Bus Interface Pins - In multi-82395DX mode, all System Bus output and I/O pins are driven by the primary 82395DX, with the exception of SADS#. See chapter 4 for more details.
4. Local / System Bus Decode Pins - These signals are generated by proper decoding of the Local and System Bus addresses. The decoding for the Local Bus decode pins, LBA# and NPI#, must be static. The decoding for the System Bus decode pins, SKEN# and SWP#, must be static over the line boundary. They must not change during a Line Fill. If a change in the decoding of these signals is made, the 82395DX must be FLUSH#ed or RESET.

3.1 Local Bus Interface Pins

3.1.1 386 DX MICROPROCESSOR/82395DX CLOCK (CLK2 I)

This signal provides the fundamental timing for the 82395DX. The 82395DX, like the 386 DX Microprocessor, divides CLK2 by two to generate the internal clock. The phase of the internal 82395DX clock is synchronized to the internal CPU clock phase by the RESET signal. All external timing parameters are specified with respect to CLK2.

3.1.2 LOCAL ADDRESS BUS

3.1.2.1 Local Bus Address Lines (A2–A31 I)

These signals, along with the byte enable signals, define the physical area of memory or I/O accessed.

3.1.2.2 Local Bus Byte Enables (BE3#–BE0# I)

These pins are used to determine which bytes are accessed in partial write cycles. On read-hit cycles these lines are ignored by the 82395DX. On write hit cycles they determine which bytes in the internal Cache SRAM must be updated, and passed to the System Bus along with the System Bus write cycle. In all system bus cycles (non-cacheable reads, read misses and all writes) these signals are mirrored by the SBE0-3# signals. These signals are active LOW.

3.1.3 LOCAL BUS CYCLE DEFINITION

3.1.3.1 Local Bus Cycle Definition Signals (W/R#,D/C#,M/IO# I)

The memory/input-output, data/code, write/read lines are the primary bus definition signals directly connected to the 386 DX Microprocessor. These signals become valid as the ADS# signal is sampled asserted. The bus cycle type encoding is identical to that of the 386 DX Microprocessor. The 386 DX Microprocessor encoding is shown in table 5.1. The bus definition signals are not driven by the 386 DX Microprocessor during bus hold and follow the timing of the address bus.

3.1.3.2 Local Bus Lock (LOCK# I)

This signal indicates a LOCK#ed cycle. LOCK#ed cycles are treated as non-cacheable cycles, except that LOCK#ed write hit cycles update the cache as well. LOCK#ed write cycles are not buffered.

The 82395DX asserts SLOCK# when the first LOCK#ed cycle is initiated on the System Bus. SLOCK# is deactivated only after all LOCK#ed System Bus cycles were executed, and LOCK# was deactivated.

3.1.4 LOCAL BUS CONTROL

3.1.4.1 Address Status (ADS# I)

The address status pin, an output of the 386 DX Microprocessor, indicates that new, valid address and cycle definition information is currently available on the Local Bus. The signals that are valid when ADS# is activated are:

A(2-31), BE(0-3)#, W/R#, D/C#, M/IO#, LOCK#, NPI# and LBA#

3.1.4.2 Local Bus Ready (READYI# I)

This is the ready input signal seen by the local bus master. Typically it is a logical OR between the 82395DX generated READYO# signal and other (optional) READY# signals generated by other Local Bus masters. It is used by the 82395DX, along with the ADS# signal, to keep track of the 386 DX Microprocessor bus state.

3.1.4.3 Local Bus Ready Output (READYO# I/O)

This output is returned to the 386 DX Microprocessor to terminate all types of 386 DX Microprocessor bus cycles, except for Local Bus cycles. This signal is wire-ORed with parallel 82395DX READYO# signals (if more than one 82395DX is used on a 386 DX Microprocessor bus). For more details on READYO# functionality in a multi-82395DX system, refer to Chapter 4.

The READYO# may serve as READY# signal for the 387 DX Math Coprocessor. For details, refer to Chapter 5.

This pin is used during the self configuration sequence, after RESET. For details, refer to Chapter 4.

3.1.5 RESET (RESET I)

This signal forces the 82395DX to begin execution at a known state. RESET falling edge is used by the 82395DX to set the phase of its internal clock identical to the 386 DX Microprocessor internal clock. The RESET falling edge must satisfy the appropriate set-up and hold times for proper chip operation. RESET must remain active for at least 1ms after power supply and CLK2 input have reached their proper DC and AC specifications.

The RESET input is used for three purposes: first, it RESETs the 82395DX and brings it to a known state. Second, it is used to synchronize the internal 82395DX clock phase to that of the 386 DX Microprocessor. Third, it initiates a self-configuration sequence in which the 82395DX determines the number of parallel 82395DX devices in the system and it's own configuration (Primary / Secondary and address space).

On power up, RESET must be active for at least 1 millisecond after power has stabilized to a voltage within spec, and after CLK2 input has stabilized to voltage and frequency within spec. This is to allow the internal circuitry to stabilize. Otherwise, RESET must be active for at least 10 clock cycles.

No access to the 82395DX is allowed for 128 clock cycles after the RESET falling edge. During RESET, all other input pins are ignored, except SHOLD, SAHOLD and SFHOLD#. Unlike the 386 DX Microprocessor, the 82395DX can respond to a System Bus HOLD request by floating its bus and asserting SHLDA even while RESET is asserted. Also the 82395DX can respond to a System Bus address HOLD request by floating its address bus. The status of the 82395DX outputs during RESET is shown in Table 3.2.

The 82395DX samples the LBA# pin during RESET and enables the decoding of Weitek 3167 Floating-Point Coprocessor address space if it is sampled low (active).

The user must make sure SAHOLD and FLUSH# are not asserted at the falling edge of RESET. If they are the Tristate Test Mode will be entered. The user must also insure that FLUSH# does not get asserted for one clock cycle while SAHOLD is negated for the same CLK cycle prior to RESET falling. If this condition exists a reserved mode will be entered.

3.1.6 CONFIGURATION (CONF# I)

The activity on this input during and after RESET allows the 82395DX to configure itself to operate in the specified address range.

Refer to Chapter 4 for more details. This pin is active LOW.

3.1.7 LOCAL DATA BUS

3.1.7.1 Local Bus Data Lines (D0–D31 I/O)

These are the Local Bus data lines of the 82395DX and must be connected to the D0–D31 signals of the Local Bus.

3.1.8 LOCAL BUS DECODE PINS

These signals are generated by proper decoding of the local bus address. The decoding of these signals must be static, the decoding must not change during normal operation of the 82395DX. If a change in the decoding of these signals is made, the 82395DX must be FLUSH#ed or RESET. These signals must be stable throughout the local bus cycle (refer to Figure 5.1).

3.1.8.1 Local Bus Access Indication (LBA# I)

This signal instructs the 82395DX that the cycle currently in progress is targeted to a Local Bus device, and must therefore be ignored by the 82395DX. The 387 DX Math Coprocessor is considered a Local

Bus Device, but LBA# need not be generated for 387 DX Math Coprocessor accesses. Weitek 3167 Floating-Point Coprocessor address space may also be decoded internally as Local Bus cycles. Note that LBA# has priority over all other types of cycles. This signal is active LOW.

3.1.8.2 No Post Input (NPI# I)

This signal instructs the 82395DX that the write cycle currently in progress must not be posted (buffered) in the write buffer. NPI# is sampled on the falling edge of CLK following the address change, see figure 5.1. NPI# is ignored during read cycles. This signal is active LOW.

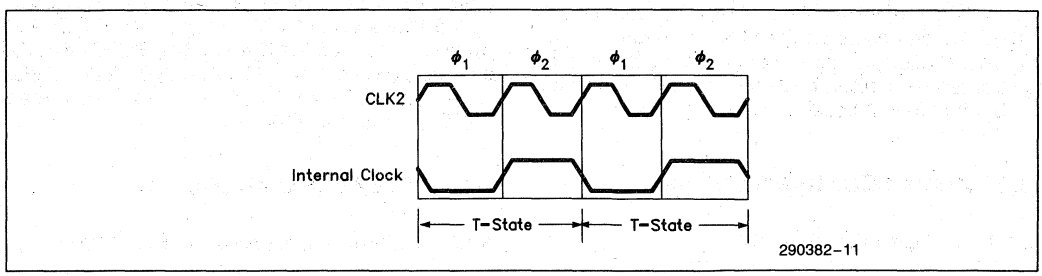


Figure 3.1 - CLK2 and Internal Clock

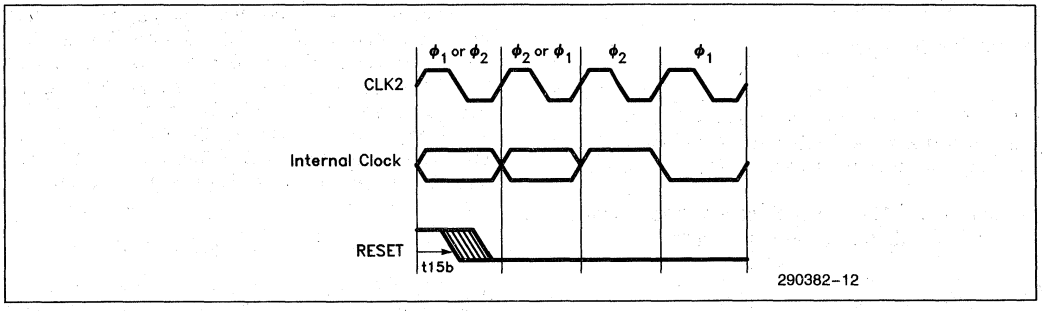


Figure 3.2 - RESET/Internal Phase Relationship

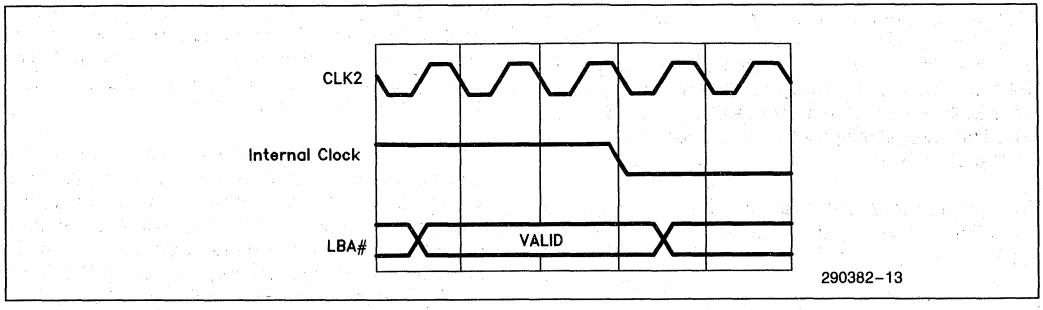


Figure 3.3 - Sampling LBA# During RESET

3.1.9 ADDRESS MASK

3.1.9.1 Address Bit 20 Mask (A20M# I)

This pin, when active (low), forces the A20 input as seen by the 82395DX to logic '0', regardless of the actual value on the A20 input pin. It must be asserted two clock cycles before ADS# for proper operation. A20M# emulates the address wraparound at 1 MByte which occurs on the 8086. This pin is asynchronous but must meet setup and hold times to guarantee recognition in a specific clock. It must be stable throughout Local Bus memory cycles.

3.2 System Bus Interface Pins

3.2.1 SYSTEM ADDRESS BUS

3.2.1.1 System Bus Address Lines (SA2–SA31 I/O *)

* SA2-3 are outputs only.

These are the SYSTEM BUS address lines of the 82395DX. When driven by the 82395DX, these signals, along with the System Bus byte enables define the physical area of memory or I/O accessed.

Activation of SEADS# conditions these signals to serve as inputs for the snooping cycle.

3.2.1.2 System Bus Byte Enables (SB0#–SB3# O)

These are the byte enable signals for the System Bus. The 82395DX drives these pins identically to BE0#–BE3# in all System Bus cycles except Line Fills. In Line Fills these signals are driven identically to BE0#–BE3# in the first read cycle of the Line Fill. They are all driven active in the remaining cycles of the Line Fill.

The system memory must ignore these pins during Line Fill, and return all four bytes. These signals are active low.

3.2.2 SYSTEM BUS CYCLE DEFINITION

3.2.2.1 System Bus Cycle Definition (SW/R#,SD/C#,SM/IO# O)

These are the System Bus cycle definition pins. When the 82395DX is the SYSTEM BUS master, it drives these signals identically to the 386 DX Microprocessor encoding.

3.2.2.2 System Bus Lock (SLOCK# O)

The SYSTEM BUS LOCK pin is one of the bus cycle definition pins. It indicates that the current bus cycle is LOCK#ed: that the 82395DX (on behalf of the CPU) must be allowed exclusive access to the System Bus across bus cycle boundaries until this signal is de-asserted. The 82395DX does not acknowledge a bus hold request when this signal is asserted. The 82395DX asserts SLOCK# when the first LOCK#ed cycle is initiated on the System Bus; SLOCK# is deactivated only after all LOCK#ed System Bus cycles were executed, and LOCK# was deactivated. SLOCK# is active LOW.

3.2.3 SYSTEM BUS CONTROL

3.2.3.1 System Bus Address Status (SADS# O)

The address status pin is used to indicate that new, valid address and cycle definition information is currently being driven onto the address, byte enables and cycle definition lines of the System Bus. SADS# can be used as an indication of a new cycle start. SADS# is driven active in the same clock as the addresses are driven. SADS# is not valid until a specified setup time before the CLK falling edge, and must be sampled by CLK falling edge before it is used by the system. This signal is active LOW.

3.2.3.2 System Bus Ready (SRDY# I)

The SRDY# signal indicates that the current bus cycle is complete. When SRDY# is sampled asserted it indicates that the external system has presented valid data on the data pins in response to a read cycle or that the external system has accepted the 82395DX data in response to a write request. This signal is ignored when the SYSTEM BUS is at STi, STH, ST1 or ST1P states.

At the first read cycle of a Line Fill, if SBRDY# is returned active and both SRDY# and SNA# are returned inactive, a burst Line Fill will be executed. If SRDY# is returned active and SNA# is returned inactive, a non-burst non-pipelined Line Fill will be executed. If SNA# is returned active and SRDY# is inactive, a non-burst pipelined line fill will be executed.

Once a burst Line Fill has started, if SRDY# is returned in the second or third DW of the transfer, the burst Line Fill will be interrupted and the cache will not be updated. The first DW will already have been transferred to the CPU. Note that in the last (fourth) bus cycle in a Line Fill, SBRDY# and SRDY# have the same effect on the 82395DX. They indicate the end of the Line Fill. This signal is active LOW.

3.2.3.3 System Bus Next Address (SNA# I)

This input, when active, indicates that a pipelined address cycle can be executed. It is sampled by the 82395DX in the same timing as the 386 DX Microprocessor samples NA#. If this signal is sampled active, then SBRDY# is treated as SRDY#, i.e. burst Line Fill is disabled. This signal is ignored once a burst Line Fill has started, as well as during the fourth DW of a Line Fill.

3.2.4 BUS ARBITRATION

3.2.4.1 System Bus Request (SBREQ O)

SBREQ is the internal cycle pending signal. This indicates to the outside world that internally the 82395DX has generated a bus request (due to a CPU's request that requires access to the System Bus). It is generated whether the 82395DX owns the bus or not and can be used to arbitrate among the various masters on the system bus. In read misses, if the bus is available and the cycle starts immediately, this signal will not be activated at all. This signal is active HIGH.

3.2.4.2 System Bus Hold Request (SHOLD I)

This signal allows another bus master complete control of the entire System Bus. In response to this pin, the 82395DX floats all its system bus interface output and input/output pins (With the exception of SHLDA and SBREQ) and asserts SHLDA after completing its current bus cycle or sequence of LOCK#ed cycles. The 82395DX maintains its bus in this state until SHOLD is deasserted. SHOLD is active HIGH. SHOLD is recognized during reset.

3.2.4.3 System Bus Hold Acknowledge (SHLDA O)

This signal goes active in response to a hold request presented on the SHOLD pin and indicates that the 82395DX has given the bus to another System Bus master. It is driven active in the same clock that the 82395DX floats its bus. When leaving a bus hold, SHLDA is driven inactive in one clock and the 82395DX resumes driving the bus. Depending on internal requests the 82395DX may, or may not begin a System Bus cycle in the clock where SHLDA is driven inactive. The 82395DX is able to support CPU Local Bus activities during System Bus hold, since the internal cache is able to satisfy the majority of those requests. This signal is active HIGH.

3.2.4.4 System Bus Fast Hold Request (SFHOLD# I)

This input allows another bus master immediate access to the System Bus. In response to this signal, the 82395DX stops driving the System Bus output and input/output pins (with the exception of SHLDA and SBREQ) in the next CLK cycle. Note that the same signals are tristated in response to a SHOLD request. Because the 82395DX always stops driving the System Bus in response to SFHOLD# active, no acknowledge is needed.

The bus remains in the high impedance state until SFHOLD# is negated.

Note that SRDY# is internally inactivated during SFHOLD# cycles. The only affect of SFHOLD# being asserted is forcing the System Bus output and I/O buffers into their high impedance state. It is the responsibility of the system designer to guarantee that bus cycles which are in progress when SFHOLD# is asserted are terminated correctly.

This pin is recognized during RESET and is active low.

3.2.5 BURST CONTROL

3.2.5.1 System Bus Burst Ready (SBRDY# I)

This signal performs the same function during a burst cycle that SRDY# does in a non-burst cycle. SBRDY# asserted indicates that the external system has presented valid data on the data pins in response to a burst Line Fill cycle. This signal is ignored when the SYSTEM BUS is at STi, STH, ST1 or ST1P states.

Note that in the last (fourth) bus cycle in a Line Fill, SBRDY# and SRDY# have the same effect on the 82395DX. They indicate the end of the Line Fill. For all cycles that cannot run in burst, e.g. noncacheable cycles, non Line Fill cycles (or pipelined Line Fill), SBRDY# has the same effect on the 82395DX as the normal SRDY# pin. This signal is active LOW.

3.2.5.2 System Bus Burst Last Cycle Indicator (SBLAST# O)

The system burst last cycle signal indicates that the next time SBRDY# is returned the burst transfer is complete. In other words, it indicates to the external system that the next SBRDY# returned is treated as a normal SRDY# by the 82395DX, i.e., another set of addresses will be driven with SADS# or the System Bus will go idle. SBLAST# is normally active.

In a cache read miss cycle, which may proceed as a Line Fill, SBLAST# starts active and later follows SKEN# by one clock. SBLAST# is active during non-burst Line Fill cycles. Refer to Chapter 6 for more details. This signal is active LOW.

3.2.6 CACHE INVALIDATION

3.2.6.1 System Bus Address Hold (SAHOLD I)

This is the Address Hold request. It allows another bus master access to the address bus of the 82395DX in order to indicate the address of an external cycle for performing an internal Cache Directory lookup and invalidation cycle. In response to this signal, the 82395DX immediately (in the next cycle) stops driving the entire system address bus (SA2–SA31). Because the 82395DX always stops driving the address bus, in response to system bus address hold request, no hold acknowledge is required. Only the address bus will be floated during address hold, other signals can remain active. For example, data can be returned for a previously specified bus cycle during address hold. The 82395DX does not initiate another bus cycle during address hold.

This pin is recognized during RESET. However, since the entire cache is invalidated by reset, any invalidation cycles run will be superfluous. This signal is active high.

3.2.6.2 System Bus External Address Strobe (SEADS# I)

This signal indicates that a valid external address has been driven onto the 82395DX pins and that this address must be used to perform an internal cache invalidation cycle. Maximum allowed invalidation cycle rate is one every two clock cycles. This signal is active low.

3.2.7 CACHE CONTROL

3.2.7.1 Flush (FLUSH# I)

This pin, when sampled active for four clock cycles or more, causes the 82395DX to invalidate its entire Tag Array. In addition, it is used to configure the 82395DX to enter various test modes. For details refer to Chapter 7. This pin is asynchronous but must meet setup and hold times to guarantee recognition in any specific clock. This signal is active LOW.

3.2.8 SYSTEM DATA BUS

3.2.8.1 System Bus Data Lines (SD0–SD31 I/O)

These are the System Bus data lines of the 82395DX. The lines must be driven with appropriate setup and hold times for proper operation. These signals are driven by the 82395DX only during write cycles.

3.2.9 SYSTEM BUS DECODE PINS

3.2.9.1 System Cacheability Enable (SKEN# I)

This is the cache enable pin. It is used to determine whether the current cycle running on the System Bus is cacheable or not. When the 82395DX generates a read cycle that may be cached, this pin is sampled 1 CLK before the first SBRDY#, SRDY# or SNA# is sampled active (for detailed timing description, refer to Chapter 6). If sampled active, the cycle will be transformed into a Line Fill. Otherwise, the Cache and Cache Directory will be unaffected. Note that SKEN# is ignored after the first cycle in a Line Fill. SKEN# is ignored during all System Bus cycles except for cacheable read miss cycles. This signal is active LOW.

3.2.9.2 System Write Protect Indication (SWP# I)

This is the write protect indicator pin. It is used to determine whether the address of the current system bus Line Fill cycle is write protected or not.

In non-pipelined cycles, the SWP# is sampled with the first SRDY# or SBRDY# of a system Line Fill cycle. In pipelined cycles, SWP# is sampled at the last ST2 stage, or at ST1P; in other words, one clock phase after SNA# is sampled active.

The write protect indicator is sampled together with the TAG address of each line in the 82395DX Cache Directory. In every cacheable write cycle, the write protect indicator is read. If active, the cycle will be a Write Protected cycle which is treated like a cacheable write miss cycle. It is buffered and it does not update the cache even if the addressed location is present in the cache. The signal is active LOW.

3.2.10 DESIGN AIDES

3.2.10.1 System Next Near Indication (SNENE# O)

This signal indicates that the current System Bus memory cycle is to the same 2048 Byte area as the

previous memory cycle. Address lines A11–A31 of the current System Bus memory cycle are identical to the address lines A11–A31 of the previous memory cycle.

This signal can be used in an external DRAM system to run CAS# only cycles, therefore increasing the throughput of the memory system. SNENE# is valid for all memory cycles, and indicates that the current memory cycle is to the same 2048 Byte area, even if there were idle or non-memory bus cycles since the last System Bus memory cycle.

For the first memory cycle after the 82395DX has exited the HOLD state, or after SAHOLD was deactivated, this pin will be inactive. This signal is active low.

3.3 Pinout Summary Tables

Table 3.1 - Input Pins

Name	Function	Synchronous/ Asynchronous	Active Level
CLK2	Clock		
RESET	Reset	Asynchronous*	High
BE0–3#	Local Bus Byte Enables	Synchronous	Low
A2–31	Local Bus Address Lines	Synchronous	—
W/R#	Local Bus Write/Read	Synchronous	—
D/C#	Local Bus Data/Control	Synchronous	—
M/IO#	Local Bus Memory/Input-Output	Synchronous	—
LOCK#	Local Bus LOCK	Synchronous	Low
ADS#	Local Bus Address Strobe	Synchronous	Low
READYI#	Local Bus READY	Synchronous	Low
LBA#	Local Bus Access Indication	Synchronous	Low
NPI#	No Post Input	Synchronous	Low
FLUSH#	FLUSH the 82395DX Cache	Asynchronous	Low
A20M#	Address Bit 20 Mask	Asynchronous	Low
CONF#	Configuration	Synchronous	Low
SHOLD	System Bus Hold Request	Synchronous	High
SRDY#	System Bus READY	Synchronous	Low
SNA#	System Bus Next Address Indication	Synchronous	Low
SBRDY#	System Bus Burst Ready	Synchronous	Low
SKEN#	System Cacheability Indication	Synchronous	Low
SWP#	System Write Protect Indication	Synchronous	Low
SAHOLD	System Bus Address HOLD	Synchronous	High
SEADS#	System Bus External Address Strobe	Synchronous	Low
SFHOLD#	System Bus Fast HOLD Request	Synchronous	Low

* The falling edge of RESET needs to be synchronous to CLK2 but the rising edge is asynchronous.

Table 3.2 - Output Pins

Name	Function	When Floated	State at RESET	Active Level
SBE0-3#	System Bus Byte Enables	SHLDA/SFHOLD#	Low	Low
SADS#	System Bus Address Strobe (1)	SHLDA/SFHOLD#	High	Low
SD/C#	System Bus Data/Control	SHLDA/SFHOLD#	High	—
SM/IO#	System Bus Memory/Input-Output	SHLDA/SFHOLD#	Low	—
SW/R#	System Bus Write/Read	SHLDA/SFHOLD#	Low	—
SHLDA	System Bus HOLD Acknowledge	—	Low (2)	High
SBREQ	System Bus Request	—	Low	High
SLOCK#	System Bus LOCK	SHLDA/SFHOLD#	High	Low
SBLAST#	System Bus Burst Last Cycle Indication	SHLDA/SFHOLD#	Low	Low
SA2-3	System Bus Address (2 lowest order bits)	SHLDA/SAHOLD/ SFHOLD#	High	—
SNENE#	System Bus Next Near Indication	SHLDA/SFHOLD#	High	Low

NOTES:
1. SADS# is driven active in ST1/ST2P and inactive for one phase in the first ST2/ST1P following the activation. SADS# is driven high before it is floated.
2. Unless SHOLD is asserted

Table 3.3 - Input-Output Pins

Name	Function	When Floated	State(1) at RESET	Active Level
D0-31	Local Data Bus (2)	Always Except READs	z	—
SD0-31	System Data Bus	Always Except WRITEs	z	—
SA4-31	System Bus Address (except the 2 lowest order bits)	SHLDA/SAHOLD/ SFHOLD#	High	—
READYO#	Local Bus READY	See Sec 4.6	High	Low

(1) Provided SHOLD, SAHOLD, and SFHOLD# are inactive
(2) Local Data is driven only in TZ.

4.0 BASIC FUNCTIONAL DESCRIPTION

The 82395DX has an interface to the 386 DX Microprocessor (Local Bus) and to the System Bus. The System Bus interface emulates the 386 DX Microprocessor bus such that the system will view the 82395DX as the front end of a 386 DX Microprocessor. Some optional enhancements, like burst support, are provided to maximize the performance.

When ADS# is sampled active, the 82395DX decodes the 386 DX Microprocessor cycle definition signals (M/IO#, D/C#, W/R# and LOCK#), as well as two Local Bus decode signals (LBA# and NPI#), to determine how to respond. LBA# indicates that the current cycle is addressed to a Local Bus device; NPI# indicates that the current memory write cycle must not be buffered. In addition, the 82395DX internally decodes the 386 DX Microprocessor accesses to the 387 DX Math Coprocessor / Weitek 3167 Floating-Point Coprocessor as Local Bus accesses. The result of the address, cycle definition and cycle qualification decoding is two categories of accesses, the Local Bus accesses (LBA# active or 387 DX Math Coprocessor / Weitek 3167 Floating-Point Coprocessor accesses) and 82395DX accesses. In 387 DX Math Coprocessor accesses, the 82395DX drives the READYO# signal active after one wait state, if the READYI# was not sampled active. Local Bus accesses are ignored by the 82395DX.

Any 82395DX access can be either to a cacheable address or to a non-cacheable address. Non-cacheable addresses are all I/O and system accesses with SKEN# returned inactive. Non-cacheable cycles are all cycles to non-cacheable addresses, LOCK#ed read cycles and Halt/Shutdown cycles. All other cycles are cacheable. For more details about non-cacheable cycles, refer to section 4.2. Non-cacheable cycles pass through the cache. They are always forwarded to the System Bus.

Cacheable read cycles can be either hit or miss. Cacheable read hit cycles are serviced by the internal cache and they don't require System Bus service. A cacheable read miss cycle generates a series of four System Bus read cycles, called a Line Fill. Of

the four cycles, the first cycle is for reading the requested data while all four are for filling the cache line. The System Bus has the ability to provide the system cacheability attribute to the 82395DX Line Fill request, via the SKEN# input, and the system write protection indicator, via the SWP# input. Refer to chapter 6 for more information about Line Fill cycles.

Cacheable write cycles, as any write cycles, are forwarded to the system bus. The write buffer algorithm terminates the write cycle on the Local Bus, allowing the 386 DX Microprocessor to continue processing in 0 wait states, while the 82395DX executes the write cycles on the System Bus. All cacheable write hit cycles, except protected writes, update the cache in a byte basis i.e. only the selected bytes are updated. Cacheable write misses do not update the cache (the 82395DX does not allocate on writes). All cacheable write cycles, except LOCK#ed writes, are buffered (unless NPI# pin is sampled active).

Cache consistency is provided by the SAHOLD, SEADS# mechanism. If any bus master performs a memory cycle which disturbs the data consistency, the address of this cycle must be provided to the 82395DX using the SAHOLD, SEADS# mechanism. Then, the 82395DX checks if that memory location resides in the cache. If it does, the 82395DX invalidates that line in the cache by marking it as invalid in the Cache Directory. The 82395DX interposes the Cache Directory between the 386DX Microprocessor and the System Bus such that the 386 DX Microprocessor is not forced to wait due to snooping and none of the snooping cycles are missed due to 386 DX Microprocessor accesses (see figure 2.6). Cacheability is resolved on the system side using the SKEN# input. SKEN# is sampled one clock before the first SRDY#/SBRDY# in nonpipelined Line Fill cycles. In pipelined Line Fill cycles, SKEN# is sampled one clock phase before sampling SNA# active. SKEN# is always sampled at PHI1.

Note that the 82395DX does not support pipelining of the 386DX Microprocessor Local Bus. The NA# input on the 386 DX Microprocessor must be tied to Vcc.

4.1 Cacheable Accesses

In a cacheable access, the 82395DX performs a cache directory look-up cycle. This is to determine if the requested data exists in the cache and to read the write protection bit. In parallel, the 82395DX performs a cache read cycle if the access is a read, or prepares the cache for a write cycle if the access is a write.

4.1.1 CACHEABLE READ HIT ACCESSES

If the Cache Directory look-up for a cacheable read access results in a hit (the requested data exists in the cache), the 82395DX drives the local data bus by the data provided from the internal cache. It also drives the 386DX Microprocessor READY# (by activating the 82395DX READYO#), so that the 386 DX Microprocessor gets the required data directly from the cache without any wait states.

The 82395DX is a four Way SET associative cache, so only one of the four ways (four banks) is selected to supply data to the 386 DX Microprocessor. The Way in which the hit occurred will provide the data. Also, the replacement algorithm (LRU) is updated such that the Way in which the hit occurred is marked as the most recently used.

4.1.2 CACHEABLE READ MISS ACCESSES

If the Cache Directory look-up results in a miss, the 82395DX transfers the request to the System Bus in order to read the data from the main memory and for updating the cache. A full line is updated in cache update cycle. As a result of a cache miss, the 82395DX performs four System Bus accesses to read four DWs from the DRAM, and write the four DWs to the cache. This is called a Line Fill cycle. The first DW accessed in a Line Fill cycle is for the DW which the 386 DX Microprocessor requested and the 82395DX provides the data and drives the READYO# one clock after it gets the first DW from the SB.

The 82395DX provides the option of supporting burst bus in order to minimize the latency of a line fill. Also, the 82395DX provides the SKEN# input, which, if inactive, converts a Line Fill cycle to a non-cacheable cycle. Write protection indicator is stored together with the TAG Valid bit and the TAG field of every line in the Cache Directory. For more details refer to chapter 6.

The 82395DX features Line Buffer cacheing. In a Line Fill the data for the four DWs is stored in a buffer, the Line Buffer, as it is accumulated. After filling the Line Buffer, the 82395DX performs the Cache Update and the Cache Directory Update. The updated Way is the least recently used Way flagged by the Pseudo LRU algorithm during the Cache Directory Lookup cycle, if all the Ways are valid. If there is a non-valid Way it will be updated.

The SRDY# (System Bus READY#) active indicates the completion of the system bus cycle and SBRDY# (System Bus Burst READY#) active indicates the completion of a burst System Bus cycle. In a 386 DXMicroprocessor-like system, the 82395DX

drives the 386 DX Microprocessor READY# one clock after the first SRDY# and, in a burst system, one clock after the first SBRDY#. This frees up the Local Bus, allowing the 386 DX Microprocessor to execute the next instruction, while filling the cache.

So, during Line Fills, there is no advantage in driving the 386 DX Microprocessor into the pipelined mode. **Therefore, the 82395DX does not drive the 386 DX Microprocessor's NA# at all. NA# must be tied to VCC.**

4.1.2.1 Burst Bus

The 82395DX offers an option to minimize the latency in Line Fills. This option is the burst bus and is only applicable to Line Fill cycles. By generation of a burst bus compatible DRAM controller, one which generates SBRDY# and SBLAST# to take advantage of the 82395DX's burst feature, the number of cycles required for a Line Fill to be completed is significantly reduced. Details of burst Line Fills can be found in chapter 6. The burst feature uses the i486 Microprocessor burst order to fill the 16 byte cache line (see Table 6.1).

4.1.3 CACHE WRITE ACCESSES

The 82395DX supports the write buffer policy, which means that main memory is always updated in any write cycle. However, the cache is updated only when the write cycle hits the cache and the accessed address is not write protected. In cache write misses, the cache is not updated (allocation in writes is not supported).

The 82395DX has a write buffer of four DWs. Only the cacheable write cycles, except LOCKed writes, are buffered so, if the write buffer is not full, the 82395DX buffers the cycle. This means that the data, address and cycle definition signals are written in one entry of the write buffer and the 82395DX drives the READYO# in the first T2 so all the buffered write cycles run without wait states. If the write buffer is full, the 82395DX delays the READYO# until the completion of the execution of the first buffered write cycle. The execution of the buffered write cycles depends on the availability of the System Bus. In non-buffered write cycles, e.g. I/O writes, the 386 DX Microprocessor is forced to wait until the execution of all the buffered writes and the non-buffered write (READYO# is driven one clock after the SRDY# of the non-buffered write). More details about the write buffer can be found in chapter 6.

In cacheable non-write protected write hit cycles, only the appropriate bytes within the line are updated. The updated bytes are selected by decoding the A2, A3 and the four BE# lines. The LRU is updated so that the hit Way is the most recently used, as in cache read hit cycles.

All cacheable writes, whether hits or misses, are executed on the system bus. The System Bus write cycle address, data and cycle definition signals are the same as the 386 DX Microprocessor signals. All buffered writes run with zero wait states if the write buffer is not full.

4.2 Noncacheable System Bus Accesses

Non-cacheable cycles are any of the following 82395DX cycles:

1) All I/O cycles.
2) All LOCKed read cycles.
3) Halt/Shutdown cycles.
4) SRAM mode cycles not addressing the internal cache or Tagram.

All the above cycles are defined as non-cacheable by the Local Bus interface controller. In addition, Line Fill cycles in which the SKEN# signal was returned inactive are aborted. They are called Aborted Line Fills (ALF).

Table 4.1 - 386 DX Microprocessor Bus Cycle Definition with Cacheability

M/IO#	D/C#	W/R#	386 DX Microprocessor Cycle Definition	Cacheable/ Non-cacheable	Writes Posted
0	0	0	Interrupt Acknowledge	Non-cacheable	—
0	0	1	Undefined	—	—
0	1	0	I/O Read	Non-cacheable	—
0	1	1	I/O Write	Non-cacheable	No
1	0	0	Memory Code Read	Cacheable	—
1	0	1	Halt/Shutdown	Non-cacheable	—
1	1	0	Memory Data Read	Cacheable	—
1	1	1	Memory Data Write	Cacheable	Yes

Non-cacheable cycles are never serviced from the cache and they don't update the cache. They are always referred to the System Bus. In non-cacheable cycles, the 82395DX transfers to the System Bus the exact 386 DX Microprocessor bus cycle. All non-cacheable write cycles are not buffered.

Description of LOCKed cycles can be found in chapter 5.

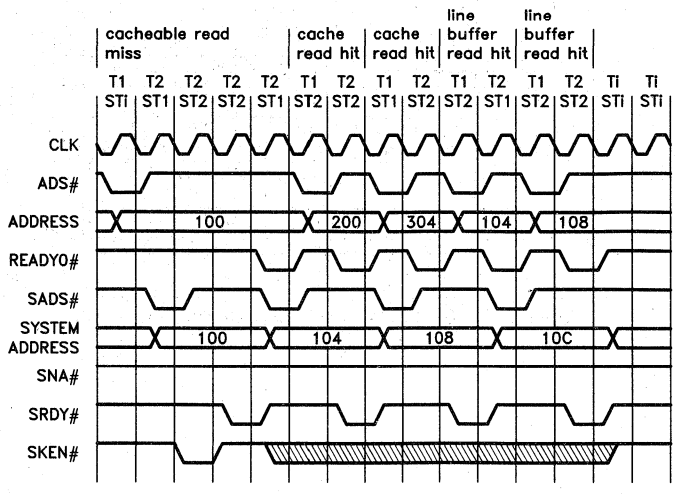
4.3 Local and System Bus Concurrency

Concurrency between local and System Busses is supported in several cases:

1. Read hit cycles can run while executing a Line Fill on the System Bus. Refer to timing diagram 4.1.
2. Read hit cycles can run while executing buffered write cycles on System Bus. Refer to timing diagram 4.2.
3. Write cycles are buffered while the System Bus is running other cycles, including other buffered writes. They are also buffered when another bus master is using the System Bus (e.g. DMA, other CPU). Refer to timing diagram 4.3.
4. Read hit cycles can run while another System Bus master is using the System Bus.

The first case is established by providing the data which the 386 DX Microprocessor requested first and later the 82395DX continues filling its line while it is servicing new cache read hit cycles. The 82395DX updates its cache and cache directory after completing the System Bus Line Fill cycle. Meanwhile, any 386 DX Microprocessor read cycles will be serviced from the cache if they hit the cache. In case the 386 DX Microprocessor read cycles are consecutive such that the 386 DX Microprocessor is requesting a double-word which belongs to the same line currently retrieved by the System Bus Line Fill cycle and the requested DW was already retrieved, the 82395DX provides the requested DW in zero wait states (a Line Buffer hit). If the requested DW wasn't already retrieved, it will be read after completing the Line Fill.

The second and third cases are attained by having the Four DW write buffer which is described in chapter 6. The READYO# signal is driven active after latching the write cycle, so all buffered cycles will run without wait states. This releases the 386 DX Microprocessor to issue a new cycle, which can also run without wait states if it does not require system bus service. Two examples are in the case of a cache read hit cycle, or another buffered write cycle, which does not require immediate System Bus service. In the case of a write cycle to the same line currently retrieved, the write cycle will wait until the Line Fill is complete and then the selected bytes within the line are written in the cache. READYO# is returned after the cache is written.



NOTE:
The second and third reads are to a different line and are serviced from the cache while the fourth and fifth reads are to the same line and are serviced from the line buffer.

Figure 4.1 - Read Hit Cycles During a Line Fill

Whenever the System Bus is released to any bus master, the 82395DX activates the snooping function. The maximum rate of snooping cycles is a cycle every other clock. Although the snooping support requires accessing the 82395DX cache directory, the 82395DX is able to interpose the cache directory accesses between the 386 DX Microprocessor cycles and the snooping device such that zero wait state cache read hit cycles are supported. All the snooping cycles are also serviced. This is how the fourth case is provided. For more details, refer to chapter 6.

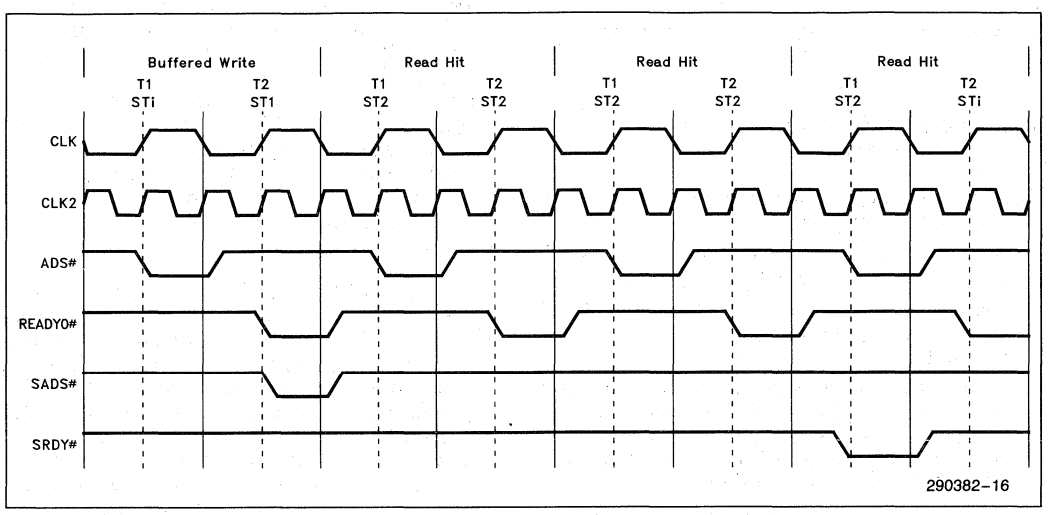


Figure 4.2 - Cache Read Hit Cycles while Executing a Buffered Write on the System Bus

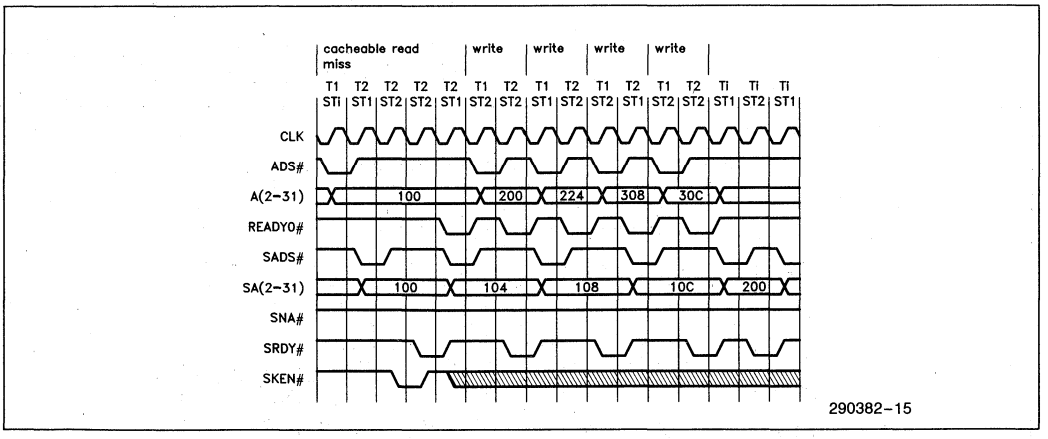
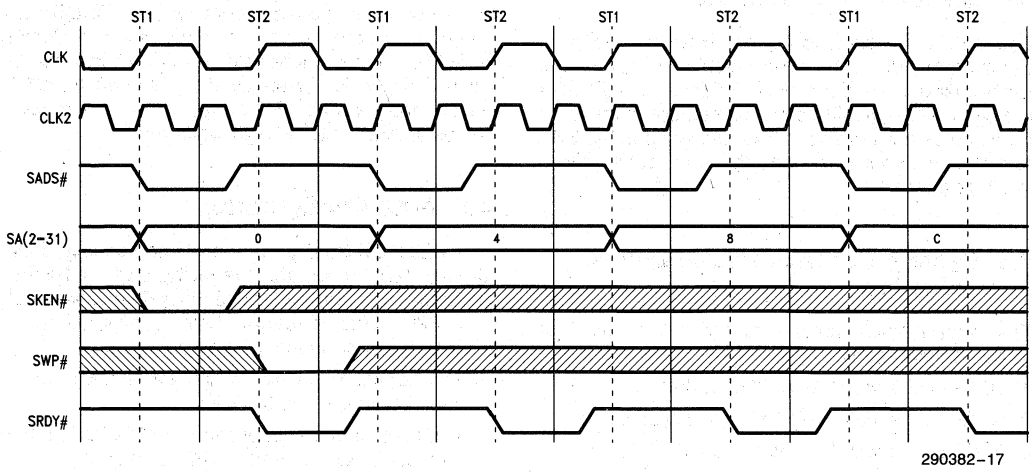
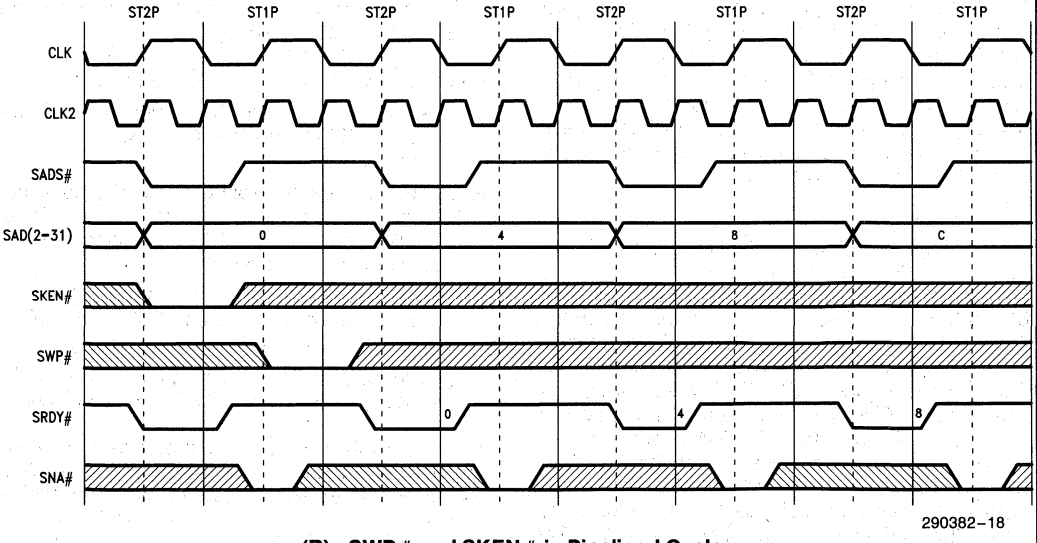


Figure 4.3 - Buffered Write Cycles During a Line Fill



(A) - SWP# and SKEN# in Non-Pipelined Cycles



(B) - SWP# and SKEN# in Pipelined Cycles

Figure 4.4 - SWP# and SKEN# Timing

5

4.4 Disabling the 82395DX

Cacheability is resolved by the SKEN# input from the system side. In order to disable the cache it is recommended to deactivate SKEN# and FLUSH the cache. This would cause all memory reads to be detected as misses and to be transferred to the System Bus. In order to disable the write buffer, NPI# must be asserted.

4.5 System Description and Device Selection

The expandability feature provides the following three configurations:

1) 16KB cache with one 82395DX device.

2) 32KB cache with two 82395DX devices.

3) 64KB cache with Four 82395DX devices.

In multi 82395DX configurations, the total Cache Directory and cache is partitioned between the various 82395DXs. For example, in the second configuration, the first 82395DX includes the first 16KB cache and the first 1K tags while the second 82395DX includes the second 16KB cache and the second 1K tags. Every 82395DX is programmed to handle a portion of the cache and the Cache Directory. The 82395DX selection is based on decoding the address of the cacheable cycle.

In multi 82395DX system, one device must be programmed as the Primary 82395DX to drive the system bus in System Bus cycles (non-cacheable cycles, write cycles and also in Line Fills). All other 82395DXs must be programmed as Secondary 82395DXs. They drive only the SADS# signal in Line Fill cycles. All other System Bus signals are driven by the Primary 82395DX. System diagram 4.6 describes the 64KB cache system. In the Local Bus, each 82395DX gets the 386 DX Microprocessor address, control and data signals. In cacheable reads, hits or misses, the selected 82395DX drives the READYO# and the local data bus. In all other cycles, the Primary drives these signals. The READYO#s of all the 82395DXs are wire-ORed together and they can be logically ORed with the READYO#s of local bus devices. An External pull-up must exist on the 82395DX READYO# to sustain it high. The selected 82395DX drives the READYO# low and keeps it low while the READYI# is not sampled active. Immediately with sampling the READYI# active, the selected 82395DX drives the READYO# high for one phase and floats it in the next phase. Therefore, zero wait state cycles are supported.

In the System Bus, the Primary 82395DX drives all the system bus outputs except SADS#. SADS# is a wire-ORed signal which is driven by the Primary 82395DX in non-cacheable reads and in write cycles. SADS# is driven by the selected 82395DX which requires a Line Fill cycle. A pull-up is required to sustain the SADS# high while not driven.

4.6 Auto Configuration

The 82395DX configures itself automatically during the first ten clocks after the falling edge of RESET. Information on the system configuration is passed to the 82395DXs through their configuration pin (CONF#), by connecting them as follows:

1. The configuration pin of first 82395DX (primary) must be connected to GND.
2. The configuration pin of second 82395DX (optional) must be connected to RESET signal.
3. The configuration pin of third 82395DX (optional) must be connected to READYO# signal.
4. The configuration pin of fourth 82395DX (optional) must be connected to VCC.

Auto configuration process works as follows:

1. If the 82395DX senses the configuration pin low during RESET, the device is configured as device #1 (primary).
2. Otherwise, if the 82395DX senses the configuration pin low one clock cycle after reset, the device is configured as device #2, and issues a READYO# pulse for one clock period.
3. Otherwise, if 82395DX senses the configuration pin low three clock cycles after RESET is sensed low, the device is configured as device #3.
4. Otherwise, the device is configured as device #4, and issues READYO# pulse for one clock period.

All the 82395DXs in the system monitor the number of pulses on READYO# during the first 4 clocks after RESET, to determine how many 82395DXs are present.

1. If no pulse was sensed, there is only one 82395DX.
2. If one pulse is sensed, there are two 82395DXs in the system.
3. If two pulses were sensed, there are 4 82395DXs in the system.

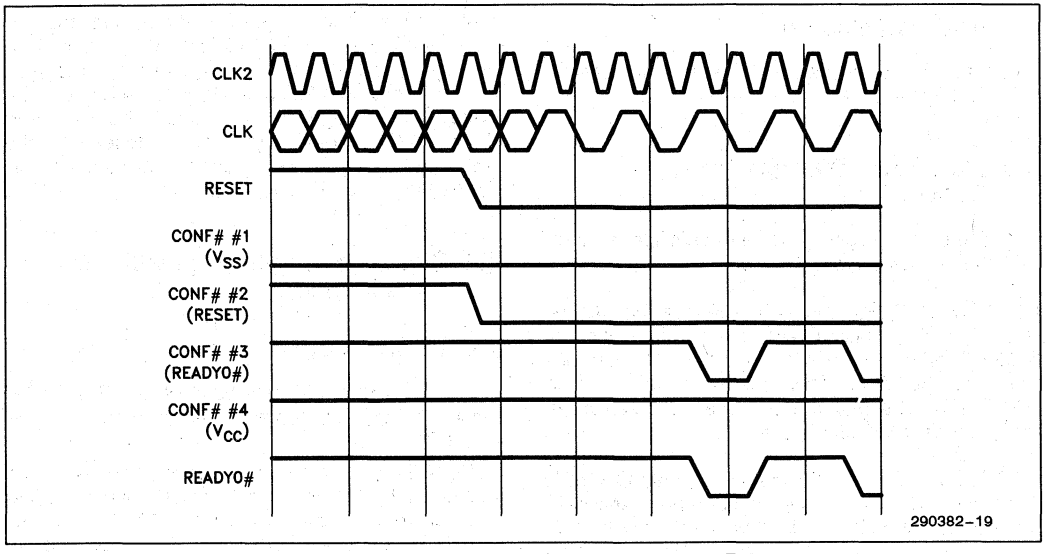


Figure 4.5 - Self Configuration of Four 82395DXs

5

4.7 Address Mapping

Table 4.2 shows the cache address configurations for 16K, 32K, and 64K cache sizes.

4.8 Multi 82395DX Operation Description

The following is a description of each cycle in a multi-82395DX environment:

Local Bus CYCLES: Cycles to any local bus device (e.g. 387 DX Math Coprocessor). The Primary 82395DX drives the READYO# in 387 DX Math Coprocessor accesses after one wait state, unless READYI# was sampled active one clock earlier. All the secondary 82395DXs are idle.

CACHEABLE READ HIT: this is the only 82395DX cycle which does not require system bus service. In this cycle, the selected 82395DX drives the local data bus and the READYO# in T2. Also, it updates its LRU bits.

CACHEABLE READ MISS: As soon as the system bus is available, the selected 82395DX, which detected the miss, drives the SADS#. In parallel, the Primary 82395DX drives the system bus address and control signals. After receiving the first SRDY# or SBRDY# and after sampling the SKEN# active, the selected 82395DX samples the system data and one clock later it provides it to the 386 DX Microprocessor and drives the READYO# active. Then, it continues in filling the line and, after collecting the four DWs, it updates its cache and Cache Directory.

CACHEABLE WRITE HITS: the selected 82395DX updates its cache, except for write protected cycles. The Primary 82395DX, however, executes the write cycle on the system bus. Notice that both the Primary and Secondary 82395DXs have the same write buffer and both handle the cycle in the same way, but the Primary 82395DX is the one which drives the system bus signals, including SADS# and READYO#. All other cycles i.e. cacheable write misses and non-cacheable cycles are handled only by the Primary 82395DX.

4.9 Signal Driving in Multi 82395DX Environment

4.9.1 Local Bus Signals

In the Local Bus, the data bus and the READYO# signals are the only signals driven by more than one 82395DX.

1. READYO#: normally not driven (floated), and must be sustained by an external pullup. In cacheable reads, the selected 82395DX drives READYO# active until READYI# is sensed active, than it drives READYO# inactive for one clock phase and then floats it. In other cycles, the primary 82395DX drives READYO# in the same manner.
2. Data Bus: The selected (or primary) 82395DX drives the data bus in the T2 state of read cycles, which ensures no contention with the 386 DX Microprocessor when a write cycle follows a read cycle.

4.9.2 SYSTEM BUS SIGNALS

In the System Bus, the Primary 82395DX drives all the System Bus signals except SADS#. So, the jeopardy of contention exists on the SADS# signal

Table 4.2 - Address Mapping for 1-4 82395DX Systems

Device No.	Total Devices in System	Address Decoding	Primary/ Secondary	Cache Data Mapping	Cache Directory SETs
1	1	——	P	0KB-16KB	0-255
1	2	A12#	P	0KB-16KB	0-255
2	2	A12	S	16KB-32KB	256-511
1	4	A13#*A12#	P	0KB-16KB	0-255
2	4	A13#*A12	S	16KB-32KB	256-511
3	4	A13 *A12#	S	32KB-48KB	512-767
4	4	A13 *A12	S	48KB-64KB	768-1023

only. SADS# is normally not driven (floated), and must be sustained by an external pullup. Every 82395DX, Primary or Secondary, after driving the SADS# active in ST1 or ST2P, will drive it inactive for one clock phase in ST2 or ST1P, and float it afterwards.

In Line Fills, the SADS# is driven by the selected 82395DX which detected the miss. In all other cycles e.g. write cycles, the SADS# is driven by the Primary 82395DX.

4.10 SHOLD/SHLDA/SBREQ Arbitration Mechanism

The Primary 82395DX is responsible for handling the SHOLD/SHLDA/SBREQ mechanism. Assuming that the SHOLD is acknowledged, the Primary 82395DX floats all its outputs immediately after completing the system bus cycle in which SHOLD was activated and it drives SHLDA active. This enables the bus master to get control of the bus. When the bus master completes its cycles, it drives the SHOLD signal inactive. Then the Primary 82395DX gets the bus back by driving the SHLDA inactive.

The Secondary 82395DXs get the SHOLD input in order to monitor the bus activity but they don't drive the SHLDA. Secondary 82395DXs do not drive the SADS# in Hold states. The Primary 82395DX drives the SBREQ signal in all System Bus cycles. In Line Fill cycles, the SBREQ signal is driven active one clock later than in other cycles. Of course, this is applicable for the case the System Bus is not available. If the System Bus is available, the SBREQ will not be driven in Line fill cycles. For more details about system arbitration, refer to Chapter 6.

4.11 System Description

A 386 DX Microprocessor/ 82395DX-based system includes the processor, optional Local bus devices (e.g. 387 DX Math Coprocessor), cache system (one 82395DX or more) and System Bus devices (memory, I/O devices and other non-cacheable devices). The 82395DX is the interface between the Local Bus and the System Bus.

A Local Bus address decoder must be used to generate LBA# and NPI# signals, and a System Bus address decoder must be used to generate SKEN# and SWP# signals.

The 82395DX READYO# may be logically ORed with READYO#s of other Local Bus devices. However, this is not required unless a Local Bus device, other than 387 DX Math Coprocessor, exists on the local bus (82395DX generates a READY signal for the 387 DX Math Coprocessor). The 386 DX Microprocessor READY# input signal must also be driven to the 82395DX READYI# pin, so that the 82395DX will be able to track the Local Bus cycles correctly.

Table 4.3 - Local Bus Signal Connections in Multi-82395DX Systems

Primary 82395DX Only		Each 82395DX in the System	
Signal	**Type**	**Signal**	**Type**
		CLK2	I
		D0–D31	I/O
		A2–A31	I
		RESET	I
		BE0-3#	I
		W/R#	I
		D/C#	I
		M/IO#	I
		LOCK#	I
		ADS#	I
		READYI#	I
		LBA#	I
		NPI#	I
		PLUSH#	I
		A20M#	I
		CONF#	I
		READYO#	I/O

To allow for expanding the cache system beyond 16KB, up to four 82395DX devices may be connected in parallel. Two 82395DX outputs are Wire-ORed between the parallel 82395DXs: READYO# and SADS#. Each of the 82395DXs' CONF# input must be tied to a different signal, to program each one of them to a distinct address decoding.

Figure 4.6 describes a maximum 386 DX Microprocessor/82395DX system, with 387 DX Math Coprocessor, four 82395DX devices, READY# generation logic and Local Bus/System Bus address decoders.

Note that optional elements in Figure 4.6 are drawn with dotted line. The Local Bus includes CLK2, RESET, BE3#–BE0#, A2–A31, D0–D31, W/R#, D/C#, M/IO#, LOCK# and ADS#. The System Bus can be broken into two groups. Those pins connected only to the primary 82395DX (82395DX #1) and those connected to each 82395DX in the system (82395DX #1–#4). See Table 4.4.

Table 4.4 - System Bus Signal Connections in Multi-82395DX Systems

Primary 82395DX Only		Each 82395DX in the System	
Signal	**Type**	**Signal**	**Type**
SA2–3	O	SD0–31	I/O
SW/R#	O	SA4–31*	I/O
SD/C#	O	SADS#	O
SM/IO#	O	SRDY#	I
SLOCK#	O	SBRDY#	I
SBREQ	O	SNA#	I
SHLDA	O	SHOLD	I
SBLAST#	O	SAHOLD	I
SNENE#	O	SEADS#	I
		SFHOLD#	I
		SKEN#	I
		SWP#	I

*SA4–31 are connected to each 82395DX in the system for snooping purposes but are driven only by the primary 82395DX.

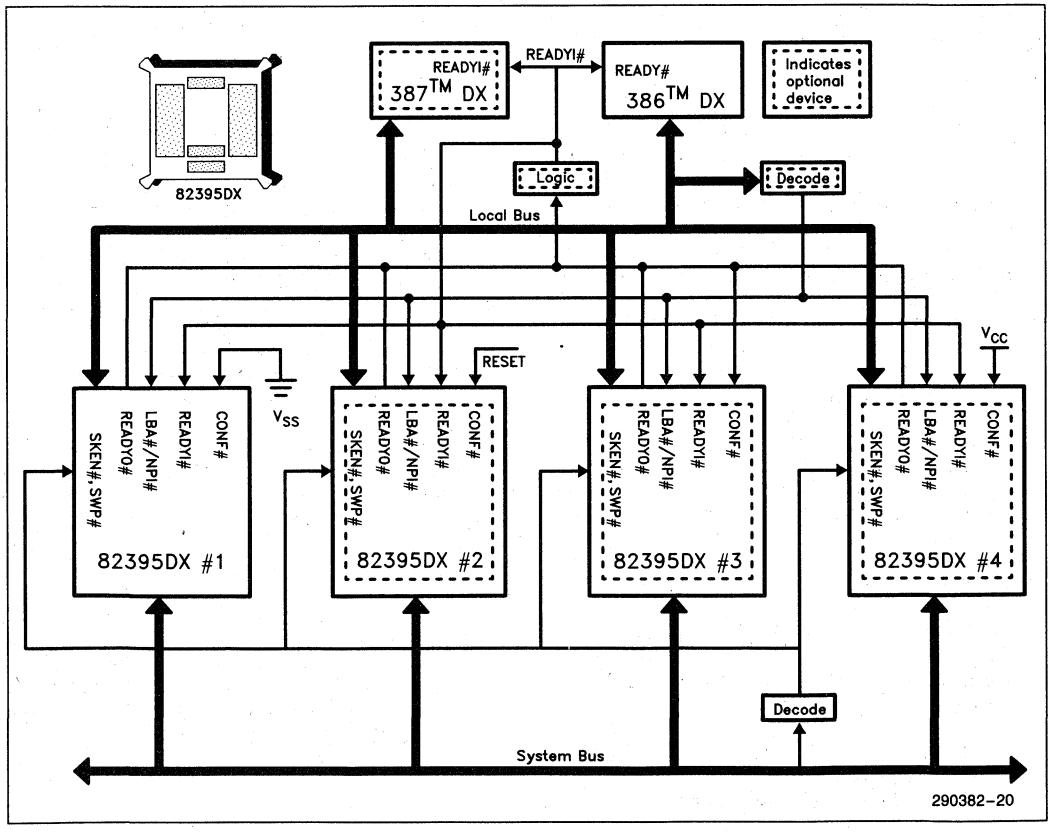


Figure 4.6 - System Description

5.0 PROCESSOR INTERFACE

The 82395DX runs synchronously with the 386 DX Microprocessor. It is a slave on the Local Bus, and it buffers between the Local Bus and the System Bus. Most of the 82395DX cycles are serviced from the internal cache, and some (82395DX cache misses, non cacheable accesses, etc.) require an access to the System Bus to complete the transaction.

To achieve maximum performance, the 82395DX serves cache hits and buffered write cycles in zero wait- state, non-pipelined cycles. The 82395DX requires that the CPU is never driven to pipelined cycles, i.e. the 386 DX Microprocessor NA# input must be strapped to inactive (high) state.

The 82395DX is directly connected to all local bus address and data lines, byte enable lines, and bus cycle definition signals. The 82395DX returns READYO# to the 386 DX Microprocessor, and keeps track of the 386 DX Microprocessor cycle status by receiving READYI# (which is the 386 DX Microprocessor READY#).

A multi 82395DX system description was presented in chapter 4.

5.1 Hardware Interface

The 82395DX requires minimal hardware on the Local Bus. Other than the 386 DX Microprocessor and other Local Bus resources (i.e. 387 DX Math Coprocessor) and the 82395DX(s) (1-4 depending on the system). Ready logic and a Local Bus decoder are optional since the user can wire OR the READYO#s and tie LBA# and NPI# high if no addresses are to be local or non-buffered. The SRAM and buffers have been integrated on chip to simplify the design. Refer to Figure 4.6.

5.2 Nonpipelined Local Bus

The 82395DX does not pipeline the Local Bus. READYO# gets returned to the 386 DX Microprocessor one cycle after SRDY# or SBRDY# are driven into the 82395DX after the first DW of a Line Fill. This allows the Local Bus to be free to execute 386 DX Microprocessor cycles while the System Bus fills the cache line (see chapter 6).This takes away the advantage gained by pipelining the Local Bus.

5.3 Local Bus Response Hit Cycles

The 82395DX's Local Bus response to hit cycles are described here:

1) Cache Read Hit (CRDH) Cycle — READYO# gets returned in T2. The Data is valid to the 386 DX Microprocessor on the rising edge of CLK2.
2) Cache Write Hit (CWTH), Buffered — Like in CRDH cycles the 82395DX returns READYO# in T2 so that the cycle runs with zero wait states on the Local Bus. The write cycle is placed in the write buffer and will be performed when the System Bus is available. If the System Bus is on HOLD up to four write cycles can be buffered before introducing any wait states on the Local Bus.
3) CWTH, Non-Buffered — In the case of a non-buffered write hit cycle the write buffers can not be used so the 386 DX Microprocessor must wait until the System Bus is free to do the write. READYO# is returned to the cycle after SRDY# is driven to the 82395DX.

5.4 Local Bus Response to Miss Cycles

In a Cache Read Miss (CRDM) cycle a Line Fill is performed on the System Bus. READYO# is returned to the 386 DX Microprocessor one cycle after SRDY# or SBRDY# for the first DW of the Line Fill is driven into the 82395DX.

5.5 Local Bus Control Signals — ADS#, READYI#

ADS# and READYI# are the two bus control inputs used by the 82395DX to determine the status of the Local Bus cycle. ADS# denotes the beginning of a 386 DX Microprocessor cycle and READYI# is the 386 DX Microprocessor cycle terminator.

ADS# active and M/IO# = 1 invokes a look-up request to the 82395DX's cache directory; the look-up is performed in T1 state. The Cache Directory access is simultaneous with all other cycle qualification activities, this Way the hit/miss decision becomes the last in the cycle qualification process. This parallelism enhances performance, and enables the 82395DX to respond to ADS# within one clock period. If the cycle is to a Local Bus device (LBA# asserted) or is non-cacheable, the hit/miss decision is ignored.

5.6 82395's Response to the 386 DX Microprocessor Cycles

Tables 5.2 - 5.4 show the 82395DX's response to the various 386 DX Microprocessor cycles. They depict the activity in the internal cache, cache directory, the System Bus and write buffers in response to various cycle definition signals. Special cycles such as: LOCK, HALT/SHUTDOWN, WP, LBA, NPI are discussed separately below.

5.6.1 LOCKED CYCLES

The 386 DX Microprocessor LOCK#ed cycles are all those cycles in which LOCK# is active. The 82395DX forces all LOCK#ed cycles to run on the System Bus. The 82395DX starts the LOCK#ed cycle after it has emptied its write buffers.

If the LOCK#ed cycle is cacheable the 82395DX will respond as follows (see table 5.2):

Cache Read Miss (CRDM) — handled similar to a non cacheable cycle.

Cache Read Hit (CRDH) — handled similar to a non cacheable cycle (LRU bits are not updated).

Cache Write Miss (CWTM) — the cache is not updated, the write is not buffered.

Cache Write Hit (CWTH) — the cache is updated if the line is not write protected. The write is not buffered. Note that this write is not buffered even though it is cacheable. The LRU mechanism is updated.

If the LOCK#ed cycle is non-cacheable (e.g. IO cycle, INTA cycle) then it will be performed as a common non-cacheable cycle with the addition of asserting SLOCK# on the System Bus.

Conceptually, a LOCK# cycle on the Local Bus is reflected into an SLOCK# cycle on the System Bus. Detailed timing considerations were presented in chapter 3. SLOCK# becomes inactive only after LOCK# has become inactive. If there are idle clocks in between the LOCK#ed cycles but LOCK# is still active - SLOCK# will remain active as well. **A consequence of this is that SLOCK# is negated one clock after LOCK# is negated.**

During LOCK#ed cycles on System Bus (i.e. when SLOCK# signal is active), the 82395DX does not acknowledge hold requests so the whole sequence of LOCK#ed cycles will run without interruption by another master.

Note that when a LOCK#ed LBA# cycle runs on the Local Bus, and the System Bus is idle and not at HLDA state, SLOCK# will be asserted even though the LBA# cycle will not be transferred to the system bus.

5.6.2 I/O, HALT/SHUTDOWN

I/O and HALT/SHUTDOWN cycles are handled as non-cacheable cycles. They are neither cached nor kept in the write buffer. The 386 DX Microprocessor HALT/SHUTDOWN cycles are memory write cycles to code area (i.e. M/IO# = 1, D/C# = 0). The 82395DX completes I/O and HALT/ SHUTDOWN cycles by returning READYO#, after receiving the SRDY#.

5.6.3 LBA# CYCLES

LBA# cycles are all the 386 DX Microprocessor cycles in which LBA# is active, or all cycles in which the 387 DX Math Coprocessor or Weitek 3167 Floating-Point Coprocessor is addressed. A CPU access to I/O space with A31 = 1 is decoded as a 387 DX Math Coprocessor access. A CPU access to memory space C0000000H through C1FFFFFFH is decoded as a Weitek 3167 Floating-Point Coprocessor access, provided that the Weitek decoding is enabled.

When an LBA# cycle is detected all other attributes are ignored. If a 387 DX Math Coprocessor access is decoded, READYO# is activated as described in section 5.6. No other activity takes place.

5.6.4 NPI# CYCLES

NPI# cycles are all the 386 DX Microprocessor memory write cycles in which NPI# is active. In response to a cycle with NPI# active, the 82395DX first executes all pending write cycles in the write buffer (if any), and then executes the current write cycle on the System Bus. READYO# is returned to the CPU only after SRDY# for the current write cycle is returned to the 82395DX.

All NPI# cycles must have at least one wait state on the System Bus or be done to non-cacheable memory.

NPI# is ignored for read cycles, as well as all write cycles that cannot be buffered.

5.6.5 LBA#/NPI# TIMING

These inputs must be valid throughout the 386 DX Microprocessor bus cycle, namely in T1 and all T2 states (See Figure 5.1).

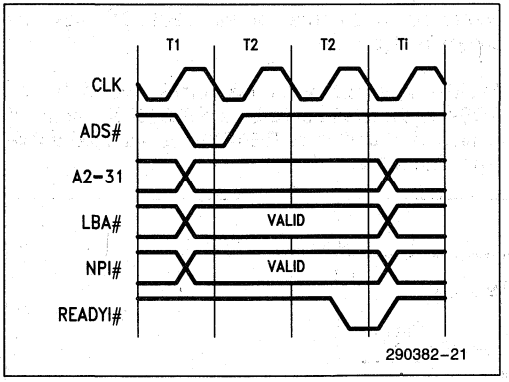


Figure 5.1 - Valid Time of LBA# and NPI#

5.7 82395DX READYO# Generation

The 82395DX READYO# generation rules are listed below:

CRDH cycles (non-LOCK#ed), READYO# is activated during the first T2 state, so the cycle runs with zero wait states.

CRDM cycles - READYO# is returned one clock after the first SRDY# or SBRDY#.

Non cacheable reads - READYO# is returned one clock after SRDY# or SBRDY#.

All cacheable writes (with the exception of LOCK#ed writes) are buffered. These cycles may be divided into two categories:

(a) The first four write cycles — while the write buffer is not fully exploited. READYO# is returned in zero wait states. The address and the data are registered in the write buffer.

(b) When the write buffer is full — READYO# is delayed until one clock after the SRDY# or SBRDY# of the first write cycle in the buffer. In other words the fifth write waits until there is one vacant entry in the write buffer.

Non cacheable writes (plus LOCK#ed writes) — these writes are not buffered. READYO# is returned one clock after SRDY# or SBRDY# of the same cycle.

READYO# activation during SRAM mode is described in Chapter 7. READYO# activation during self configuration is listed in Chapter 4.

In all 387 DX Math Coprocessor accesses, the 82395DX monitors the READYI#. If it wasn't activated immediately after ADS#, READYO# will be activated in the next clock i.e. a one wait state cycle. So, the 82395DX READYO# can be used to terminate any 387 DX Math Coprocessor access.

Note that the timing of the 82395's READYO# generation for 387 DX Math Coprocessor cycles is incompatible with 80287 timing. When activated, READYO# remains active until READYI# is sampled active. This procedure enables adding control logic to control the 386 DX Microprocessor READYI# generation (see Figure 5.2).

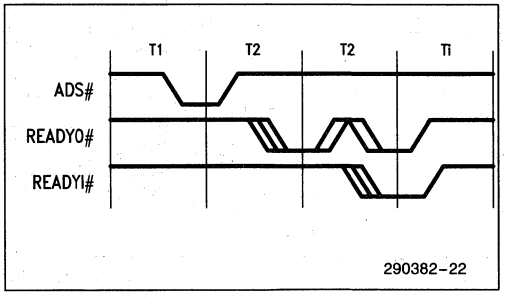


Figure 5.2 - Externally Delayed READY

In a multi-82395DX system, each device on the Local Bus must be able to return READYO#. Therefore, READYO# is wired OR on the Local Bus. READYO# is normally floated, and it is connected to the positive power supply by a pull-up resistor. An external OR gate ORs the 82395DXs' READYO#s with the READYO# of all other Local Bus devices.

5

5.8 A20 Mask Signal

The A20M# signal is provided to allow for emulation of the address wraparound at 1 MByte which occurs on the 8086. A20M# pin is synchronized internally by the 82395DX, then ANDed with the A20 input pin. The product of synchronized A20M# and A20 is presented to the rest of the 82395DX logic, as shown in Figure 5.3.

A20M# must be valid two clock cycles before ADS# is sampled active by the 82395DX, and must remain valid until after READYI# is sampled active (see Figure 5.4).

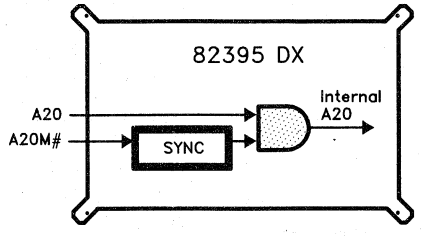


Figure 5.3 - A20 MAsk Logic

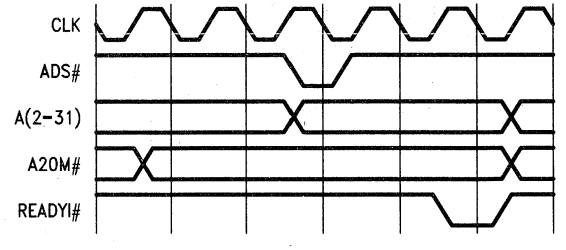


Figure 5.4 - Valid Time of A20M#

5.9 82395DX Cycle Overview

Table 5.1 - 386 DX Microprocessor Bus Cycle Definition

M/IO#	D/C#	W/R#	386 DX Microprocessor Cycle Definition
0	0	0	Interrupt Acknowledge
0	0	1	Undefined
0	1	0	I/O Read
0	1	1	I/O Write
1	0	0	Memory Code Read
1	0	1	Halt/Shutdown
1	1	0	Memory Data Read
1	1	1	Memory Data Write

Table 5.2 describes the activity in the cache, in the Tagram, on the System Bus and in the write buffers. The cycles are defined in table 5.1. Table 5.2 is sorted in a descending order. The more dominant the attribute the higher it is located. For example, if the cycle is both LBA# and I/O it is considered an LBA# cycle. Table 5.2 is for non test modes.

Table 5.2 - Activity by Functional Groupings

Cycle Type	WP	Cache	TAGRAM		System Bus	Posted Write	Comm.
			LRU	TAG			
1. LBA & 387/Weitek Cycles	N/A	—	—	—	—	N/A	
2. I/O Write, I/O Read, Halt/Shutdown, INTA, LOCK#ed Read	N/A	—	—	—	Non Cacheable Cycle	No	2
3. LOCK#ed Write Hit	Yes		Update	—	Memory Write	No	2
4. LOCK#ed Write Hit	No	Cache Write	Update	—	Memory Write	No	2
5. LOCK#ed Write Miss	N/A	—	—	—	Memory Write	No	2
6. Other Read Hit	N/A	Cache Read	Update	—	—	N/A	1
7. Other Read Miss SKEN# Active	N/A	Cache Write	Update	Update	Line Fill	N/A	2
8. Other Read Miss SKEN# Inactive	N/A	—	—	—	Noncacheable Read No Line Fill	N/A	2
9. Other Write Hit NPI# Inactive	Yes	—	Update	—	Memory Write	Yes	1
10. Other Write Hit NPI# Active	Yes	—	Update	—	Memory Write	No	2
11. Other Write Hit NPI# Inactive	No	Cache Write	Update	—	Memory Write	Yes	1
12. Other Write Hit NPI# Active	No	Cache Write	Update	—	Memory Write	No	2
13. Other Write Miss NPI# Inactive	N/A	—	—	—	Memory Write	Yes	1
14. Other Write Miss NPI# Active	N/A	—	—	—	Memory Write	No	2

Table 5.3 describes line buffer hit cycles. Hit/miss here means to the specific DW in the line buffer.

Table 5.3. Activity in Line Buffer Hit Cycles

Cycle Type	WP	Cache	TAGRAM		System Bus	Posted Write	Comm.
			LRU	TAG			
15. LOCK#ed Write	Yes	—	—	—	Memory Write	No	2
16. LOCK#ed Write	No	Cache Write	—	—	Memory Write	No	4
17. Read Hit	N/A	LB Read	—	—	—	N/A	1
18. Read Miss	N/A	LB Read	—	—	—	N/A	3
19. Other Write NPI# Inactive	Yes	—	—	—	Memory Write	Yes	6
20. Other Write NPI# Active	Yes	—	—	—	Memory Write	No	2
21. Other Write NPI# Inactive	No	Cache Write	—	—	Memory Write	Yes	5
22. Other Write NPI# Inactive	No	Cache Write	—	—	Memory Write	No	4

Table 5.4 describes the line buffer hit cycles, when the Line Fill is interrupted (by: FLUSH#, snoop hit to the line buffer or interrupted burst, even if the Line Fill continues on the System Bus in the first two cases). The table includes only the cycles which wait to the end of the Line Fill or to the CPU cache update. Hit/Miss here means to the right DW in the line buffer.

Table 5.4. Activity in the Line Buffer During ALF Cycles

Cycle Type	WP	Cache	TAGRAM		System Bus	Posted Write	Comm.
			LRU	TAG			
23. LOCK#ed Write	N/A	—	—	—	Memory Write	No	2
24. Read Miss (Restart)	N/A	Cache Write	Update	Replace	Line Fill	N/A	2
25. Other Write NPI# Inactive	N/A	—	—	—	Memory Write	Yes	5
26. Other Write NPI# Active	N/A	—	—	—	Memory Write	No	2

Table 5.5 depicts the 82395DX Test Cycles.

Table 5.5. Activity in Test Cycles

Cycle Type	WP	A16	Cache	TAGRAM		System Bus	Posted Write	Comm.
				LRU	TAG			
27. High Impedance	N/A	N/A	—	—	—	—	N/A	
28. SRAM Mode Read Add 256K-512K	N/A	0	—	LRU RD	TAG RD	—	N/A	
29. SRAM Mode Read Add 256K-512K	N/A	1	Cache	—	—	—	N/A	
30. SRAM Mode Write Add 256K-512K	N/A	0	—	LRU WR	TAG WR	—	N/A	
31. SRAM Mode Write Add 256K-512K	N/A	1	Cache Write	—	—	—	N/A	
32. SRAM Mode Read Add <>256K-512K	N/A	N/A	—	—	—	Noncacheable Cycle	No	2
33. SRAM Mode Write Add <>256K-512K	N/A	N/A	—	—	—	Noncacheable Cycle	N/A	

Remarks for Tables 5-2 through 5-5:

1. READYO# is active in the first T2. (In read cycles, in write it depends if the write buffer is full).
2. READYO# is active one clock cycle after SRDY#/SBRDY# of this cycle is asserted. In case of Line Fill, READYO# is active one clock cycle after first SRDY#/SBRDY# of this cycle is asserted.
3. READYO# is active immediately after the current line fill is finished.
4. READYO# is active after the previous line fill and the write cycle are terminated by SRDY# or SBRDY#, and the cache is updated.
5. READYO# is active after the cache is updated for the previous Line Fill, or after the Line Fill is aborted.
6. READYO# is active on the third T2 (2 wait states) if the write buffer is not full.
7. "OTHER" means the cycle does not fall within the first five categories.

6.0 SYSTEM BUS INTERFACE

The System Bus (SB) interface is similar to the 386 DX Microprocessor interface. It runs synchronously to the 386 DX Microprocessor clock. In general, the interface is similar to the 82385 in terms of: System Bus pipelining, snooping support and write cycle buffering. In addition, the following enhancements are provided:

1) Line Fill buffer.

2) Optional burst Line Fill.

3) System cacheability attribute, SKEN#.

4) System Write Protection attribute, SWP#.

5) The SBREQ/SHOLD/SHLDA arbitration mechanism to support multi master systems.

6) The SEADS# snooping mechanism to support concurrency on the System Bus and on the general purpose bus.

7) SFHOLD# mechanism to resolve deadlocks in multiprocessing systems.

8) Four Double-Word write buffer (16 bytes).

9) SNENE# (System NExt NEar) function to simplify the design of page mode DRAM system, and save wait states.

The 82395DX System Bus interface has identical bus signals to the 386 DX Microprocessor bus. It has the bus control signals (SADS#, SRDY# and SNA#), the cycle definition signals (SLOCK#, SW/R#, SD/C# and SM/IO#), the address and byte enable signals (SA2–SA31 and SBE0#–SBE3#) and the data signals (SD0–SD31). In addition, the 82395DX has the SBRDY# signal for burst support. The SKEN# signal for the system cacheability attribute. The SWP# signal for the system Write Protection attribute. The SAHOLD and SEADS# signals for snooping support. The SBREQ, SHOLD and SHLDA signals for system arbitration. And SNENE# for DRAM hook-up. Also, the 82395DX provides a signal, SBLAST#, which when asserted, indicates that the current cycle is the last cycle in a burst transfer.

The 82395DX System Bus interface can support any device, non cacheable, I/O or cacheable memory with any number of wait states. The 82395DX is able to support one clock burst cycles. The 82395's System Bus state machine is similar to the 386 DX Microprocessor bus state machine (refer to the "386 DX Microprocessor data sheet"). Note that during burst Line Fill, the 82395DX remains is ST2 state until SRDY# or SBRDY# is asserted for the fourth cycle of the burst transfer. Figure 6.1 describes the 82395's System Bus state machine.

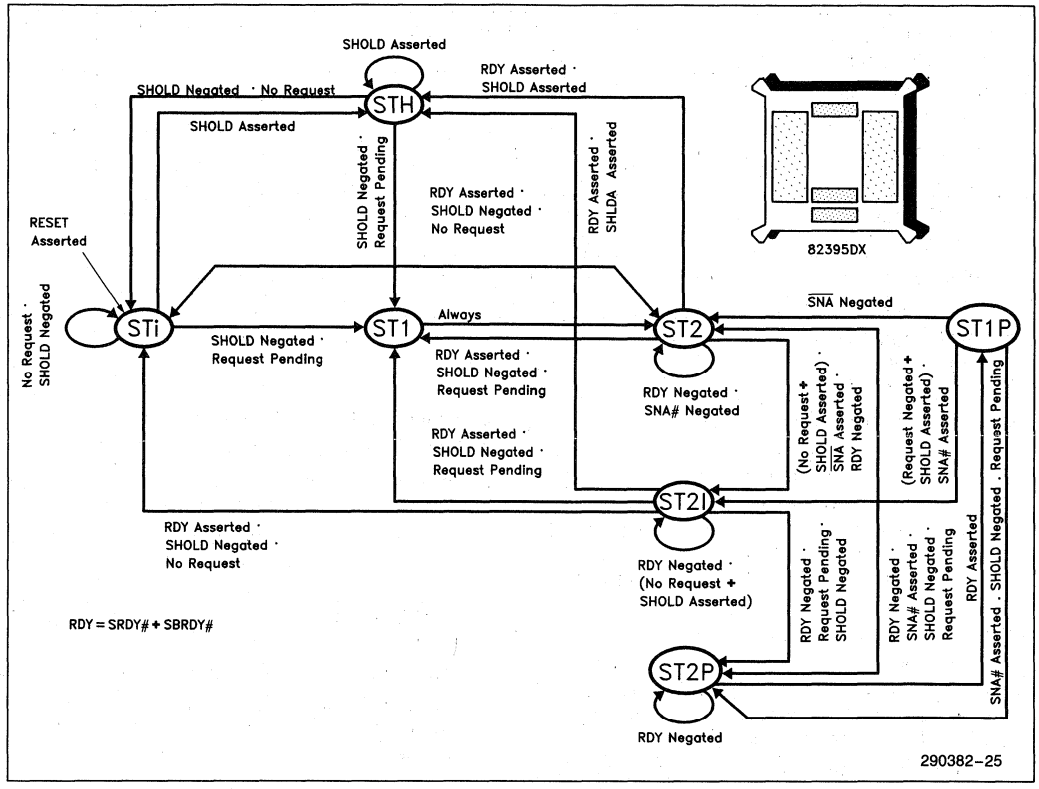


Figure 6.1 - SB State Machine

6.1 System Bus Cycle Types

Following five types of SB cycles are supported:

1) Buffered write cycle
2) Non buffered write cycle
3) Buffered/non-buffered write protected cycles.
4) Non cacheable read cycle
5) Cacheable read cycle

6.1.1 BUFFERED WRITE CYCLE

All the cacheable write cycles, except LOCK#ed write cycles or non-buffered write cycles (as indicated by NPI# pin sampled active), are buffered. These cycles are terminated on the Local Bus before they are terminated on the System Bus.

The following Figures (6.2 - 6.3) include waveforms of several cases of buffered write cycles:

The 82395DX has a four DW deep write buffer but five writes cycles can be buffered if one of the buffered writes is being executed.

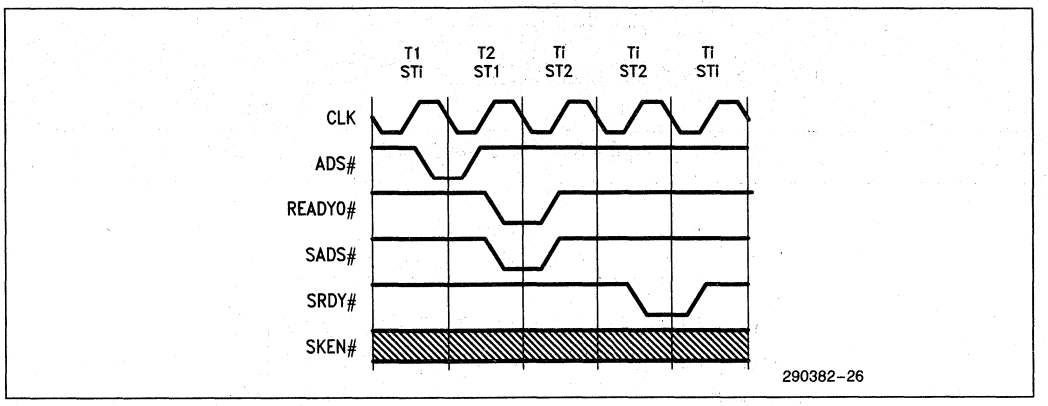
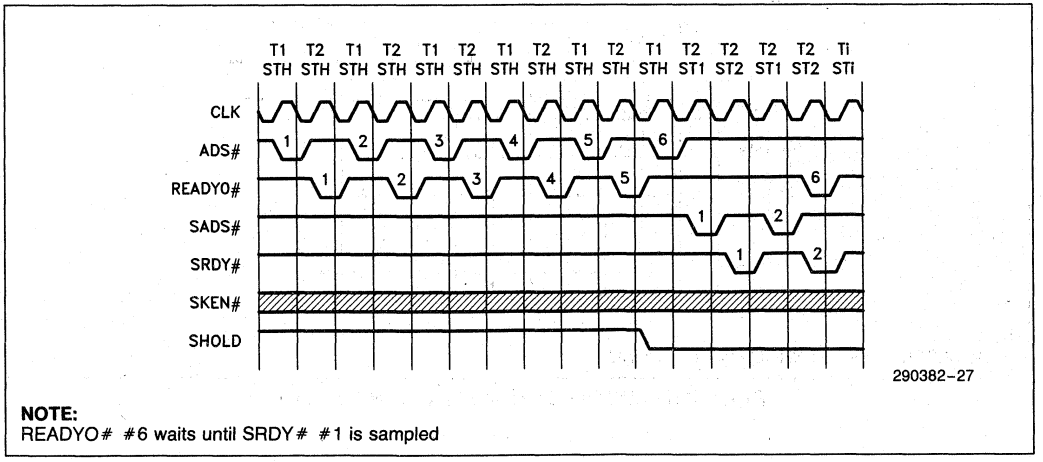


Figure 6.2 - Single Buffered Write Cycle



NOTE:
READYO# #6 waits until SRDY# #1 is sampled

Figure 6.3 - Multiple Buffered Write Cycles During System Bus HOLD

6.1.2 NON-BUFFERED WRITE CYCLE

These cycles are terminated on the System Bus one clock before they are terminated on the Local Bus.

The following Figures (6.4 - 6.5) include waveforms of several cases of non buffered write cycles.

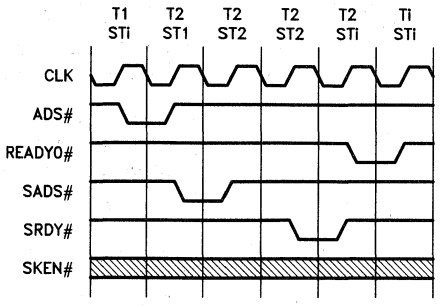
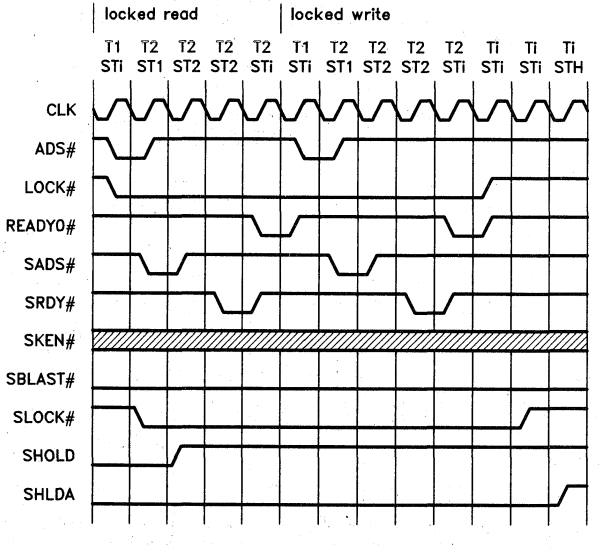


Figure 6.4 - I/O Write Cycle



NOTE:
While SLOCK# is active SHOLD input is ignored

Figure 6.5 - LOCK#ed "Ready Modify Write" cycle

6.1.3 WRITE PROTECTED CYCLES

The Write Protection attribute is provided by the system bus SWP# input. The SWP# is sampled with the first SRDY# or SBRDY# in every Line Fill cycle. The write protection indicator is registered in the Cache Directory together with the TAG address and TAG Valid bit of every line. In every cacheable write cycle, the write protection indicator is read simultaneously with the Hit/Miss decision. If the write cycle is a hit and the write protection indicator is set, the cache will not be updated. In all other cases, the write protection indicator is ignored.

6.1.4 NON-CACHEABLE READ CYCLE

Non cacheable read cycles are terminated on the System Bus one clock before they are terminated on the Local Bus.

The following Figures (6.6 - 6.7) include waveforms of several cases of non cacheable read cycles.

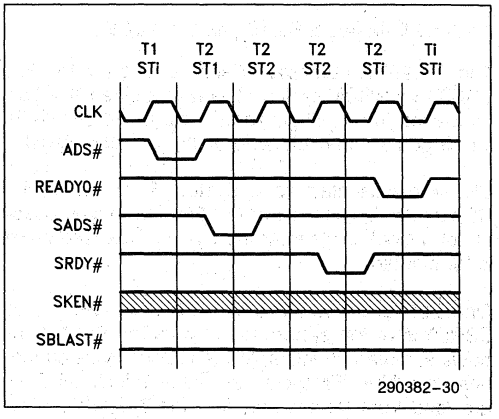
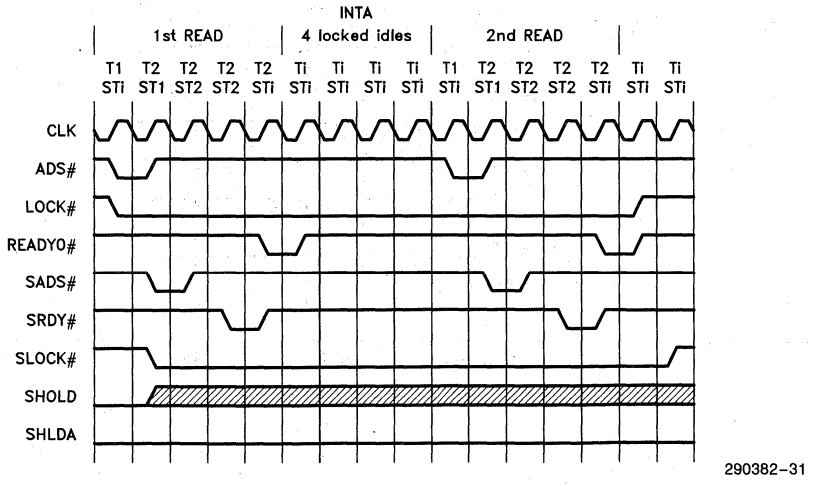


Figure 6.6 - I/O Read Cycle



NOTE:
While SLOCK# is active SHOLD input is ignored
Even if the System Bus is in its idle state, SLOCK# is active because LOCK# is active.

Figure 6.7 - INTA LOCK#ed Cycle

6.1.5 CACHEABLE READ MISS CYCLES

The 82395DX attempts to start a Line Fill for non LOCK#ed CRDM cycles. However, a Line Fill will be converted into a single read cycle if the access is indicated as non-cacheable by the SKEN# signal.

CRDM cycles start as a System Bus read cycle. READYO# is returned to the 386 DX Microprocessor one clock cycle after the System Bus read cycle is terminated.

One CLK cycle before the first SNA#, SRDY# or SBRDY# of the system read cycle, the SKEN# input is sampled. If active, the read miss cycle continues as a Line Fill cycle, and three additional DWs are read from the memory into the 82395DX. Also, the SWP# input will be sampled with the first SNA#, SRDY# or SBRDY# so the WP flag of the line will be updated in the Cache Directory.

6.1.5.1 Aborted Line Fill (ALF) Cycles

The System Bus can respond that the area of memory included in a particular request is non-cacheable, by returning SKEN# inactive. As soon as the 82395DX samples SKEN# inactive, it converts the cycle from a cache Line Fill, which requires additional read cycles to be completed, to a single cycle.

In this case SBLAST# will stay active. Also, the 82395DX will not generate another system cycle for the same Line Fill, because the cycle has already been finished by the first SBRDY# or SRDY# after SKEN# was sampled inactive.

The following Figure 6.8 includes waveforms of an ALF cycle.

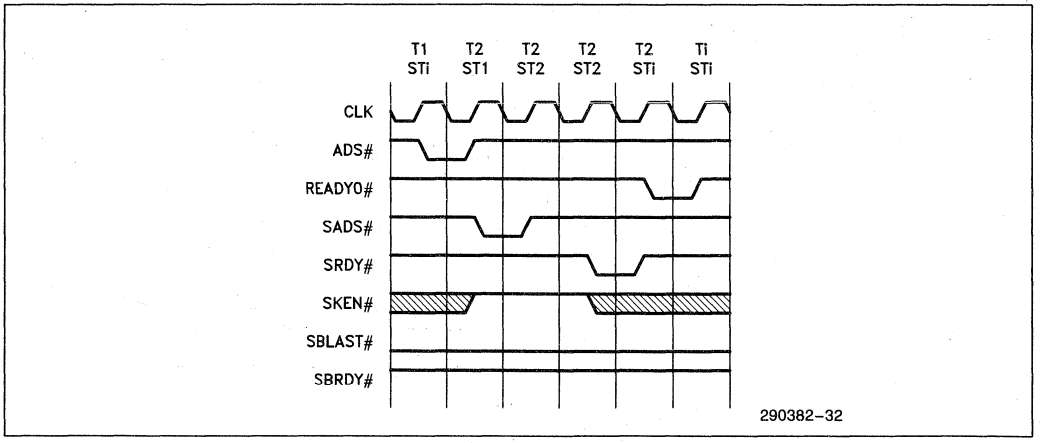


Figure 6.8 - Aborted Line Fill cycle

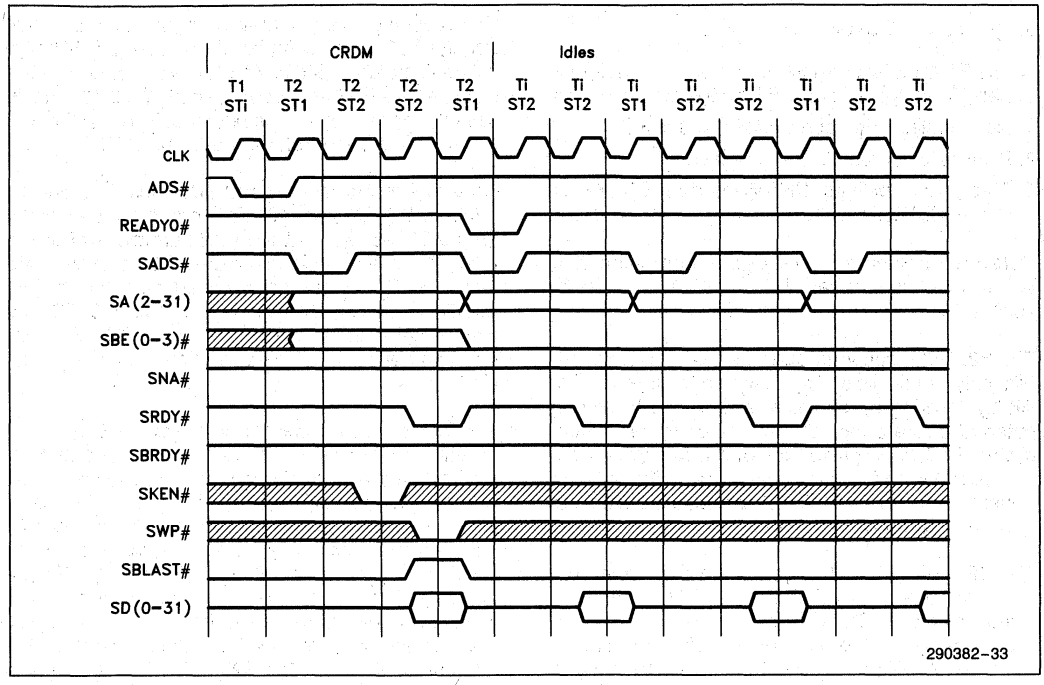


Figure 6.9 - Line Fill Without Burst or Pipeline

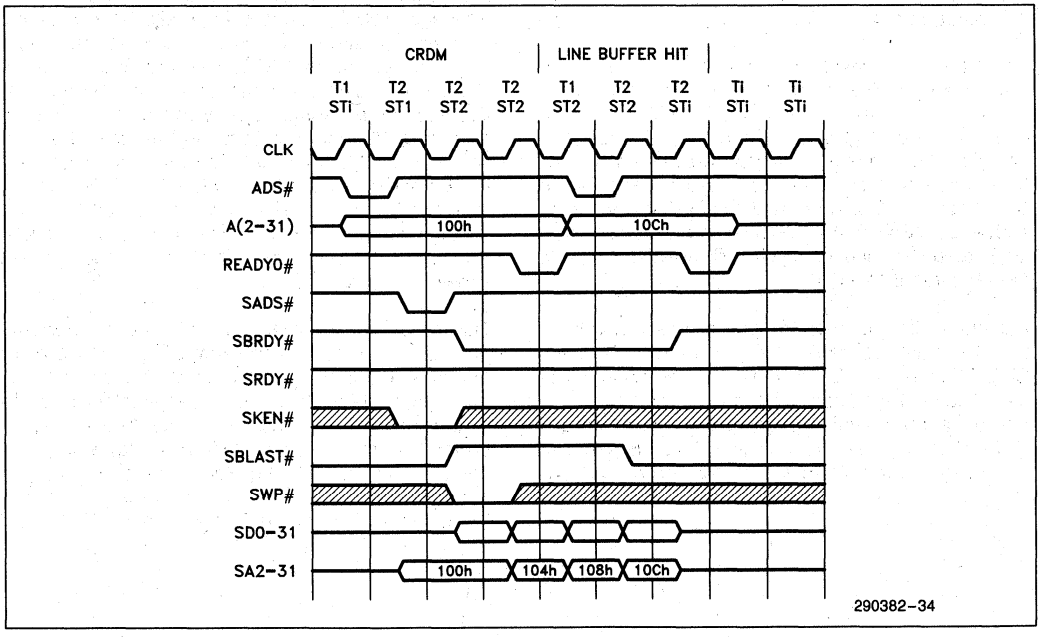


Figure 6.9A - Burst Mode Line Fill followed by a Line Buffer Hit Cycle

5

6.1.5.2 Line Fill Cycles

A Line Fill transfer consists of four back to back read cycles. Three types of Line Fill cycles are supported:

1. Non pipeline, Non burst, SNA# inactive.
2. Pipelined, non burst, SNA# active.
3. Burst, non pipelined, SNA# inactive, SRDY# inactive, SBRDY# active.

Note that a pipelined burst cycle is not supported. When SNA# is sampled active, SBRDY# is treated as SRDY#.

The 82395DX supports burst cycles in system Line Fills only. Burst cycles are designed to allow fast line fills by allowing consecutive read cycles to be executed at a rate of one DW per clock cycle. In burst cycles SADS# is pulsed for one clock cycle while the address and control lines are valid until the transfer is completed. SA2–3 are updated every bus cycle during the burst transfer.

The 82395DX starts the Line Fill as a normal read cycle, and waits for SBRDY# or SRDY# to be returned active. If SNA# is sampled active at least one clock cycle before either SBRDY# or SRDY#, the Line Fill will be non burst pipelined. (See Figure 6.10). If SNA# is sampled active at the same clock cycle as SBRDY# or SRDY#, the line fill will be non-burst, non-pipelined.

If SKEN# is sampled inactive one clock before either SNA#, SBRDY# or SRDY#, then the access is considered non-cacheable and Line Fill will not be executed. (See Figure 6.8) Otherwise, if SRDY# is sampled active, the line fill cycle resumes as a non-burst sequence of three more cycles (see Figure 6.9). Finally, if SBRDY# and SKEN# are sampled active (and SNA# and SRDY# are sampled inactive), then the Line Fill cycle will be a burst cycle (see Figures 6.11 - 6.12).

If a system cannot support burst cycles, a non burst line fill must be requested by merely returning SRDY# instead of SBRDY#, in the first read cycle (see Figure 6.9). Once a burst cycle started, it will not be aborted until it's completed, regardless if SKEN# is sampled inactive or SHOLD is sampled active, i.e. all four DWs will be read from memory. However, the system may abort a burst Line Fill transfer before it's completed, by returning SRDY# active (instead of SBRDY#) for the second or third DW in a Line Fill transaction (see Figure 6.13). In this case the cache will not be updated. The first DW will already have been transferred to the CPU.

Note that in the last (fourth) bus cycle in a line fill transfer, SBRDY# or SRDY# has the same effect on the 82395DX. That is to indicate the end of the Line Fill. For all cycles that cannot run in burst mode (non-Line Fill cycles or pipelined Line Fill cycles) SBRDY# has the same effect on the 82395DX as the normal SRDY# pin. SRDY# and SBRDY# are the same apart from their function during burst cycles.

The fastest burst cycle possible requires two clocks for the first data item to be returned to the 82395DX with subsequent data items returned every clock. Such a bus cycle is shown in Figure 6.11. An example of a burst cycle where two clocks are required for every burst item is shown in Figure 6.12. When initiating any read, the 82395DX presents the address for the data item requested. When the 82395DX converts this cycle into a cache Line Fill, the first data item returned must correspond to the address sent out by the 82395DX. This address is the original address that is requested by the 386 DX Microprocessor. The 82395DX updates this address after each SBRDY# according to table 6.1 (SA2 and SA3 are updated). This is also true for non-burst Line Fill cycles. The 82395DX presents each request for data in an order determined by the first address in the transfer. For example, if the first address was 104, the next three addresses in the burst will be 100, 10C, and 108. The burst order used by the 82395DX is shown in Table 6.1. This remains true whether the external system responds with a sequence of normal bus cycles or with a burst cycle. An example of the sequencing of burst addresses is shown in Figure 6.12.

This order was designed to optimize the performance of 64- bit memory systems. The second cycle of a burst reads the DW that forms the other half of an aligned 64-bit block, no matter whether that DW is at a higher or lower address. The third and fourth cycles then read the two DWs which form the other half of an aligned 128-bit block. The order in which the third and fourth DWs are accessed corresponds to the order used for the first and second DWs.

Table 6.1 - Line Fill Address Order

First Address	Second Address	Third Address	Fourth Address
0	4	8	C
4	0	C	8
8	C	0	4
C	8	4	0

In the following cases, a Line Fill cycle will not update the cache:

1. Aborted burst: burst cycle will be aborted if SRDY# is returned active in the second or third bus cycle. The Line Fill will not resume, and the cache will not be updated.
2. Snoop hit to line buffer: If, during a Line Fill transfer, a snoop cycle is initiated after the first SRDY# or SBRDY#, and the address matches the address of the line being retrieved, the Line Fill cycle will continue as usual but the cache will not be updated.
3. FLUSH during Line Fill cycle: the Line Fill cycle will continue as usual, but the cache will not be updated.

Figures (6.9 - 6.13) include waveforms of several cases of Line Fill cycles.

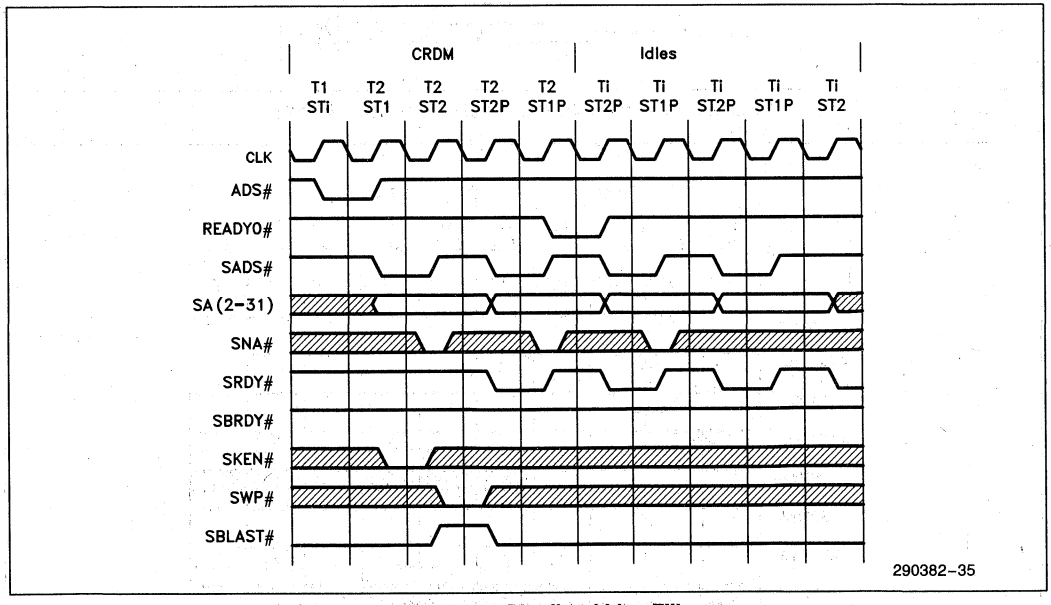


Figure 6.10 - Pipelined Line Fill

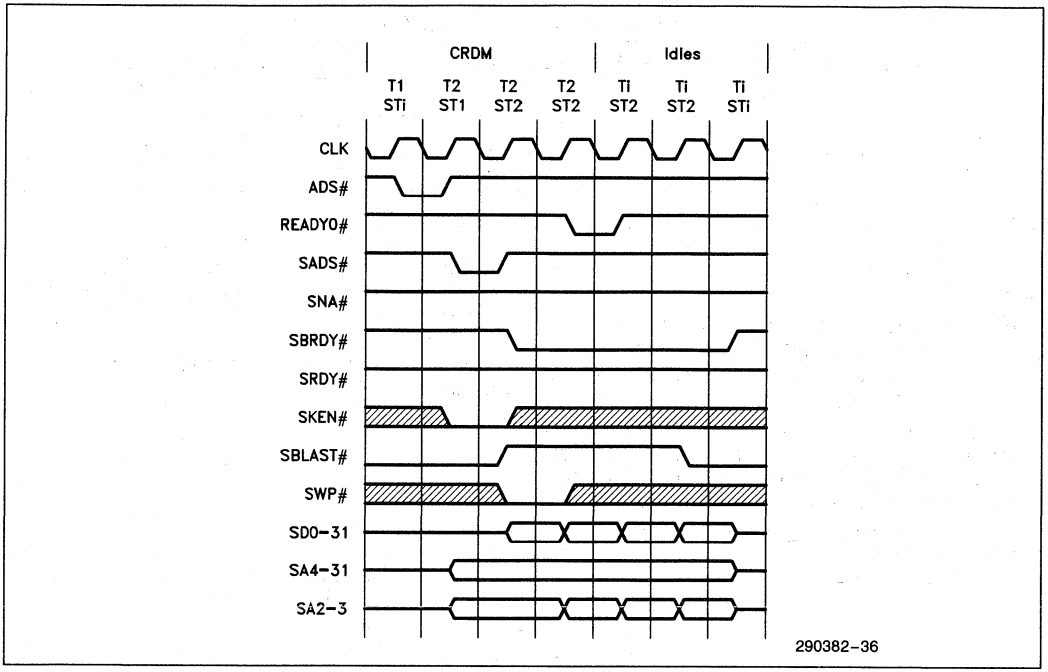


Figure 6.11 - Fastest Burst cycle (one clock burst)

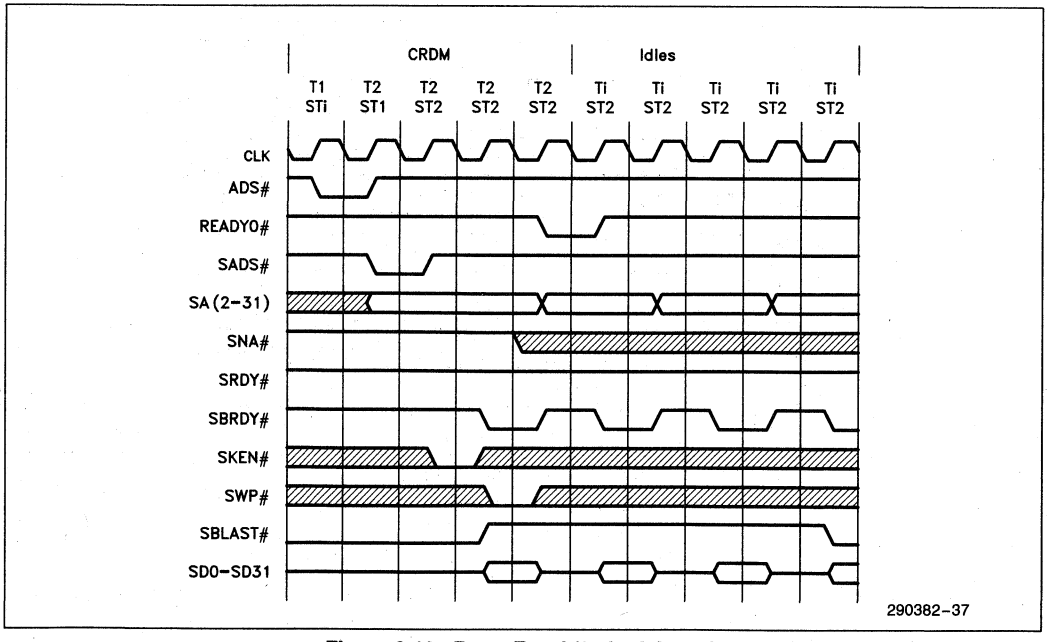


Figure 6.12 - Burst Read (2 clock burst)

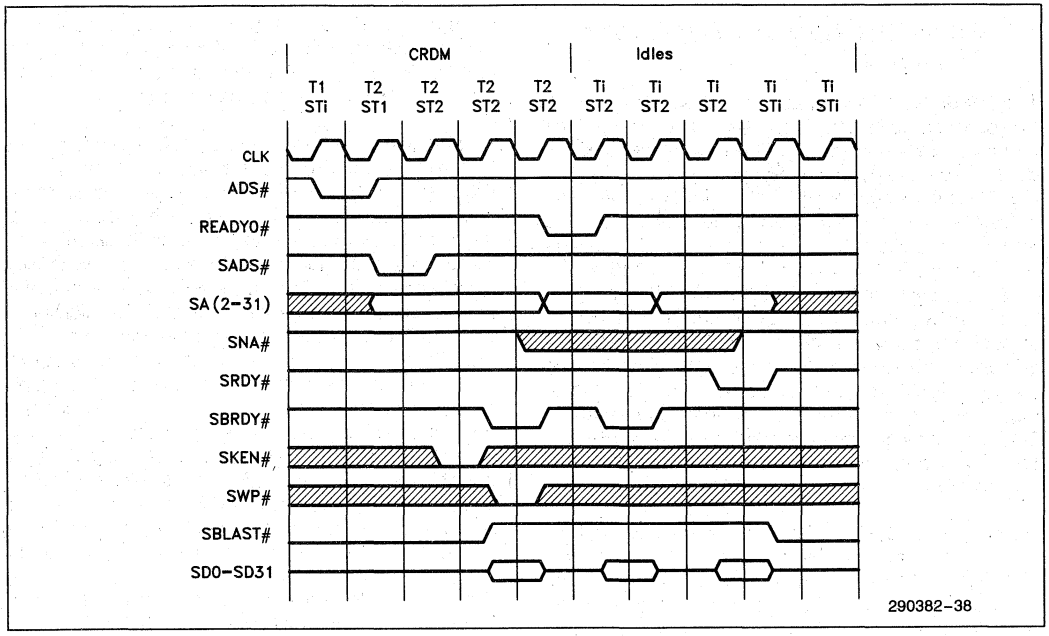


Figure 6.13 - Interrupted Burst Read (2 clock burst)

5

6.2 82395DX Latency in System Bus Accesses

The 82395DX acts as a buffer between the 386 DX Microprocessor and the main memory causing some latency in initiating the System Bus cycle (SADS# delay from ADS#) and in completing the cycle (386 READYO# delay from SRDY# or SBRDY#). The 82395DX drives the SADS# one clock after the ADS#. In cacheable cycles, the 82395DX starts driving the SADS# before it decides whether the cycle is a cache hit or miss since the hit/miss decision is valid in the second clock (the first T2 cycle). In case the cycle is a hit, the 82395DX deactivates SADS#. This causes an undesirable glitch on the SADS# signal, and also it causes an SADS# timing incompatibility with the 386 DX Microprocessor i.e. SADS# delay is slightly longer than the ADS# delay. For proper system functionality, SADS# must be sampled by the next clock edge.

At the end of System Bus non- cacheable read cycle, or non- buffered write cycle, the 82395DX drives READYO# active one clock after SRDY#. In a Line Fill cycle, READYO# is activated one clock after the first SBRDY# or SRDY# is sampled active. The setup timing requirements of SRDY# and system data force one wait state at the end of the cycle.

6.3 SHLDA Latency

For non-LOCK#ed cycles the worst case delay between SHOLD and SHLDA would be when SHOLD is activated during ST2P state, followed by a Line Fill. In this case, the HOLD request will be acknowledged only after the Line Fill is completed. In LOCKed cycles SHLDA will not be asserted until after SLOCK# is negated. The latency would be:

Latency = (Number of ST2Pcycles) + (Number of Line Fill cycles) OR (Number of LOCK#ed cycles)

6.4 Cache Consistency Support

The 82395DX supports snooping using the SEADS# mechanism. Besides insuring the consistency, this mechanism provides multi processing support by having the 82395DX System Bus and the Local Bus running concurrently.

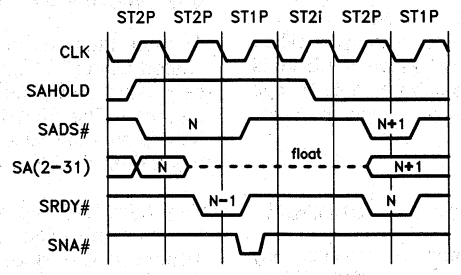
The 82395DX will always float its address bus in the clock immediately following the one in which SAHOLD is received. Thus, no address hold acknowledge is required. When the address bus is floated, the rest of the 82395DX's System Bus will remain active, so that data can be received from a bus cycle that was already underway. Another bus cycle will not begin, and the SADS# signal will not be generated. However, multiple data transfers for burst cycles can occur during address holds.

A companion input to SAHOLD, SEADS# indicates that an external address is actually valid on the address inputs of the 82395DX. When this signal is activated, the 82395DX will read the external address and perform an internal cache invalidation cycle to the address indicated. The internal invalidation cycle occurs one clock after SEADS# is sampled active. In case of contention with 386 DX Microprocessor look up, the invalidation is serviced two clocks after SEADS# was activated. The maximum rate of invalidation cycles is one every other clock. Multiple cache invalidations can occur in a single address hold transfer. SEADS# is not masked by SAHOLD inactive, so cache invalidations can occur during a normal bus cycle. This also means that if SEADS# is driven active when the 82395DX is driving the address bus, the values that are being driven by the 82395DX will be used for a cache invalidation cycle.

If the 82395DX is running a line fill cycle and an invalidation is driven into the 82395DX in the same clock the first data is returned, or in any subsequent clock, the 82395DX will invalidate that line even if it is the same cache line that the 82395DX is currently filling.

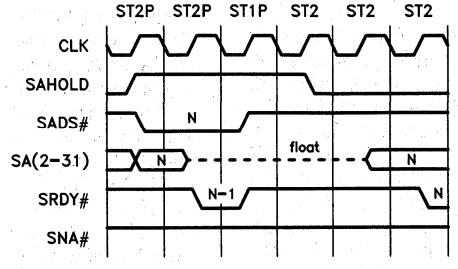
SAHOLD in pipelined cycles: The activation of SAHOLD only causes the system address to be floated in the next clock without changing the behavior of pipelined cycles. If SAHOLD is activated before entering the ST2P state, the 82395DX will move into non-pipeline and drive the SADS# only after the deactivation of SAHOLD. However, if SAHOLD is asserted in the ST2P state and the Nth cycle has already started, the system address is floated but SADS# is kept active until SRDY# (for the N-1 th cycle) is returned. It is the system designers' responsibility to latch the address bus. Note that the address driven on the System Bus after SAHOLD is deasserted (in pipelined cycles) depends on whether SNA# has been sampled active during the SAHOLD state and another cycle is pending. As seen from Figure 6.14, the (N+1)th address will be driven by the 82395DX once SAHOLD was deactivated and SNA# was sampled active, provided there is a cycle pending in the 82395DX. The following figures describe the 82395DX behavior in two cases. First, when SNA# is sampled active and second, in the case of SNA# sampled inactive.

Note that the maximum rate of snooping cycles is every other clock. The first clock edge in which SEADS# is sampled active causes the 82395DX to



290382-39

(A) - SNA# sampled active



290382-40

(B) - SNA# sampled inactive

Figure 6.14 - SAHOLD Behavior in Pipelined Cycles

5

latch the system address bus and initiate a cache invalidation cycle. If SEADS# is driven active for more than one clock, only one snooping cycle will be initiated on the first clock edge at which SEADS# is sampled active. The SA2–31 setup and hold timings are specified to the same clock edge in which SEADS# is sampled active.

6.5 Bus Deadlock Resolution Support

In a multi-master system another bus master may require the use of the bus to enable the 82395DX to complete it's current bus request. In this situation, the 82395DX will float it's entire System Bus until the other bus master has completed it's bus transaction.

The 82395DX will float it's System Bus immediately in response to the external system asserting the Fast HOLD (SFHOLD#) signal. The only effect of this signal being sampled active is forcing the 82395DX System Bus pins to float. It is the system designer's responsibility to ensure that no 82395DX cycle is prematurely terminated, and that no new 82395DX cycle is generated during Fast HOLD. When SFHOLD# is deasserted the System Bus address, cycle definition and data are redriven by the 82395DX and the cycle is not restarted. SRDY# and SBRDY# are not recognized during SFHOLD# states. SFHOLD# asserted internally disables SRDY# and SBRDY#.

6.6 Arbitration Mechanism

As more than one device may be connected to the shared system bus, there is a need for arbitration between the devices that wish to utilize the shared resource. The 82395DX supplies the interface signals to an external arbiter (either centralized or distributed) to enable it to perform the task.

The 82395DX provides a normal bus SHOLD/SHLDA handshake protocol, exactly as the 386 DX Microprocessor does on the Local Bus. SHOLD is used to indicate to the 82395DX that another bus master desires control of the 82395DX System Bus. Whenever the 82395DX completes its current bus cycle (a full line transfer if the cycle is a Line Fill), or sequence of LOCK#ed bus cycles, it will grant its external bus to the requesting device by floating it and by driving SHLDA active. The 82395DX will relinquish its System Bus at the end of a bus cycle, even if it has other cycles internally pending. As soon as the 82395DX responds with SHLDA, it tristates all bus control and address outputs. Now, if the System Bus is required by the 82395DX (on behalf of a 386 DX Microprocessor request on the Local Bus) but is not available, processing will cease. Then the 82395DX will have to re-arbitrate on the System Bus by driving SBREQ active.

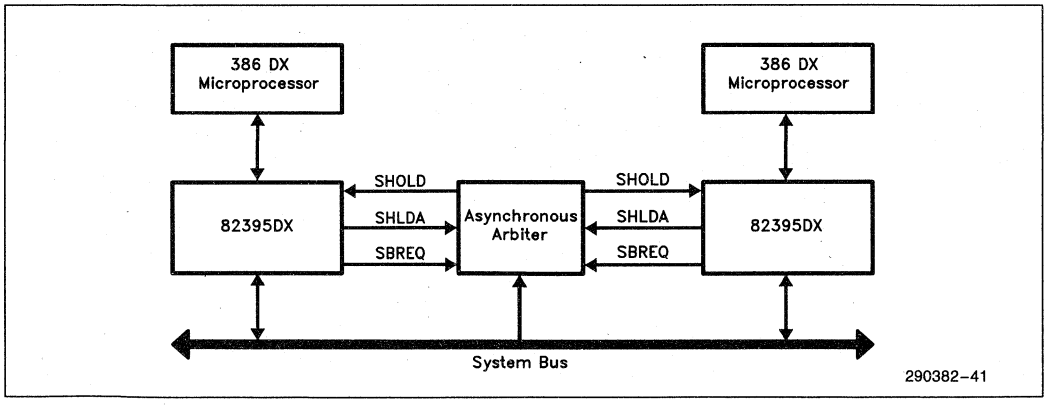


Figure 6.15 - Multiple 82395DX Bus Arbitration Scheme

The SBREQ output is activated whenever the 82395DX has internally generated a bus cycle request. It is inactivated immediately after the 82395DX asserts SADS# of the cycle. By examining this signal, external logic can determine when the 82395DX requires the use of the System Bus and intelligently arbitrate the System Bus among multiple processors. This pin is always driven, regardless of the state of bus hold (See Figure 6.16).

The SHOLD input has higher priority than the pending request. In the case of LOCK#ed System Bus cycles, SHOLD requests will not be acknowledged. Another case is a non-burst Line Fill, where SHOLD is acknowledged after reading the fourth DW, even though SHOLD was activated before.

6.7 Next Near Cycles

For all System Bus cycles, the 82395DX generates a signal, SNENE#, to indicate that the current cycle is in the same 2048 Byte area as the previous memory cycle. Namely, it indicates that address lines A11–A31 of the current System Bus memory cycle are identical to address lines A11–A31 of the previous memory cycle. This signal can be used by an external DRAM system to run CAS# only cycles, therefore increasing the throughput of the memory system. SNENE# timing is identical to system address timing, namely it is valid from SADS# active until SRDY# or SBRDY# is sampled active (non-pipelined cycles) or until SNA# is sampled active (pipelined cycles). SNENE# is valid for all memory cycles, and must be ignored in I/O and idle cycles.

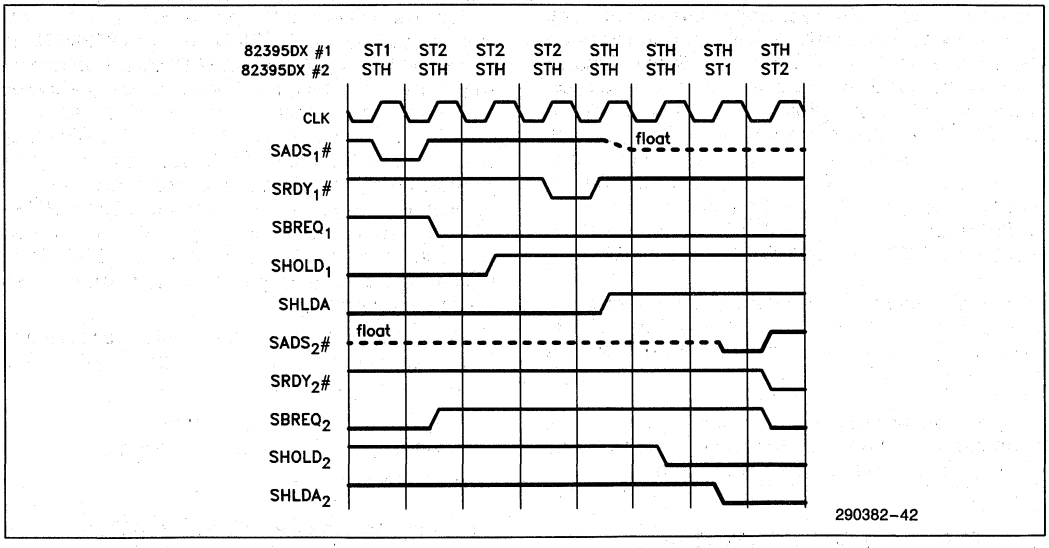


Figure 6.16 - SHOLD/SHLDA/SBREQ Mechanism

After the 82395DX exits the SHOLD state, SNENE# is always inactive. SNENE# is always inactive in the first memory cycle after a Halt/Shutdown cycle.

If SAHOLD is sampled active while the System Bus is idle, the next 82395DX cycle will have SNENE# inactive. If SAHOLD is sampled active while the 82395DX is running a System Bus cycle, SNENE# will not change until the next SADS# is issued. During SHLDA, SNENE# is floated and the first cycle after SHLDA is deactivated will have SNENE# inactive. SNENE# can run in the pipeline, the same as the system address.

6.8 Write Buffer

The 82395DX is able to internally store up to four write cycles (address, data and status information). All those write cycles will run without wait states on the Local Bus. They will run on the System Bus as soon as the bus is available. In case of a write cycle which cannot be stored since the buffer is full, the 386 DX Microprocessor will be forced to wait until one of the buffered write cycles is completed. READYO# is returned two CLK's after SRDY# or SBRDY# is asserted if the write buffer is full. If the write buffer is not full READYO# is returned one clock after SRDY# or SBRDY# is asserted.

All non cacheable write cycles and LOCK#ed writes are not buffered. In this case, the 82395DX will activate READYO# after getting the SRDY# for the non buffered cycle.

The write buffer maintains the exact original order of appearance of the Local Bus requests. It allows no reordering and no bypassing of any sort.

7.0 TESTABILITY FEATURES

This chapter discusses the requirements for properly testing an 82395DX based system after power up and during normal system operation.

7.1 SRAM Test Mode

This mode is invoked by driving the FLUSH# pin active for less than four clocks during normal operation. SRAM test mode may only be invoked when the 82395DX is in idle state, namely there is no cycle in progress, and no cycle is pending in the 82395DX. The 82395DX exits this mode with subsequent activation of the FLUSH# pin for minimum of 1 clock cycle. If FLUSH# is activated for at least eight clock cycles during SRAM test mode, the 82395DX will FLUSH# its cache directory in addition to terminating the SRAM test mode.

SRAM test mode is provided for system diagnostics purposes. In this mode, the 82395DX cache and cache directory are treated as a standard SRAM. The 82395DXs in the system are mapped into address space 256K-512K of the 386 DX Microprocessor memory space, and allows the CPU non-cacheable, non-buffered access to the rest of the memory and address space. Each 82395DX occupies 32KB of address space: 16KB for the cache and 16KB (not fully utilized) for the TAGRAM. The 82395DX, in SRAM mode, will recognize 387 DX Math Coprocessor/Weitek 3167 Floating-Point Coprocessor cycles and Local Bus cycles and handle them the same as it does in its normal mode. This way, the CPU may execute code that tests the 82395DX as a regular memory component, with the only limitation that no code or data may reside in the memory space 256K-512K during this mode. During SRAM test mode, all accesses to memory space other than 256K-512K are handled exactly as in normal mode with the following exceptions:

1. All read cycles are non-cacheable - read hits are not serviced from the cache and read misses don't cause Line Fills.
2. All write cycles are not buffered.
3. All write cycles do not update the cache.
4. Snooping is disabled.

The local address pins indicate the 82395DX internal addresses. The partitioning is as follows:

- A16=0 selects the cache directory. A16=1 select the cache.
- A15–14 select the "way".
- A12 and A13 select one 82395DX in a multi 82395DX system.
- A11–A4 are the set address.
- A3–2 select a DW in the line. Applicable in cache accesses (A16=1).

The user can write to any byte in any line in case of a cache write cycle and write to all the Tagram fields (25 bits) in one Way in one Tagram write cycle. The memory mapping of the SRAM mode is the described in Table 7.1.

As can be seen from table 7.1, the address space allocated for either Tagram or Cache is 4096 (4K) addresses per way, per 82395DX. The address allocation within each 4K segment is shown in tables 7.2 and 7.3.

The data presented on the 82395DX local data pins is the SRAM data input. The SRAM data output is also driven on the local data pins. The BE(0-3)# pins indicate the bytes which must be written. During SRAM test mode, all the AC specifications are met. Figures 7.1 and 7.2 depict the SRAM mode read and write cycles respectively. Note that two wait states are inserted during SRAM test mode read cycles and one wait state is inserted in write cycles. The system may extend the number of wait states by gating READYO# for any number of clock cycles (1 clock cycle in Figure 7.1, 0 clock cycles in Figure 7.2).

The user can write to any byte in any line in case of a cache write cycle and write to all the Tagram fields (25 bits) in one way in one Tagram write cycle. The memory mapping of the SRAM test mode described in table 7.1.

Table 7.1 - SRAM Memory Map

Cache/Tagram	Way	82395DX	Start Address
Cache	3	4	0005F000 h
Cache	3	3	0005E000 h
Cache	3	2	0005D000 h
Cache	3	1	0005C000 h
Cache	2	4	0005B000 h
Cache	2	3	0005A000 h
Cache	2	2	00059000 h
Cache	2	1	00058000 h
Cache	1	4	00057000 h
Cache	1	3	00056000 h
Cache	1	2	00055000 h
Cache	1	1	00054000 h
Cache	0	4	00053000 h
Cache	0	3	00052000 h
Cache	0	2	00051000 h
Cache	0	1	00050000 h
Tagram	3	4	0004F000 h
Tagram	3	3	0004E000 h
Tagram	3	2	0004D000 h
Tagram	3	1	0004C000 h
Tagram	2	4	0004B000 h
Tagram	2	3	0004A000 h
Tagram	2	2	00049000 h
Tagram	2	1	00048000 h
Tagram	1	4	00047000 h
Tagram	1	3	00046000 h
Tagram	1	2	00045000 h
Tagram	1	1	00044000 h
Tagram	0	4	00043000 h
Tagram	0	3	00042000 h
Tagram	0	2	00041000 h
Tagram	0	1	00040000 h

As can be seen from Tables 7.2 and 7.3, the address space allocated for either Tagram or Cache is 4096 (4K) addresses per way, per 82395DX. The address allocation within each 4K segment is shown in table 7.2 for the Cache and table 7.3 for the Tagram.

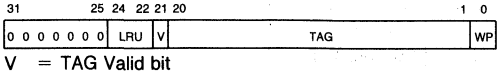
Table 7.2 - Cache Address Allocation

SET	DW	Start Address
255	3	FFC h
255	2	FF8 h
255	1	FF4 h
255	0	FF0 h
.	.	.
.	.	.
.	.	.
1	3	01C h
1	2	018 h
1	1	014 h
1	0	010 h
0	3	00C h
0	2	008 h
0	1	004 h
0	0	000 h

Table 7.3 - TAGRAM Address Allocation

SET	Start Address
255	FFC h
255	FF8 h
255	FF4 h
255	FF0 h
.	.
.	.
.	.
1	01C h
1	018 h
1	014 h
1	010 h
0	00C h
0	008 h
0	004 h
0	000 h

Double Word format in Tagram read/write:



V = TAG Valid bit
WP = Write Protect bit
"0" = Indicates don't care bits. Writing to these bits will have no effect. When reading the Tagram these bits will have a value of 0.

NOTE:
In Tagram accesses, BE0#-BE3# are ignored in both read and write cycles.

The data presented on the 82395DX D0–D31 pins is the SRAM data input for write cycles and is also the SRAM data output for read cycles during the SRAM test mode. The BE3# –BE0# pins indicate the bytes which will be written to. During SRAM test mode all the AC specifications are met. Figures 7.1 and 7.2 depict the SRAM test mode read and write cycles respectively. The system may extend the number of wait states by gating READYO# for any number of clock cycles (one clock cycle in Figure 7.1, zero in Figure 7.2).

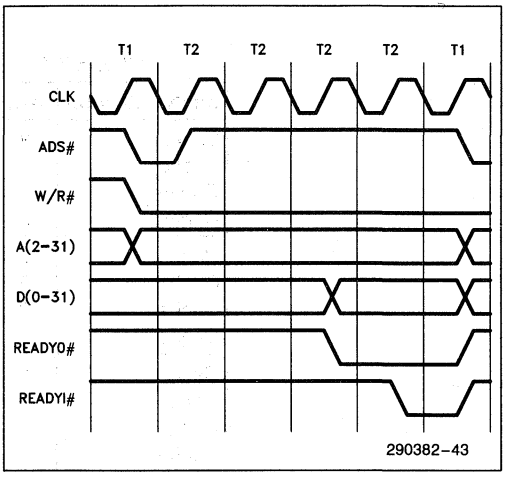


Figure 7.1 - SRAM Mode Read Cycle

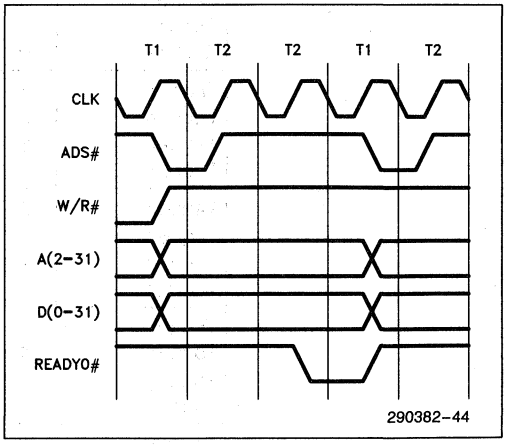


Figure 7.2 - SRAM Mode Write Cycle

7.2 Tristate Output Test Mode

The 82395DX provides the option of isolating itself from other devices on the board for system debugging, by floating all it's outputs. Output tristate mode is invoked by driving the SAHOLD and FLUSH# pins active during RESET. The 82395DX will remain in this mode after RESET is deactivated, if SAHOLD and FLUSH# pins are sampled active in the CLK2 prior to RESET going low (See Figure 7.3). The 82395DX exits this mode with the next activation of RESET with SAHOLD or FLUSH# driven inactive.

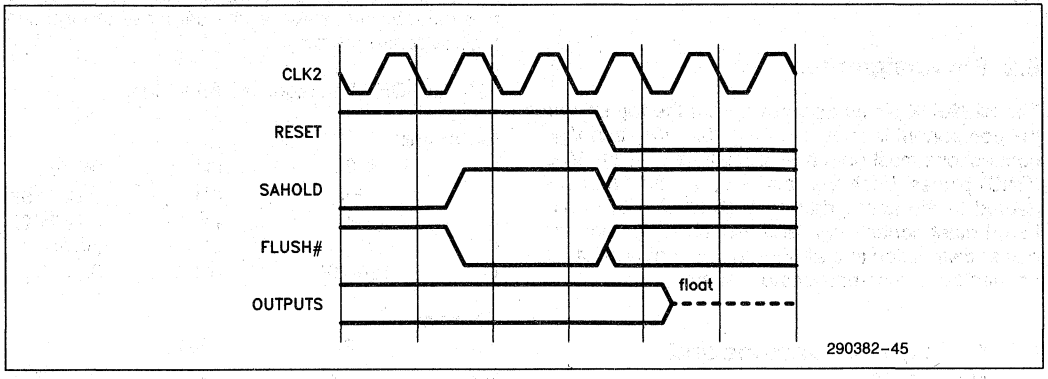


Figure 7.3 - Entering the Tristate Test Mode

8.0 MECHANICAL DATA

8.1 Introduction

This chapter discusses the physical package and its connections.

8.2 Pin Assignment

The 82395DX pinout as viewed from the top side of the component is shown in figure 0.1. V_{CC} and V_{SS} connections must be made to multiple V_{CC} and V_{SS} (GND) planes. Each V_{CC} and V_{SS} pin must be connected to the appropriate voltage level. The circuit board must contain V_{CC} and V_{SS} (GND) planes for power distribution and all V_{CC} and V_{SS} pins must be connected to the appropriate planes.

8.3 Package Dimensions and Mounting

The 82395DX package is a 196 lead plastic quad flat pack (PQFP). For information on dimensions refer to Table 8.1 and Figures 8.1–8.3.

8.4 Package Thermal Specification

The 82395DX is specified for operation when the case temperature is within the range of 0–85 °C. The case temperature may be measured in any environment, to determine whether the 82395DX is within the specified operating range. The case temperature must be measured at the center of the top surface opposite the pins.

196 Pin PQFP Package Key Attributes:

Electrical:

L	6-20	nH	(lead)
L	3-6	nH	(V_{CC}/V_{SS})
C	<2.3	pF	(Loading)
C	<1.6	pF	(Id/Id)
C	130-200	nH	(V_{CC}/V_{SS})

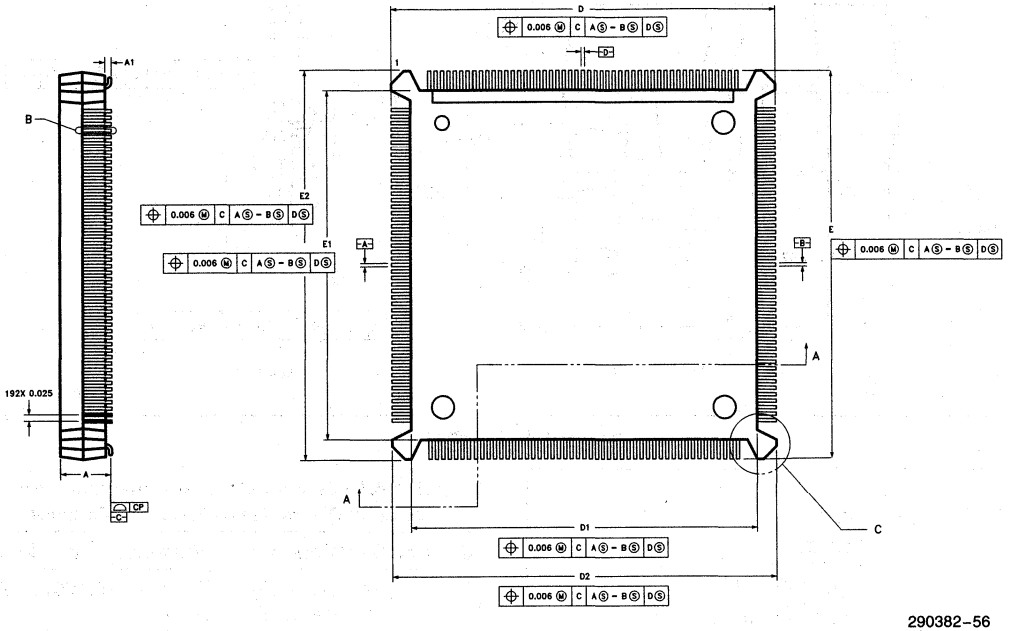
Thermal:

Θ_{ja}	24	°C/W	@2W
Θ_{jc}	5	°C/W	@2W

Lead Stiffness:

In-Plane	17	gm/mil
Transverse	18	gm/mil

Thermal characterization of the 196 lead PQFP package yielded the information contained in Figures 8.4–8.6.



NOTES:
1. Interpret dimensions and tolerances in accordance with ANSI Y14.5M–1982.
2. Data enclosed in parentheses is for reference only.

Figure 8.1 - Principal Dimensions and Data

5

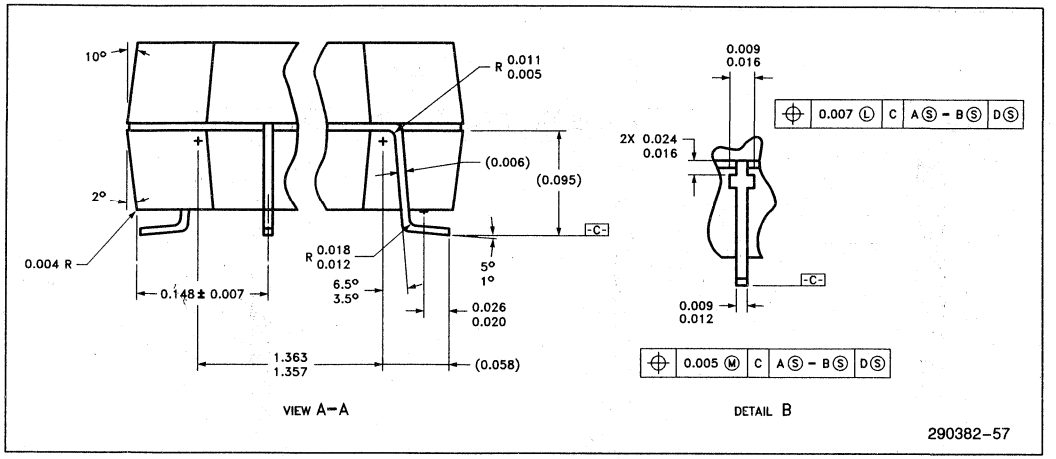


Figure 8.2 - Typical Lead

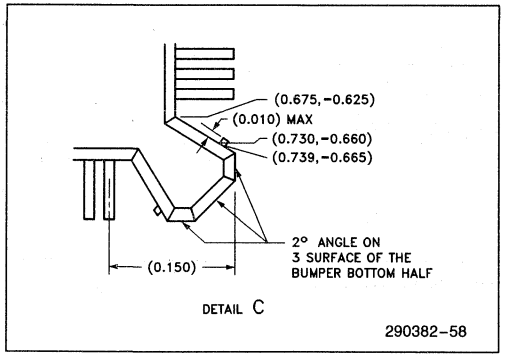


Figure 8.3 - Detail C

Table 8.1 - Symbol List and Dimensions for 196 Lead Plastic Quad Flat Pack Package

Symbol	Description of Dimensions	Min	Max
A	**Package Height:** Distance from the seating plane to the highest point of body.	0.160	0.175
A1	**Standoff:** The distance from the seating plane to the base plane.	0.020	0.035
D, E	**Overall Package Dimension:** Lead tip to lead tip.	1.470	1.485
D1, E1	**Plastic Body Dimension**	1.347	1.353
D2, E2	**Bumper Distance**		
	Without FLASH	1.497	1.503
	With FLASH	1.497	1.510
CP	**Seating Plane Coplanarity**	0.000	0.004

NOTES:
1. All dimensions and tolerances conform to ANSI Y14.5M-1982.
2. Dimensions are in inches.
3. Data enclosed in parenthesis is for reference only.

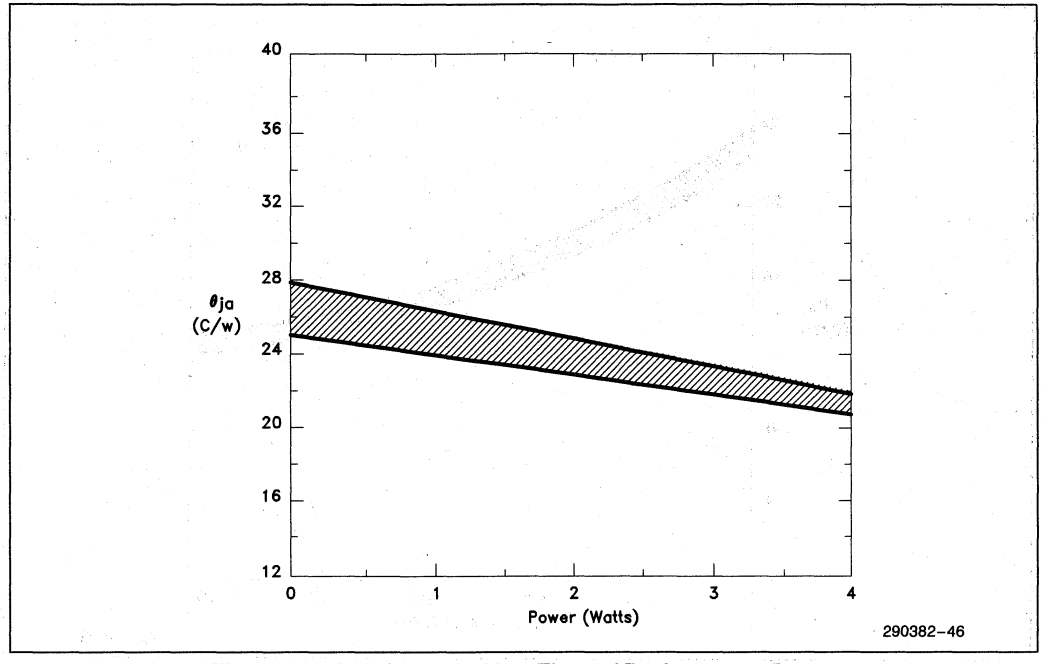


Figure 8.4 - Junction to Ambient Thermal Resistance vs Power

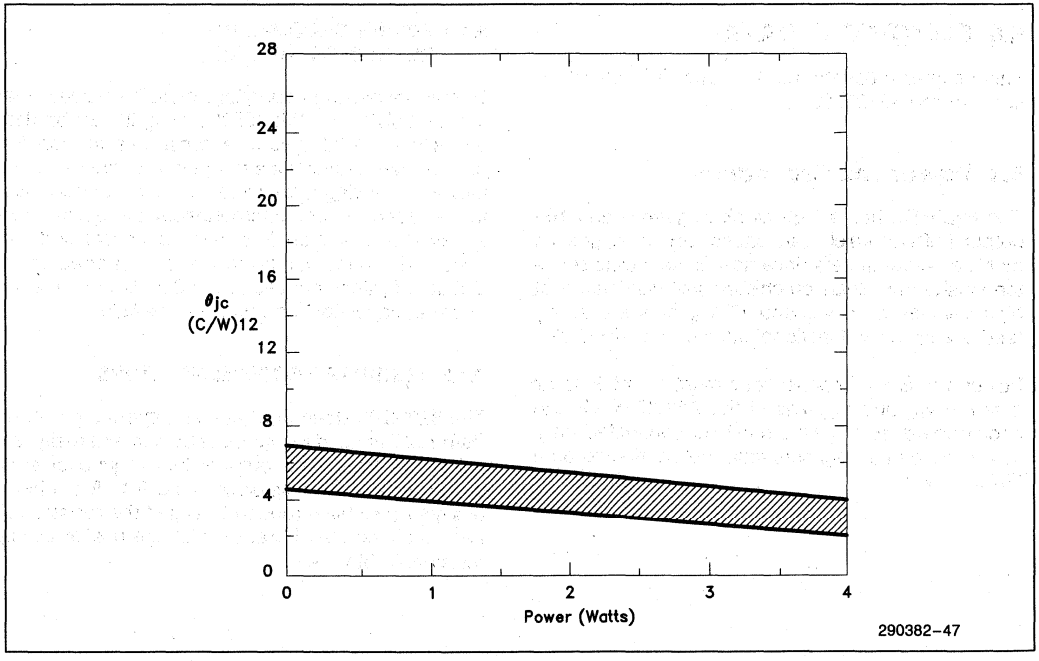


Figure 8.5 - Junction to Case Thermal Resistance vs Power

5

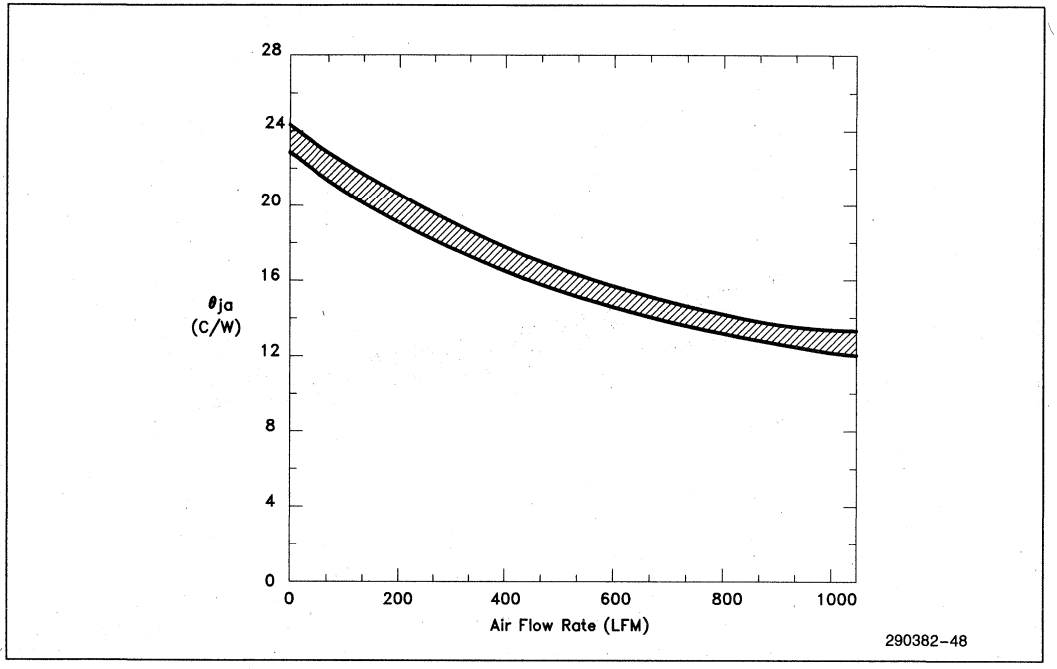


Figure 8.6 - Junction to Ambient Thermal Resistance vs Air Flow Rate

9.0 ELECTRICAL DATA

This chapter presents the A.C. and D.C. specifications for the 82395DX.

9.1 Power and Grounding

The 82395DX has a high clock frequency and 108 output buffers which can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 15 V_{CC} and 17 V_{SS} pins separately feed power to the functional units of the 82395DX.

Power and ground connections must be made to all external V_{CC} and V_{SS} pins of the 82395DX. On the circuit board, all V_{CC} pins must be connected on a V_{CC} plane and all V_{SS} pins must be connected on a GND plane.

9.1.1 POWER DECOUPLING RECOMMENDATIONS

Liberal decoupling capacitors must be placed near the 82395DX. The 82395DX driving it's 32 bit data buses and 30 bit system address bus at high frequency can cause transient power surges, particularly when driving large capacitive loads. Low inductance capacitors and interconnects are recommended for the best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the 82395DX and the decoupling capacitors as much as possible.

9.1.2 RESISTOR RECOMMENDATIONS

The 82395DX does not have any internal pullup resistors. All unused inputs must be tied externally to a solid logic level. The outputs that require external pullup resistors are listed in table 9.1. A particular designer may have reason to adjust the resistor values recommended here, or alter the use of pull-up resistors in other ways.

9.2 Absolute Maximum Ratings

Storage Temperature ..... −65 ° C to 150 ° C

Case Temperature under Bias ..... −65 ° C to 110 ° C

Supply voltage with Respect to V_{SS} ..... −0.5V to 6.5V

Voltage on Other Pins ..... −0.5V to V_{CC} +0.5V

NOTICE: This data sheet contains information on products in the sampling and initial production phases of development. The specifications are subject to change without notice.

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

Table 9.1 - Pullup Resistor Recommendations

Signal	Pullup Value	Purpose
READYO#	20K Ohms ±10%	Lightly pull READYO# inactive in multi-82395DX systems. Allows the selected 82395DX to drive READYO# while it is inactive for the others.
SADS#	20K Ohms ±10%	Lightly pull SADS# inactive in multi-82395DX systems. Allows the selected 82395DX to drive SADS# while it is inactive for the others.
SLOCK#	20K Ohms ±10%	Lightly pull SLOCK# inactive for 82395DX SHOLD states.

9.3 DC SPECIFICATIONS Tcase = 0°C to +85°C, Vcc = 5V ±5%

Table 9.2 - DC Specifications

Symbol	Parameter	Limits		Units	Test Conditions
		Min	Max		
VIL	Input Low Voltage	−0.3	0.8	V	
VIH	Input High Voltage	2.0	Vcc+0.3	V	
VCIL	CMOS Input Low	−0-.3	0.8	V	See Note 6
VCIH	CMOS Input High	Vcc−0.8	Vcc+0.3	V	See Note 6
VOL	Output Low Voltage		0.45	V	See Note 1
VOH	Output High Voltage	2.4		V	See Note 2
VCOL	CMOS Output Low Voltage		0.45	V	See Notes 1,7
VCOH	CMOS Output High Voltage	Vcc−0.45		V	See Notes 2,7
ILI	Input Leakage		±15	uA	OV<Vin<Vcc
ILO	Output Leakage		±15	uA	0.45V<Vout<Vcc
Cin	Cap. Input		10	pF	See Note 4
ICC	Power Supply Cur				See Note 3
	33MHz		550	mA	
	25MHz		470	mA	
	20MHz		430	mA	

NOTES:
1. This parameter is measured at IOL=4mA for all the outputs.
2. This parameter is measured at IOH=1mA for all the outputs.
3. Measured with inputs driven to CMOS levels, V_{CC} = 5.25 V, T_A = 0°C, using a typical pattern consisting of 33% read, write and idle cycles.
4. CLK2 input capacitance is 20pF.
5. No activity on the Local/System Bus.
6. Applies to CLK2, READYO# inputs.
7. Applies to READYO# output.

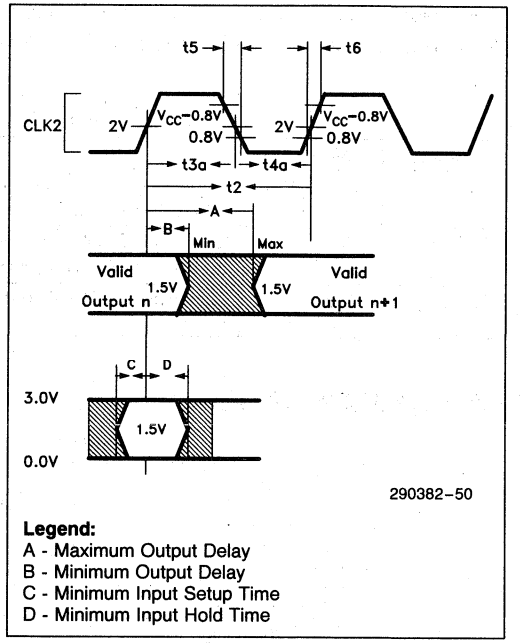
9.4 AC Characteristics

Some of the 82395DX AC parameters are clock-frequency dependent. Thus, while the part functions properly at the entire frequency range specified by the t1 spec, the AC parameters are guaranteed at three distinct frequencies only: 20MHz, 25MHz and 33MHz. Note that, for example, when a 33MHz part operates at 25Mhz CLK frequency, the AC parameters under "25MHz" column must be used.

- Functional operating range: VCC = 5V ±5%, Tcase = 0°C to +85°C.
- All AC parameters are measured relative to 1.5V for falling and rising, CLK2 is at 2V.
- All outputs tested at a 50pF load. In case of overloaded signals, the derating factor is 1ns for every extra 25pF load.
- All parameters are referred to PHI1 unless otherwise noted.
- The reference Figure of CLK2 parameters and AC measurements level is Figure 9.1 and RESET and internal phase is Figure 3.2.

9.4.1 TIMING CONSIDERATIONS FOR CACHE EXTENSIONS

The values listed in Tables 9.3 and 9.4 for the AC parameters are valid for a design using one 82395DX with its 16KB cache or two 82395DXs to extend the cache size to 32KB. For a design using four 82395DXs to extend the cache size to 64KB, some timing adjustments must be made due to the increased capacitive load on the signal traces. The capacitive derating curve (see Figure 9.6) must be used to accurately determine the impact on AC timings.



Legend:
A - Maximum Output Delay
B - Minimum Output Delay
C - Minimum Input Setup Time
D - Minimum Input Hold Time

Figure 9.1 - Drive Levels and Measurement Points for AC Specifications

9.4.2 AC CHARACTERISTICS TABLES Tcase = 0°C to 85°C, Vcc = 5V ±5%

Table 9.3 - Local Bus Signal AC Parameters

Symbol	Parameter	20 MHz		25 MHz		33 MHz		Units	Notes
		Min	Max	Min	Max	Min	Max		
t1	Operating Frequency	15.4	20	15.4	25	15.4	33	MHz	Internal CLK
t2	CLK2 Period	25	32.5	20	32.5	15	32.5	ns	
t3a	CLK2 High Time	8		7		6.25		ns	Measured at 2V
t3b	CLK2 High Time	5		5		4.5		ns	Measured at 3.7V
t4a	CLK2 Low Time	8		7		6.25		ns	Measured at 2V
t4b	CLK2 Low Time	6		5		4.5		ns	Measured at 0.8V
t5	CLK2 Fall Time		7		7		4	ns	Note 1
t6	CLK2 Rise Time		7		7		4	ns	Note 2
t7a	A2–A31 Setup Time	24		17		13		ns	
t7b	LOCK# Setup Time	12		11		9		ns	
t7c	BE0–3# Setup Time	18		14		13		ns	
t8	A2–A31, BE0–3#, LOCK# Hold Time	3		3		3		ns	
t9a	M/IO#, D/C#, W/R# Setup Time	20		17		13		ns	
t9b	ADS# Setup Time	23		17		13.5		ns	
t10	M/IO#, D/C#, W/R#, ADS# Hold Time	3		3		3		ns	
t11	READYI# Setup Time	12		9		7		ns	
t12	READYI# Hold Time	4		4		4		ns	
t13	LBA#, NPI# Setup Time	10		9		5.5		ns	Note 7
t14	RESET Setup Time	12		10		5		ns	
t15a	LBA#, NPI# Hold Time	3		3		3		ns	
t15b	RESET Hold Time	4		3		2		ns	
t16	D0–31 Setup Time	10		11		4		ns	Note 3
t17	D0–31 Hold Time	2		2		2		ns	Note 3
t18	D0–31 Valid Delay	3	38	3	32	3	24	ns	
t19	D0–31 Float Delay		25		20		17	ns	Note 5
t20	READYO# Valid Delay	4	32	4	25	4	17.5	ns	
t21	READYO# Float Delay		25		20		15	ns	Notes 4,5
t22	READYO# Setup Time	16		13		11		ns	
t23	READYO# Hold Time	4		4		4		ns	
t24a	CONF# Setup Time	12		10		5		ns	Note 8
t24b	CONF# Setup Time	16		13		11		ns	Note 9
t25a	CONF# Hold Time	4		3		2		ns	Note 8
t25b	CONF# Hold Time	4		4		4		ns	Note 9

Table 9.4 - System Bus Signal AC Parameters

Symbol	Parameter	20 MHz		25 MHz		33 MHz		Units	Notes
		Min	Max	Min	Max	Min	Max		
t31	SA2-31, SBE0-3#, SLOCK#, SD/C#, SW/R#, SM/IO# Valid Delay	3	28	3	21	3	15	ns	
t32	SA2-31, SBE0-3#, SLOCK#, SD/C#, SW/R#, SM/IO# Float Delay	3	30	3	30	3	20	ns	Note 5
t33	SBLAST#, SHLDA, SBREQ, SNENE# Valid Delay	3	28	3	22	3	20	ns	
t34	SBLAST#, SNENE# Float Delay	3	30	3	25	3	20	ns	Note 5
t35	SD0-31 Write Data Valid Delay	3	38	3	27	3	24	ns	Note 4
t36	SD0-31 Float Delay		27		22	3	17	ns	Notes 4,5
t37	SA4-31 Setup Time	10		9		7		ns	
t38	SA4-31 Hold Time	3		3		3		ns	
t39	SD0-31 Read Setup Time	11		7		5		ns	
t40	SD0-31 Read Hold Time	3		3		3		ns	
t41	SNA# Setup Time	18		13		7		ns	Note 3
t42	SNA# Hold Time	3		3		3		ns	Note 3
t43a	SKEN# Setup Time	17		12		6.5		ns	
t43b	SHOLD, SAHOLD, SFHOLD# Setup Time	18		15		12		ns	
t43c	SWP# Setup Time	17		12		10		ns	
t44	SHOLD, SKEN#, SWP#, SFHOLD#, SAHOLD Hold Time	3		3		3		ns	
t45a	SEADS# Setup Time	14		11		7		ns	
t45b	SRDY#, SBRDY# Setup Time	14		11		9		ns	
t46	SEADS#, SRDY#, SBRDY# Hold Time	4		4		4		ns	
t47	FLUSH#, A20M# Setup Time	18		13		8		ns	Note 6
t48	FLUSH#, A20M# Hold Time	3		3		3		ns	Note 6
t49	SADS# Valid Delay	3	28	3	22	3	16	ns	
t50	SADS# Float Delay		25		20		15	ns	Notes 4,5

NOTES:
1. Tf is Measured at 3.7V to 0.8V. Tf is not 100% tested.
2. Tr is Measured at 0.8V to 3.7V. Tr is not 100% tested.
3. The specification is relative to PHI2 i.e. signal sampled by PHI2.
4. The specification is relative to PHI2 i.e. signal driven by PHI2.
5. Float condition occurs when maximum output current becomes less than ILO in magnitude. Float delay is not 100% tested.
6. The signal is allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.
7. The signal is not sampled. It must be valid through the entire cycle (as the Address lines).
8. When tested as the second 82395DX.
9. When tested as the third 82395DX.

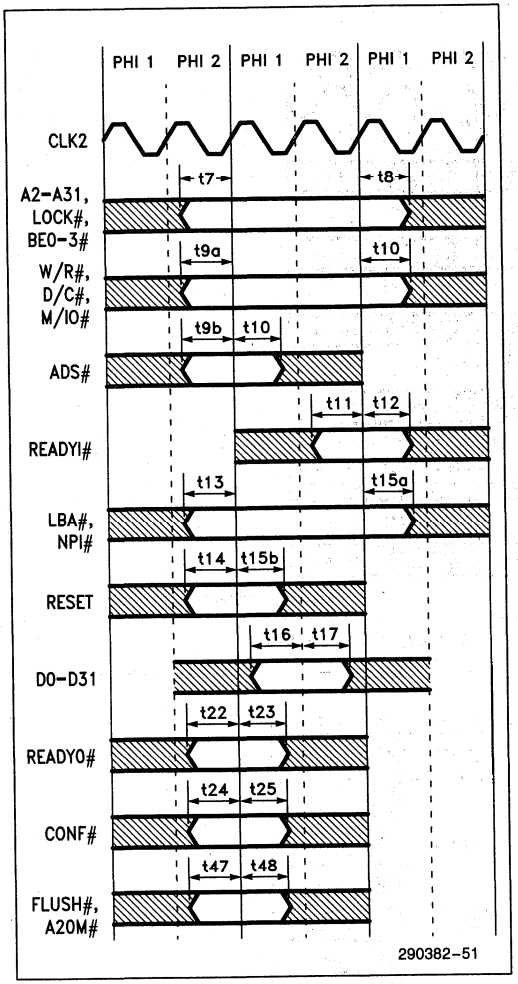


Figure 9.2 - AC Timing Waveforms - Local Bus Input Setup and Hold Timing

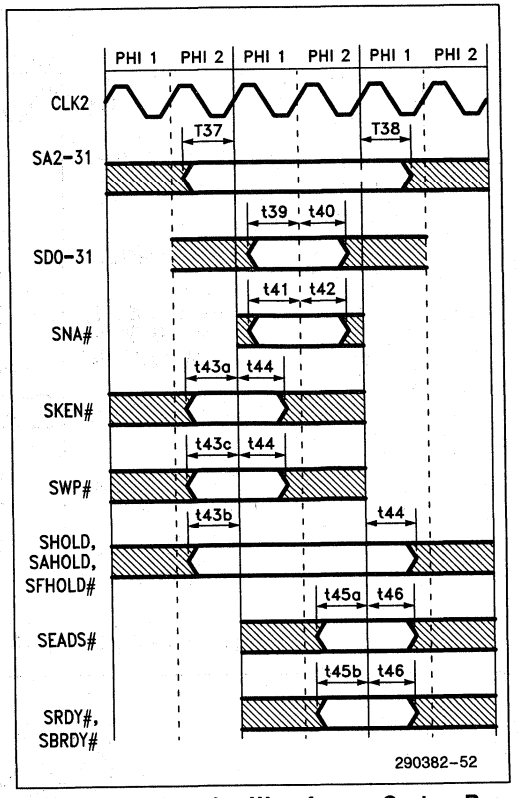


Figure 9.3 - AC Timing Waveforms - System Bus Input Setup and Hold Timing

5

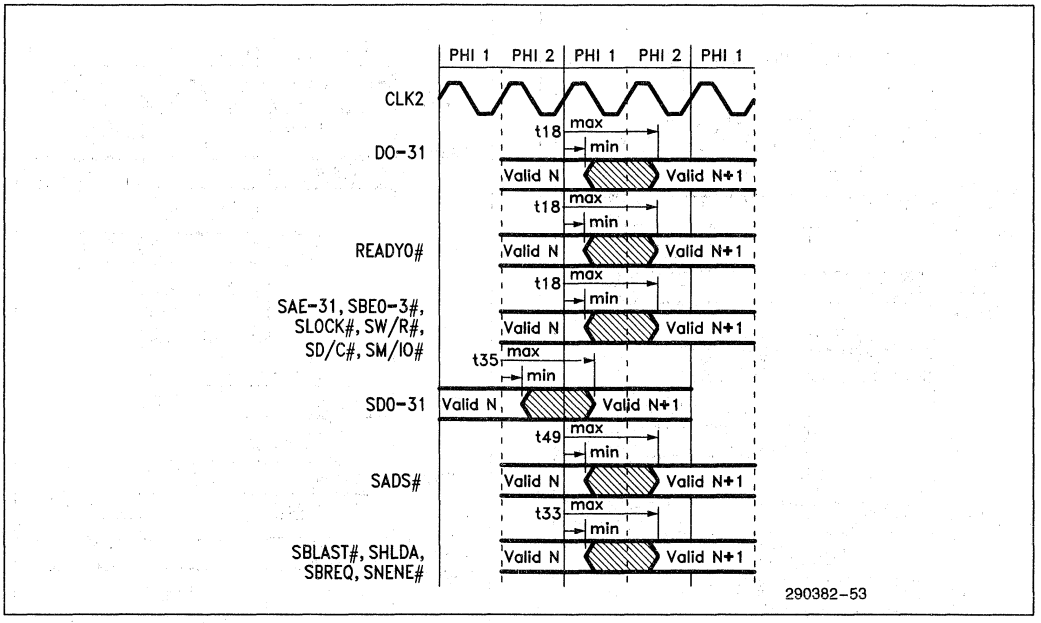


Figure 9.4 - AC Timing Waveforms - Output Valid Delay

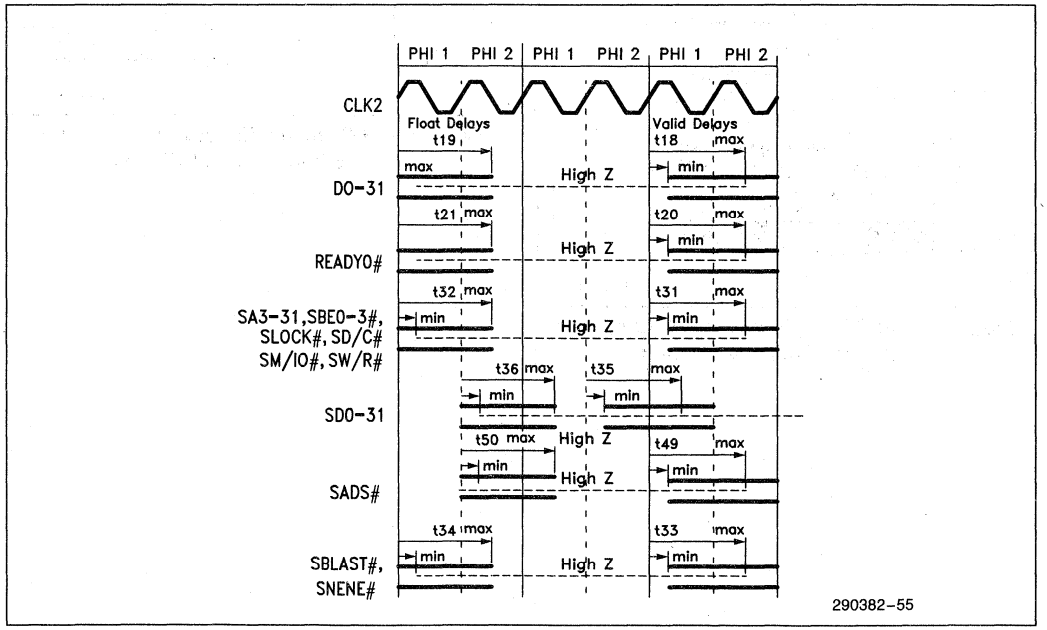


Figure 9.5 - AC Timing Waveforms - Output Float Delays

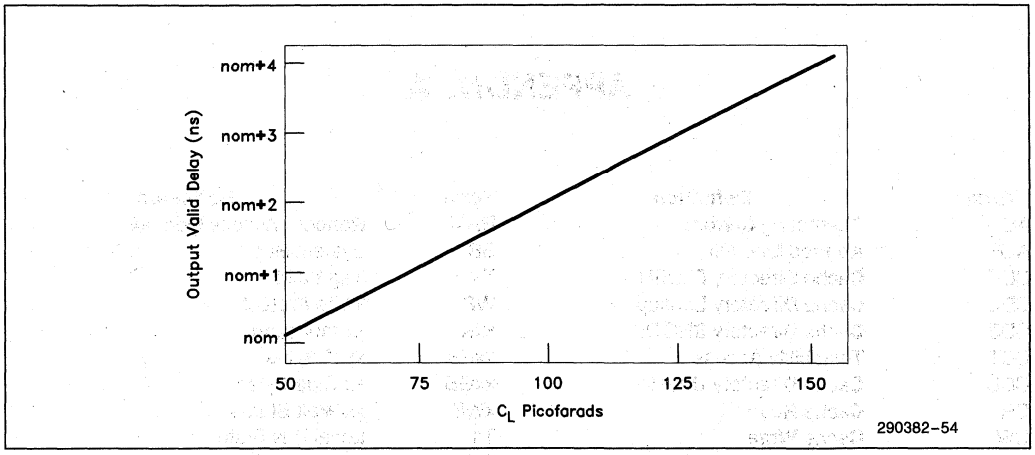


Figure 9.6 - Typical Output Valid Delay vs Load Capacitance at Maximum Operating Temperature (C_L = 50pF)

APPENDIX A

Term	Definition	Term	Definition
AC	**A**lternating **C**urrent	RAM	**R**andom **A**ccess **M**emory
ALF	**A**borted **L**ine **F**ill	SB	**S**ystem **B**us
CDF	**C**ache **D**irectory **F**LUSH	TV	**T**ag **V**alid
CDL	**C**ache **D**irectory **L**ookup	WP	**W**rite **P**rotect
CDS	**C**ache **D**irectory **S**NOOP	xxK	xx thousand
CDT	**T**estability Access	xxKB	xx **K B**ytes
CDU	**C**ache **D**irectory **U**pdate	xxGB	xx **G**iga **B**ytes
CR	**C**ache **R**ead	xWS	xx **W**ait **S**tates
CW	**C**ache **W**rite	T1	Local Bus State
CU	**C**ache **U**pdate	T2	Local Bus State
CT	**T**estability **A**ccess	TI	Local Bus State
CPU	**C**entral **P**rocessing **U**nit	TH	Local Bus State
CHMOS	**C**omplimentary **H**igh Performance **M**etal **O**xide **S**emiconductor	ST1	System Bus State
		ST1P	System Bus State
CRDH	**C**ache **R**ead **H**it	ST2	System Bus State
CRDM	**C**ache **R**ead **M**iss	ST2P	System Bus State
CWTH	**C**ache **W**rite **H**it	STI	System Bus State
DC	**D**irect **C**urrent	STH	System Bus State
DRAM	**D**ynamic **R**andom **A**ccess **M**emory	PHI1	1st CLK2 cycle in a 2 CLK2 CLK cycle
DMA	**D**irect **M**emory **A**ccess	PHI2	2nd CLK2 cycle in a 2 CLK2 CLK cycle
DW	**D**ouble **W**ord	C	Celsius
GND	Ground	V	Volts
I/O	**I**nput/**O**utput	μA	10^{-6} Amps
LB	**L**ocal **B**us	mA	10^{-3} Amps
LBA	**L**ocal **B**us **A**ccess	pF	10^{-12} Farads
LRU	**L**east **R**ecently **U**sed	MHz	10^{6} Hertz
PQFP	**P**lastic **Q**uad **F**lat **P**ack	ns	10^{-9} seconds

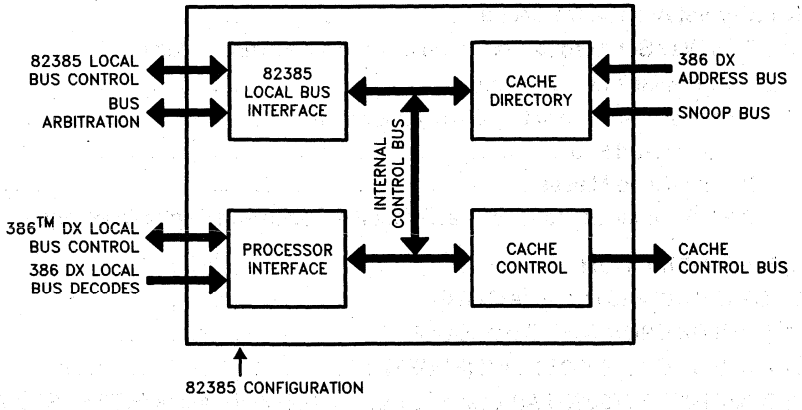
82385
HIGH PERFORMANCE
32-BIT CACHE CONTROLLER

- **Improves 386™ DX System Performance**
  - **— Reduces Average CPU Wait States to Nearly Zero**
  - **— Zero Wait State Read Hit**
  - **— Zero Wait State Posted Memory Writes**
  - **— Allows Other Masters to Access the System Bus More Readily**
- **Hit Rates up to 99%**
- **Optimized as 386 DX Companion**
  - **— Simple 386 DX Interface**
  - **— Part of 386 DX-Based Compute Engine Including 387™ DX Math Coprocessor and 82380 Integrated System Peripheral**
  - **— 20 MHz, 25 MHz, and 33 MHz Operation**
- **Software Transparent**
- **Synchronous Dual Bus Architecture**
  - **— Bus Watching Maintains Cache Coherency**
- **Maps Full 386 DX Address Space (4 Gigabytes)**
- **Flexible Cache Mapping Policies**
  - **— Direct Mapped or 2-Way Set Associative Cache Organization**
  - **— Supports Non-Cacheable Memory Space**
  - **— Unified Cache for Code and Data**
- **Integrates Cache Directory and Cache Management Logic**
- **High Speed CHMOS* IV Technology**
- **132-Pin PGA Package**
- **132-Lead Plastic Quad Flat Pack (PQFP)**

The 82385 Cache Controller is a high performance 32-bit peripheral for the Intel386 Microprocessor. It stores a copy of frequently accessed code and data from main memory in a zero wait state local cache memory. The 82385 enables the 386 DX to run at its full potential by reducing the average number of CPU wait states to nearly zero. The dual bus architecture of the 82385 allows other masters to access system resources while the 386 DX operates locally out of its cache. In this situation, the 82385's "bus watching" mechanism preserves cache coherency by monitoring the system bus address lines at no cost to system or local throughput.

5

The 82385 is completely software transparent, protecting the integrity of system software. High performance and board savings are achieved because the 82385 integrates a cache directory and all cache management logic on one chip.



290143-1

82385 Internal Block Diagram

*CHMOS is a patented process of Intel Corporation.
Intel386™, 386™ DX, 387™ DX are trademarks of Intel Corporation.

October 1990
Order Number: 290143-006

CONTENTS

PAGE

1.0 82385 FUNCTIONAL OVERVIEW 5-552
1.1 82385 Overview 5-552
1.2 System Overview I: Bus Structure 5-552
1.2.1 386™DX Local Bus/82385 Local Bus/System Bus 5-553
1.2.2 Bus Arbitration 5-553
1.2.3 Master/Slave Operation 5-553
1.2.4 Cache Coherency 5-554
1.3 System Overview II: Basic Operation 5-555
1.3.1 386 DX Memory Code and Data Read Cycles 5-555
1.3.1.1 Read Hits 5-555
1.3.1.2 Read Misses 5-555
1.3.2 386 DX Memory Write Cycles 5-555
1.3.3 Non-Cacheable Cycles 5-555
1.3.3.1 16-Bit Memory Space 5-556
1.3.4 386 DX Local Bus Cycles 5-556
1.3.5 Summary of 82385 Response to All 386 DX Cycles 5-556
1.3.6 Bus Watching 5-558
1.3.7 Cache Flush 5-558

2.0 82385 CACHE ORGANIZATION 5-558
2.1 Direct Mapped Cache 5-558
2.1.1 Direct Mapped Cache Structure and Terminology 5-558
2.1.2 Direct Mapped Cache Operation 5-559
2.1.2.1 Read Hits 5-559
2.1.2.2 Read Misses 5-559
2.1.2.3 Other Operations That Affect the Cache and Cache Directory 5-560
2.2 Two Way Set Associative Cache 5-560
2.2.1 Two Way Set Associative Cache Structure and Terminology 5-560
2.2.2 LRU Replacement Algorithm 5-561
2.2.3 Two Way Set Associative Cache Operation 5-561
2.2.3.1 Read Hits 5-561
2.2.3.2 Read Misses 5-561
2.2.3.3 Other Operations That Affect the Cache and Cache Directory 5-561

3.0 82385 PIN DESCRIPTION 5-562
3.1 386 DX CPU/82385 Interface Signals 5-562
3.1.1 386 DX CPU/82385 Clock (CLK2) 5-562
3.1.2 386 DX CPU/82385 Reset (RESET) 5-562
3.1.3 386 DX CPU/82385 Address Bus (A2–A31), Byte Enables (BE0#–BE3#) and Cycle Definition Signals (M/IO#, D/C#, W/R#, LOCK#) 5-562
3.1.4 386 DX CPU/82385 Address Status (ADS#) and Ready Input (READYI#) 5-563
3.1.5 386 DX Next Address Request (NA#) 5-563
3.1.6 Ready Output (READYO#) and Bus Ready Enable (BRDYEN#) 5-563

CONTENTS

PAGE

3.2 Cache Control Signals ... 5-563
3.2.1 Cache Address Latch Enable (CALEN) ... 5-563
3.2.2 Cache Transmit/Receive (CT/R#) ... 5-563
3.2.3 Cache Chip Selects (CS0#–CS3#) ... 5-563
3.2.4 Cache Output Enables (COEA#, COEB#) and Write Enables (CWEA#, CWEB#) ... 5-563
3.3 386 DX Local Bus Decode Inputs ... 5-564
3.3.1 386 DX Local Bus Access (LBA#) ... 5-564
3.3.2 Non-Cacheable Access (NCA#) ... 5-564
3.3.3 16-Bit Access (X16#) ... 5-564
3.4 82385 Bus Interface Signals ... 5-564
3.4.1 82385 Bus Byte Enables (BBE0#–BBE3#) ... 5-564
3.4.2 82385 Bus Lock (BLOCK#) ... 5-564
3.4.3 82385 Bus Address Status (BADS#) ... 5-565
3.4.4 82385 Bus Ready Input (BREADY#) ... 5-565
3.4.5 82385 Bus Next Address Request (BNA#) ... 5-565
3.5 82385 Bus Data Transceiver and Address Latch Control Signals ... 5-565
3.5.1 Local Data Strobe (LDSTB), Data Output Enable (DOE#), and Bus Transmit/Receive (BT/R#) ... 5-565
3.5.2 Bus Address Clock Pulse (BACP) and Bus Address Output Enable (BAOE#) ... 5-565
3.6 Status and Control Signals ... 5-565
3.6.1 Cache Miss Indication (MISS#) ... 5-565
3.6.2 Write Buffer Status (WBS) ... 5-565
3.6.3 Cache Flush (FLUSH) ... 5-566
3.7 Bus Arbitration Signals (BHOLD and BHLDA) ... 5-566
3.8 Coherency (Bus Watching) Support Signals (SA2–SA31, SSTB#, SEN) ... 5-566
3.9 Configuration Inputs (2W/D#, M/S#, DEFOE#) ... 5-566

4.0 386 DX LOCAL BUS INTERFACE ... 5-566
4.1 Processor Interface ... 5-567
4.1.1 Hardware Interface ... 5-567
4.1.2 Ready Generation ... 5-569
4.1.3 NA# and 386 DX Local Bus Pipelining ... 5-570
4.1.4 LBA#, NCA#, and X16# Generation ... 5-571
4.1.5 82385 Handling of 16-Bit Space ... 5-572
4.2 Cache Interface ... 5-572
4.2.1 Cache Configurations ... 5-572
4.2.2 Cache Control . . . Direct Mapped ... 5-575
4.2.3 Cache Control . . . Two Way Set Associative ... 5-577
4.3 387™ DX Interface ... 5-577

CONTENTS

PAGE

5.0 82385 LOCAL BUS AND SYSTEM INTERFACE 5-578
5.1 The 82385 Bus State Machine 5-578
5.1.1 Master Mode 5-578
5.1.2 Slave Mode 5-581
5.2 The 82385 Local Bus 5-582
5.2.1 82385 Bus Counterparts to 386 DX Signals 5-582
5.2.1.1 Address Bus (BA2–BA31) and Cycle Definition Signals (BM/IO#, BD/C#, BW/R#) 5-582
5.2.1.2 Data Bus (BD0–BD31) 5-582
5.2.1.3 Byte Enables (BBE0#–BBE3#) 5-583
5.2.1.4 Address Status (BADS#) 5-583
5.2.1.5 Ready (BREADY#) 5-583
5.2.1.6 Next Address (BNA#) 5-583
5.2.1.7 Bus Lock (BLOCK#) 5-583
5.2.2 Additional 82385 Bus Signals 5-584
5.2.2.1 Cache Read/Write Miss Indication (MISS#) 5-584
5.2.2.2 Write Buffer Status (WBS) 5-584
5.2.2.3 Cache Flush (FLUSH) 5-591
5.3 Bus Watching (Snoop) Interface 5-591
5.4 Reset Definition 5-591

6.0 SYSTEM DESIGN GUIDELINES 5-594
6.1 Introduction 5-594
6.2 Power and Grounding 5-594
6.2.1 Power Connections 5-594
6.2.2 Power Decoupling 5-594
6.2.3 Resistor Recommendations 5-595
6.2.3.1 386 DX Local Bus 5-595
6.2.3.2 82385 Local Bus 5-595
6.3 82385 Signal Connections 5-595
6.3.1 Configuration Inputs 5-595
6.3.2 CLK2 and RESET 5-596
6.4 Unused Pin Requirements 5-596
6.5 Cache SRAM Requirements 5-596
6.5.1 Cache Memory without Transceivers 5-596
6.5.2 Cache Memory with Transceivers 5-596

CONTENTS

PAGE

7.0 SYSTEM TEST CONSIDERATIONS 5-597
7.1 Introduction 5-597
7.2 Main Memory (DRAM) Testing 5-597
7.2.1 Memory Testing Routine 5-597
7.3 82385 Cache Memory Testing 5-597
7.3.1 Test Routine in the NCA# or LBA# Memory Map 5-597
7.3.2 Test Routine in Cacheable Memory 5-598
7.4 82385 Cache Directory Testing 5-598
7.5 Special Function Pins 5-598

8.0 MECHANICAL DATA 5-598
8.1 Introduction 5-598
8.2 Pin Assignment 5-598
8.3 Package Dimensions and Mounting 5-603
8.4 Package Thermal Specification 5-603

9.0 ELECTRICAL DATA 5-608
9.1 Introduction 5-608
9.2 Maximum Ratings 5-608
9.3 D.C. Specifications 5-608
9.4 A.C. Specifications 5-609
9.4.1 Frequency Dependent Signals 5-609

10.0 REVISION HISTORY 5-619

5

1.0 82385 FUNCTIONAL OVERVIEW

The 82385 Cache Controller is a high performance 32-bit peripheral for the Intel386 microprocessor. This chapter provides an overview of the 82385, and of the basic architecture and operation of an 386 DX CPU/82385 system.

1.1 82385 OVERVIEW

The main function of a cache memory system is to provide fast local storage for frequently accessed code and data. The cache system intercepts 386 DX memory references to see if the required data resides in the cache. If the data resides in the cache (a hit), it is returned to the 386 DX without incurring wait states. If the data is not cached (a miss), the reference is forwarded to the system and the data retrieved from main memory. An efficient cache will yield a high "hit rate" (the ratio of cache hits to total 386 DX accesses), such that the majority of accesses are serviced with zero wait states. The net effect is that the wait states incurred in a relatively infrequent miss are averaged over a large number of accesses, resulting in an average of nearly zero wait states per access. Since cache hits are serviced locally, a processor operating out of its local cache has a much lower "bus utilization" which reduces system bus bandwidth requirements, making more bandwidth available to other bus masters.

The 82385 Cache Controller integrates a cache directory and all cache management logic required to support an external 32 Kbyte cache. The cache directory structure is such that the entire physical address range of the 386 DX (4 Gigabytes) is mapped into the cache. Provision is made to allow areas of memory to be set aside as non-cacheable. The user has two cache organization options: direct mapped and 2-way set associative. Both provide the high hit rates necessary to make a large, relatively slow main memory array look like a fast, zero wait state memory to the 386 DX.

1.2 SYSTEM OVERVIEW I: BUS STRUCTURE

A good grasp of the bus structure of a 386 DX CPU/82385 system is essential in understanding both the 82385 and its role in an 386 DX system. The following is a description of this structure.

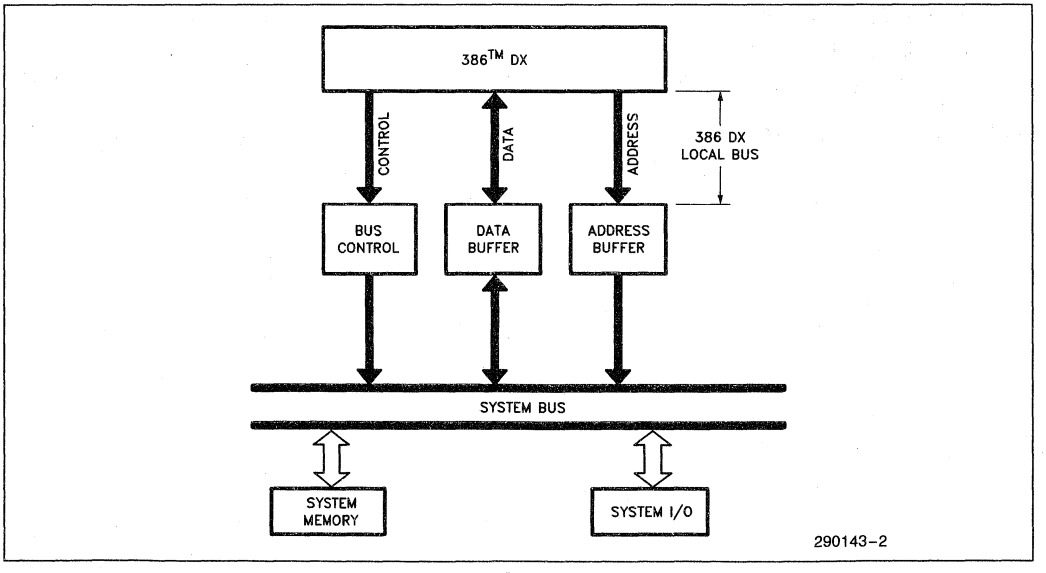


Figure 1-1. 386 DX System Bus Structure

1.2.1 386 DX Local Bus/82385 Local Bus/System Bus

Figure 1-1 depicts the bus structure of a typical 386 DX system. The "386 DX Local Bus" consists of the physical 386 DX address, data, and control busses. The local address and data busses are buffered and/or latched to become the "system" address and data busses. The local control bus is decoded by bus control logic to generate the various system bus read and write commands.

The addition of an 82385 Cache Controller causes a separation of the 386 DX bus into two distinct busses: the actual 386 DX local bus and the "82385 Local Bus" (Figure 1-2). The 82385 local bus is designed to look like the front end of an 386 DX by providing 82385 local bus equivalents to all appropriate 386 DX signals. The system ties to this "386 DX-like" front end just as it would to an actual 386 DX. The 386 DX simply sees a fast system bus, and the system sees a 386 DX front end with low bus bandwidth requirements. The cache subsystem is transparent to both. Note that the 82385 local bus is not simply a buffered version of the 386 DX bus, but rather is distinct from, and able to operate in parallel with the 386 DX bus. Other masters residing on either the 82385 local bus or system bus are free to manage system resources while the 386 DX operates out of its cache.

1.2.2 Bus Arbitration

The 82385 presents the "386 DX-like" interface which is called the 82385 local bus. Whereas the 386 DX provides a Hold Request/Hold Acknowledge bus arbitration mechanism via its HOLD and HLDA pins, the 82385 provides an equivalent mechanism via its BHOLD and BHLDA pins. (These signals are described in Section 3.7.) When another master requests the 82385 local bus, it issues the request to the 82385 via BHOLD. Typically, at the end of the current 82385 local bus cycle, the 82385 will release the 82385 local bus and acknowledge the request via BHLDA. The 386 DX is of course free to continue operating on the 386 DX local bus while another master owns the 82385 local bus.

1.2.3 Master/Slave Operation

The above 82385 local bus arbitration discussion is true when the 82385 is programmed for "Master" mode operation. The user can, however, configure the 82385 for "Slave" mode operation. (Programming is done via a hardware strap option.) The roles of BHOLD and BHLDA are reversed for an 82385 in slave mode; BHOLD is now an output indicating a request to control the bus, and BHLDA is an input indicating that a request has been granted. An 82385 programmed in slave mode drives the 82385 local bus only when it has requested and subsequently been granted bus control. This allows multiple 386 DX CPU/82385 subsystems to reside on the same 82385 local bus (Figure 1-3).

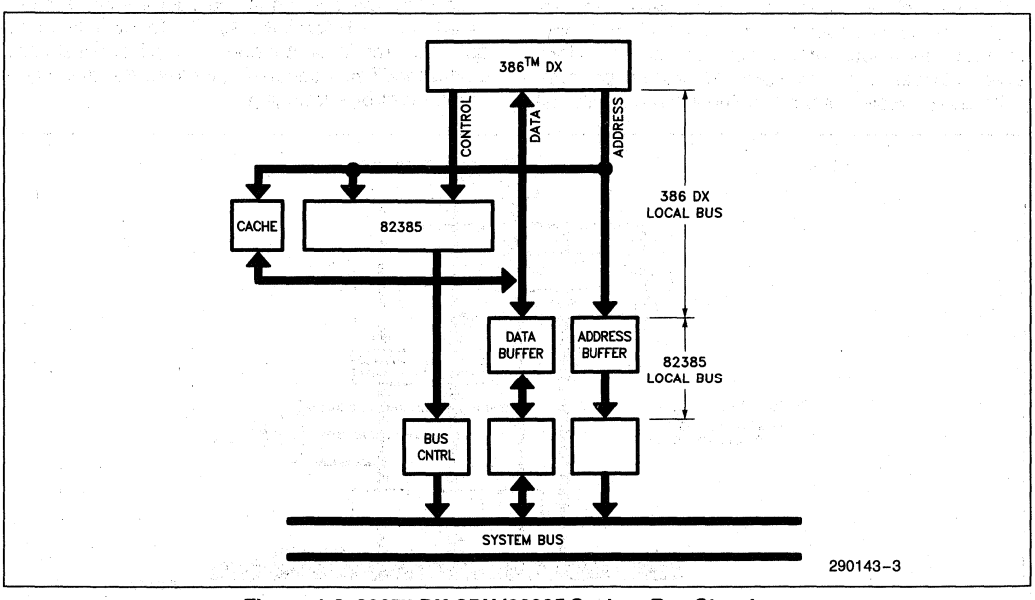


Figure 1-2. 386™ DX CPU/82385 System Bus Structure

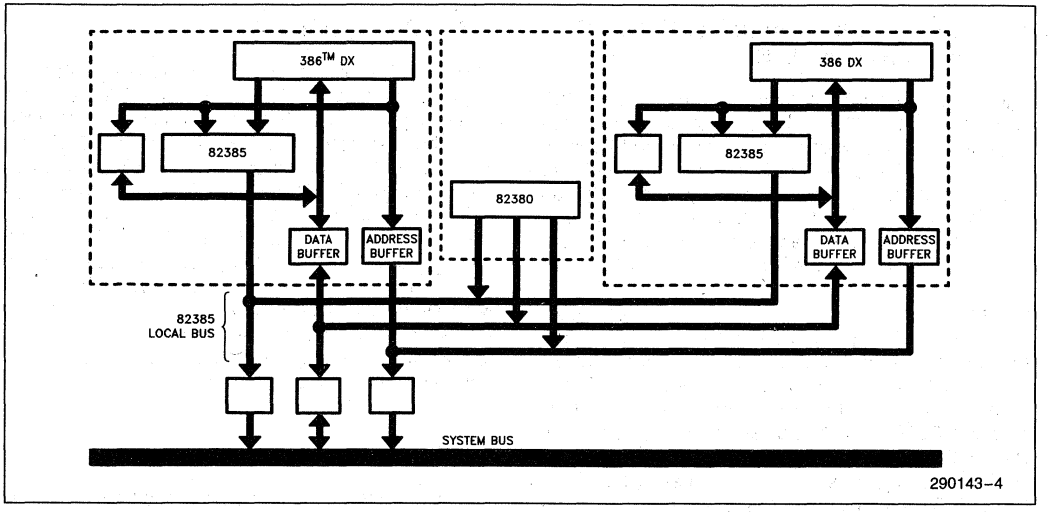


Figure 1-3. Multi-Master/Multi-Cache Environment

1.2.4 Cache Coherency

Ideally, a cache contains a copy of the most heavily used portions of main memory. To maintain cache "coherency" is to make sure that this local copy is identical to main memory. In a system where multiple masters can access the same memory, there is always a risk that one master will alter the contents of a memory location that is duplicated in the local cache of another master. (The cache is said to contain "stale" data.) One rather restrictive solution is to not allow cache subsystems to cache shared memory. Another simple solution is to flush the cache anytime another master writes to system memory. However, this can seriously degrade system performance as excessive cache flushing will reduce the hit rate of what may otherwise be a highly efficient cache.

The 82385 preserves cache coherency via "bus watching" (also called snooping), a technique that neither impacts performance nor restricts memory mapping. An 82385 that is not currently bus master monitors system bus cycles, and when a write cycle by another master is detected (a snoop), the system address is sampled and used to see if the referenced location is duplicated in the cache. If so (a snoop hit), the corresponding cache entry is invalidated, which will force the 386 DX to fetch the up-to-date data from main memory the next time it accesses this modified location. Figure 1-4 depicts the general form of bus watching.

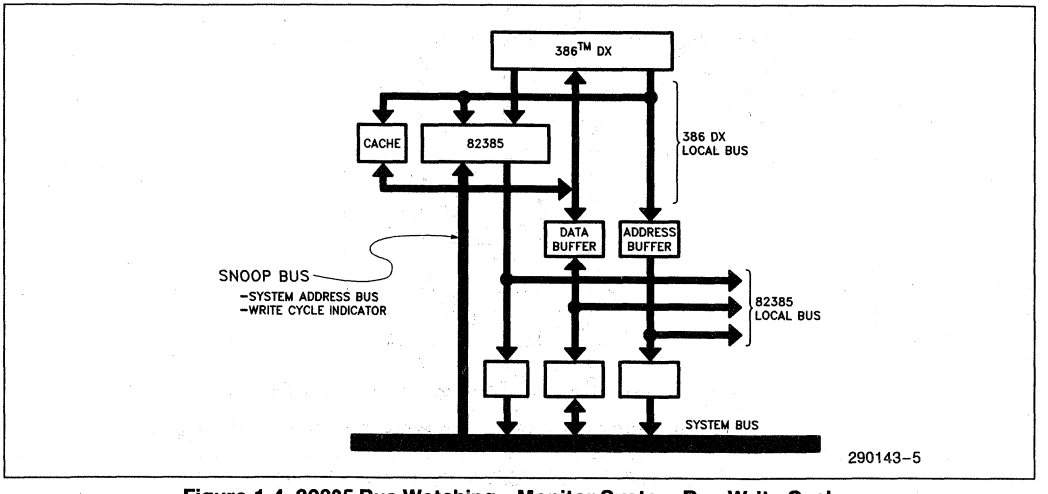


Figure 1-4. 82385 Bus Watching—Monitor System Bus Write Cycles

1.3 SYSTEM OVERVIEW II: BASIC OPERATION

This discussion is an overview of the basic operation of an 386 DX CPU/82385 system. Items discussed include the 82385's response to all 386 DX cycles, including interrupt acknowledges, halts, and shutdowns. Also discussed are non-cacheable and local accesses.

1.3.1 386 DX Memory Code and Data Read Cycles

1.3.1.1 READ HITS

When the 386 DX initiates a memory code or data read cycle, the 82385 compares the high order bits of the 386 DX address bus with the appropriate addresses (tags) stored in its on-chip directory. (The directory structure is described in Chapter 2.) If the 82385 determines that the requested data is in the cache, it issues the appropriate control signals that direct the cache to drive the requested data onto the 386 DX data bus, where it is read by the 386 DX. The 82385 terminates the 386 DX cycle without inserting any wait states.

1.3.1.2 READ MISSES

If the 82385 determines that the requested data is not in the cache, the request is forwarded to the 82385 local bus and the data retrieved from main memory. As the data returns from main memory, it is directed to the 386 DX and also written into the cache. Concurrently, the 82385 updates the cache directory such that the next time this particular piece of information is requested by the 386 DX, the 82385 will find it in the cache and return it with zero wait states.

The basic unit of transfer between main memory and cache memory in a cache subsystem is called the line size. In an 82385 system, the line size is one 32-bit aligned doubleword. During a read miss, all four 82385 local bus byte enables are active. This ensures that a full 32-bit entry is written into the cache. (The 386 DX simply ignores what it did not request.) In any other type of 386 DX cycle that is forwarded to the 82385 local bus, the logic levels of the 386 DX byte enables are duplicated on the 82385 local bus.

The 82385 does not actively fetch main memory data independently of the 386 DX. The 82385 is essentially a passive device which only monitors the address bus and activates control signals. The read miss is the only mechanism by which main memory data is copied into the cache and validated in the cache directory.

In an isolated read miss, the number of wait states seen by the 386 DX is that required by the system memory to respond with data plus the cache comparison cycle (hit/miss decision). The cache system must determine that the cycle is a miss before it can begin the system memory access. However, since misses most often occur consecutively, the 82385 will begin 386 DX address pipelined cycles to effectively "hide" the comparison cycle beyond the first miss (refer to Section 4.1.3).

The 82385 can execute a main memory access on the 82385 local bus only if it currently owns the bus. If not, an 82385 in master mode will run the cycle after the current master releases the bus. An 82385 in slave mode will issue a hold request, and will run the cycle as soon as the request is acknowledged. (This is true for any read or write cycle that needs to run on the 82385 local bus.)

1.3.2 386 DX Memory Write Cycles

The 82385's "posted write" capability allows the majority of 386 DX memory write cycles to run with zero wait states. The primary memory update policy implemented in a posted write is the traditional cache "write through" technique, which implies that main memory is always updated in any memory write cycle. If the referenced location also happens to reside in the cache (a write hit), the cache is updated as well.

Beyond this, a posted write latches the 386 DX address, data, and cycle definition signals, and the 386 DX local bus cycle is terminated without any wait states, even though the corresponding 82385 local bus cycle is not yet completed, or perhaps not even started. A posted write is possible because the 82385's bus state machine, which is almost identical to the 386 DX bus state machine, is able to run 82385 local bus cycles independently of the 386 DX. The only time the 386 DX sees write cycle wait states is when a previously latched (posted) write has not yet been completed on the 82385 local bus or during an I/O write (which is not posted). A 386 DX write can be posted even if the 82385 does not currently own the 82385 local bus. In this case, an 82385 in master mode will run the cycle as soon as the current master releases the bus, and an 82385 in slave mode will request the bus and run the cycle when the request is acknowledged. The 386 DX is free to continue operating out of its cache (on the 386 DX local bus) during this time.

1.3.3 Non-Cacheable Cycles

Non-cacheable cycles fall into one of two categories: cycles decoded as non-cacheable, and cycles

that are by default non-cacheable according to the 82385's design. All non-cacheable cycles are forwarded to the 82385 local bus. Non-cacheable cycles have no effect on the cache or cache directory.

The 82385 allows the system designer to define areas of main memory as non-cacheable. The 386 DX address bus is decoded and the decode output is connected to the 82385's non-cacheable access (NCA#) input. This decoding is done in the first 386 DX bus state in which the non-cacheable cycle address becomes available. Non-cacheable read cycles resemble cacheable read miss cycles, except that the cache and cache directory are unaffected. NCA defined non-cacheable writes, like most writes, are posted.

The 82385 defines certain cycles as non-cacheable without using its non-cacheable access input. These include I/O cycles, interrupt acknowledge cycles, and halt/shutdown cycles. I/O reads and interrupt acknowledge cycles execute as any other non-cacheable read. I/O write cycles are not posted. The 386 DX is not allowed to continue until a ready signal is returned from the system. Halt/Shutdown cycles are posted. During a halt/shutdown condition, the 82385 local bus duplicates the behavior of the 386 DX, including the ability to recognize and respond to a BHOLD request. (The 82385's bus watching mechanism is functional in this condition.)

1.3.3.1 16-BIT MEMORY SPACE

The 82385 does not cache 16-bit memory space (as decoded by the 386 DX BS16# input), but does make provisions to handle 16-bit space as non-cacheable. (There is no 82385 equivalent to the 386 DX BS16# input.) In a system without an 82385, the 386 DX BS16# input need not be asserted until the last state of a 16-bit cycle for the 386 DX to recognize it as such (unless NA# is sampled active earlier in the cycle.) The 82385, however, needs this information earlier, specifically at the end of the first 386 DX bus state in which the address of the 16-bit cycle becomes available. The result is that in a system without an 82385, 16-bit devices can inform the 386 DX that they are 16-bit devices "on the fly," while in a system with an 82385, devices decoded as 16-bit (using the 386 DX BS16#) must be located in address space set aside for 16-bit devices. If 16-bit space is decoded according to 82385 guidelines (as described later in the data sheet), then the 82385 will handle 16-bit cycles just like the 386 DX does, including effectively locking the two halves of a non-aligned 16-bit transfer from interruption by another master.

1.3.4 386 DX Local Bus Cycles

386 DX Local Bus Cycles are accesses to resources on the 386 DX local bus other than to the 82385 itself. The 82385 simply ignores these accesses: they are neither forwarded to the system nor do they affect the cache. The designer sets aside memory and/or I/O space for local resources by decoding the 386 DX address bus and feeding the decode to the 82385's local bus access (LBA#) input. The designer can also decode the 386 DX cycle definition signals to keep specific 386 DX cycles from being forwarded to the system. For example, a multi-processor design may wish to capture and remedy a 386 DX shutdown locally without having it detected by the rest of the system. Note that in such a design, the local shutdown cycle must be terminated by local bus control logic. The 387 Math Coprocessor is considered a 386 DX local bus resource, but it need not be decoded as such by the user since the 82385 is able to internally recognize 387 accesses via the M/IO# and A31 pins.

1.3.5 Summary of 82385 Response to All 386 DX Cycles

Table 1-1 summarizes the 82385 response to all 386 DX bus cycles, as conditioned by whether or not the cycle is decoded as local or non-cacheable. The table describes the impact of each cycle on the cache and on the cache directory, and whether or not the cycle is forwarded to the 82385 local bus. Whenever the 82385 local bus is marked "IDLE", it implies that this bus is available to other masters.

Table 1-1. 82385 Response to 386 DX Cycles

386 DX Bus Cycle Definition					82385 Response when Decoded as Cacheable			82385 Response when Decoded as Non-Cacheable			82385 Response when Decoded as an 386 DX Local Bus Access		
M/IO#	D/C#	W/R#	386 DX Cycle		Cache	Cache Directory	82385 Local Bus	Cache	Cache Directory	82385 Local Bus	Cache	Cache Directory	82385 Local Bus
0	0	0	INT ACK	N/A	—	—	INT ACK	—	—	INT ACK	—	—	IDLE
0	0	1	UNDEFINED	N/A			UNDEFINED			UNDEFINED			IDLE
0	1	0	I/O READ	N/A	—	—	I/O READ	—	—	I/O READ	—	—	IDLE
0	1	1	I/O WRITE	N/A	—	—	I/O WRITE	—	—	I/O WRITE	—	—	IDLE
1	0	0	MEM CODE READ	HIT	CACHE READ	—	IDLE	—	—	MEM CODE READ	—	—	IDLE
				MISS	CACHE WRITE	DATA VALIDATION	MEM CODE READ						
1	0	1	HALT/ SHUTDOWN	N/A	—	—	HALT/ SHUTDOWN	—	—	HALT/ SHUTDOWN	—	—	IDLE
1	1	0	MEM DATA READ	HIT	CACHE READ	—	IDLE	—	—	MEM DATA READ	—	—	IDLE
				MISS	CACHE WRITE	DATA VALIDATION	MEM DATA READ						
1	1	1	MEM DATA WRITE	HIT	CACHE WRITE	—	MEM DATA WRITE	—	—	MEM DATA WRITE	—	—	IDLE
				MISS	—	—	MEM DATA WRITE						

NOTES:
- A dash (—) indicates that the cache and cache directory are unaffected. This table does not reflect how an access affects the LRU bit.
- An "IDLE" 82385 Local Bus implies that this bus is available to other masters.
- The 82385's response to 80387 accesses is the same as when decoded as an 386 DX Local Bus access.
- The only other operations that affect the cache directory are:
  1. RESET or Cache Flush—all tag valid bits cleared.
  2. Snoop Hit—corresponding line valid bit cleared.

1.3.6 Bus Watching

As previously discussed, the 82385 "qualifies" an 386 DX bus cycle in the first bus state in which the address and cycle definition signals of the cycle become available. The cycle is qualified as read or write, cacheable or non-cacheable, etc. Cacheable cycles are further classified as hit or miss according to the results of the cache comparison, which accesses the 82385 directory and compares the appropriate directory location (tag) to the current 386 DX address. If the cycle turns out to be non-cacheable or a 386 DX local bus access, the hit/miss decision is ignored. The cycle qualification requires one 386 DX state. Since the fastest 386 DX access is two states, the second state can be used for bus watching.

When the 82385 does not own the system bus, it monitors system bus cycles. If another master writes into main memory, the 82385 latches the system address and executes a cache look-up to see if the altered main memory location resides in the cache. If so (a snoop hit), the cache entry is marked invalid in the cache directory. Since the directory is at most only being used every other state to qualify 386 DX accesses, snoop look-ups are interleaved between 386 DX local bus look-ups. The cache directory is time multiplexed between the 386 DX address and the latched system address. The result is that all snoops are caught and serviced without slowing down the 386 DX, even when running zero wait state hits on the 386 DX local bus.

1.3.7 Cache Flush

The 82385 offers a cache flush input. When activated, this signal causes the 82385 to invalidate all data which had previously been cached. Specifically, all tag valid bits are cleared. (Refer to the 82385 directory structure in Chapter 2.) Therefore, the cache is empty and subsequent cycles are misses until the 386 DX begins repeating the new accesses (hits). The primary use of the FLUSH input is for diagnostics and multi-processor support.

NOTE:
The use of this pin as a coherency mechanism may impact software transparency.

2.0 82385 CACHE ORGANIZATION

The 82385 supports two cache organizations: a simple direct mapped organization and a slightly more complex, higher performance two way set associative organization. The choice is made by strapping an 82385 input (2W/D#) either high or low. This chapter describes the structure and operation of both organizations.

2.1 DIRECT MAPPED CACHE

2.1.1 Direct Mapped Cache Structure and Terminology

Figure 2-1 depicts the relationship between the 82385's internal cache directory, the external cache memory, and the 386 DX's 4 Gigabyte physical address space. The 4 Gigabytes can conceptually be thought of as cache "pages" each being 8K doublewords (32 Kbytes) deep. The page size matches the cache size. The cache can be further divided into 1024 (0 thru 1023) sets of eight doublewords (8 x 32 bits). Each 32-bit doubleword is called a "line." The unit of transfer between the main memory and cache is one line.

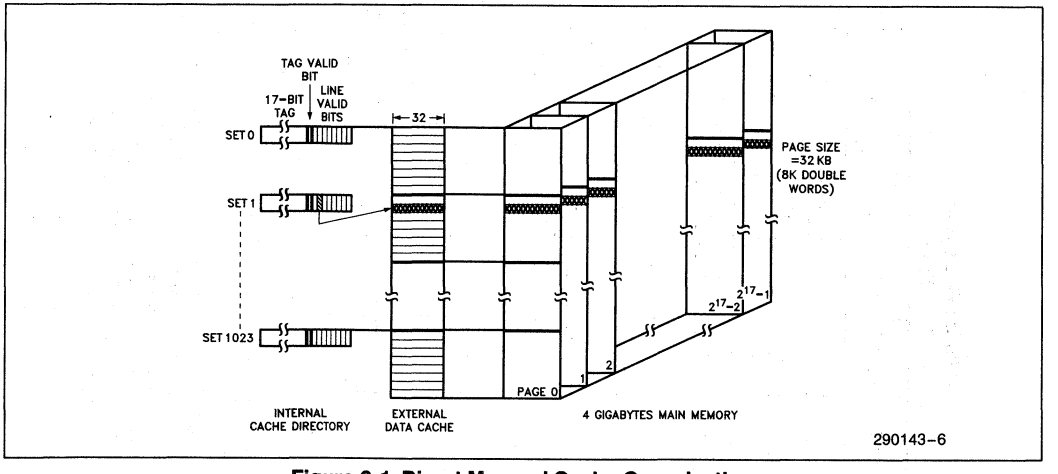


Figure 2-1. Direct Mapped Cache Organization

Each block in the external cache has an associated 26-bit entry in the 82385's internal cache directory. This entry has three components: a 17-bit "tag," a "tag valid" bit, and eight "line valid" bits. The tag acts as a main memory page number (17 tag bits support 2^{17} pages). For example, if line 9 of page 2 currently resides in the cache, then a binary 2 is stored in the Set 1 tag field. (For any 82385 direct mapped cache page in main memory, Set 0 consists of lines 0–7, Set 1 consists of lines 8–15, etc. Line 9 is shaded in Figure 2-1.) An important characteristic of a direct mapped cache is that line 9 of any page can only reside in line 9 of the cache. All identical page offsets map to a single cache location.

The data in a cache set is considered valid or invalid depending on the status of its tag valid bit. If clear, the entire set is considered invalid. If true, an individual line within the set is considered valid or invalid depending on the status of its line valid bit.

The 82385 sees the 386 DX address bus (A2–A31) as partitioned into three fields: a 17-bit "tag" field (A15–A31), a 10-bit "set-address" field (A5–A14), and a 3-bit "line select" field (A2–A4). (See Figure 2-2.) The lower 13 address bits (A2–A14) also serve as the "cache address" which directly selects one of 8K doublewords in the external cache.

2.1.2 Direct Mapped Cache Operation

The following is a description of the interaction between the 386 DX, cache, and cache directory.

2.1.2.1 READ HITS

When the 386 DX initiates a memory read cycle, the 82385 uses the 10-bit set address to select one of 1024 directory entries, and the 3-bit line select field to select one of eight line valid bits within the entry. The 13-bit cache address selects the corresponding doubleword in the cache. The 82385 compares the 17-bit tag field (A15–A31 of the 386 DX access) with the tag stored in the selected directory entry. If the tag and upper address bits match, and if both the tag and appropriate line valid bits are set, the result is a hit, and the 82385 directs the cache to drive the selected doubleword onto the 386 DX data bus. A read hit does not alter the contents of the cache or directory.

2.1.2.2 READ MISSES

A read miss can occur in two ways. The first is known as a "line" miss, and occurs when the tag and upper address bits match and the tag valid bit is set, but the line valid bit is clear. The second is called a "tag" miss, and occurs when either the tag and upper address bits do not match, or the tag valid bit is clear. (The line valid bit is a "don't care" in a tag miss.) In both cases, the 82385 forwards the 386 DX reference to the system, and as the returning data is fed to the 386 DX, it is written into the cache and validated in the cache directory.

In a line miss, the incoming data is validated simply by setting the previously clear line valid bit. In a tag miss, the upper address bits overwrite the previously stored tag, the tag valid bit is set, the appropriate line valid bit is set, and the other seven line valid bits are cleared. Subsequent tag hits with line misses will only set the appropriate line valid bit. (Any data associated with the previous tag is no longer considered resident in the cache.)

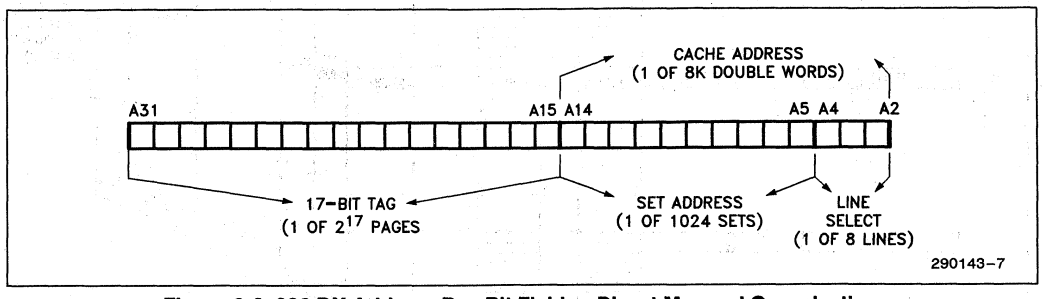


Figure 2-2. 386 DX Address Bus Bit Fields—Direct Mapped Organization

2.1.2.3 OTHER OPERATIONS THAT AFFECT THE CACHE AND CACHE DIRECTORY

The other operations that affect the cache and/or directory are write hits, snoop hits, cache flushes, and 82385 resets. In a write hit, the cache is updated along with main memory, but the directory is unaffected. In a snoop hit, the cache is unaffected, but the affected line is invalidated by clearing its line valid bit in the directory. Both an 82385 reset and cache flush clear all tag valid bits.

When an 386 DX CPU/82385 system "wakes up" upon reset, all tag valid bits are clear. At this point, a read miss is the only mechanism by which main memory data is copied into the cache and validated in the cache directory. Assume an early 386 DX code access seeks (for the first time) line 9 of page 2. Since the tag valid bit is clear, the access is a tag miss, and the data is fetched from main memory. Upon return, the data is fed to the 386 DX and simultaneously written into line 9 of the cache. The set directory entry is updated to show this line as valid. Specifically, the tag and appropriate line valid bits are set, the remaining seven line valid bits cleared, and a binary 2 written into the tag. Since code is sequential in nature, the 386 DX will likely next want line 10 of page 2, then line 11, and so on. If the 386 DX sequentially fetches the next six lines, these fetches will be line misses, and as each is fetched from main memory and written into the cache, its corresponding line valid bit is set. This is the basic flow of events that fills the cache with valid data. Only after a piece of data has been copied into the cache and validated can it be accessed in a zero wait state read hit. Also, a cache entry must have been validated before it can be subsequently altered by a write hit, or invalidated by a snoop hit.

An extreme example of "thrashing" is if line 9 of page two is an instruction to jump to line 9 of page one, which is an instruction to jump back to line 9 of page two. Thrashing results from the direct mapped cache characteristic that all identical page offsets map to a single cache location. In this example, the page one access overwrites the cached page two data, and the page two access overwrites the cached page one data. As long as the code jumps back and forth the hit rate is zero. This is of course an extreme case. The effect of thrashing is that a direct mapped cache exhibits a slightly reduced overall hit rate as compared to a set associative cache of the same size.

2.2 TWO WAY SET ASSOCIATIVE CACHE

2.2.1 Two Way Set Associative Cache Structure and Terminology

Figure 2-3 illustrates the relationship between the directory, cache, and 4 Gigabyte address space.

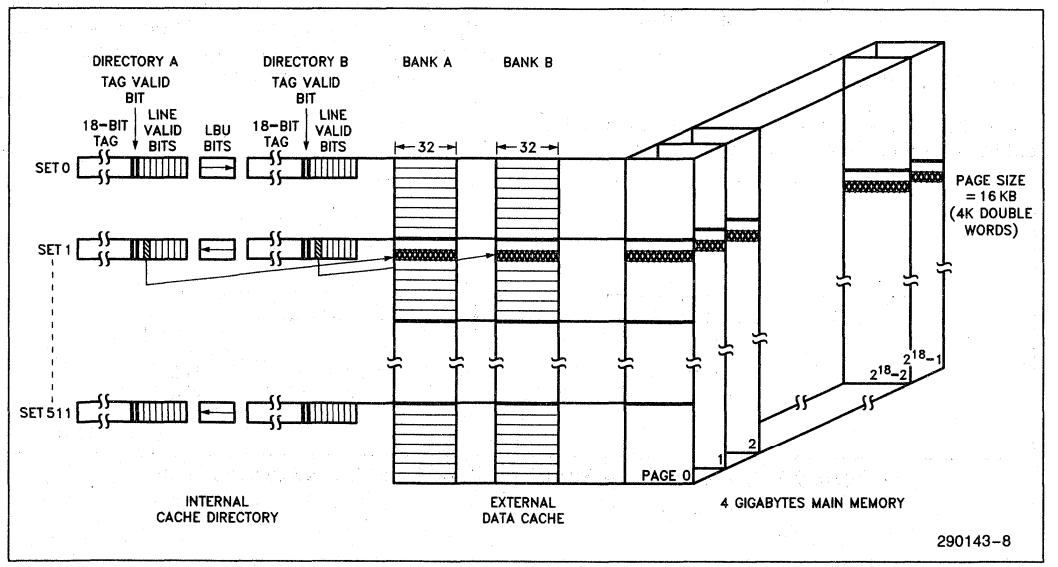


Figure 2-3. Two-Way Set Associative Cache Organization

Whereas the direct mapped cache is organized as one bank of 8K doublewords, the two way set associative cache is organized as two banks (A and B) of 4K doublewords each. The page size is halved, and the number of pages doubled. (Note the extra tag bit.) The cache now has 512 sets in each bank. (Two banks times 512 sets gives a total of 1024. The structure can be thought of as two half-sized direct mapped caches in parallel.) The performance advantage over a direct mapped cache is that all identical page offsets map to two cache locations instead of one, reducing the potential for thrashing. The 82385's partitioning of the 386 DX address bus is depicted in Figure 2-4.

2.2.2 LRU Replacement Algorithm

The two way set associative directory has an additional feature: the "least recently used" or LRU bit. In the event of a read miss, either bank A or bank B will be updated with new data. The LRU bit flags the candidate for replacement. Statistically, of two blocks of data, the block most recently used is the block most likely to be needed again in the near future. By flagging the least recently used block, the 82385 ensures that the cache block replaced is the least likely to have data needed by the CPU.

2.2.3 Two Way Set Associative Cache Operation

2.2.3.1 READ HITS

When the 386 DX initiates a memory read cycle, the 82385 uses the 9-bit set address to select one of 512 sets. The two tags of this set are simultaneously compared with A14–A31, both tag valid bits checked, and both appropriate line valid bits checked. If either comparison produces a hit, the corresponding cache bank is directed to drive the selected doubleword onto the 386 DX data bus. (Note that both banks will never concurrently cache the same main memory location.) If the requested data resides in bank A, the LRU bit is pointed toward B. If B produces the hit, the LRU bit is pointed toward A.

2.2.3.2 READ MISSES

As in direct mapped operation, a read miss can be either a line or tag miss. Let's start with a tag miss example. Assume the 386 DX seeks line 9 of page 2, and that neither the A or B directory produces a tag match. Assume also, as indicated in Figure 2-3, that the LRU bit points to A. As the data returns from main memory, it is loaded into offset 9 of bank A. Concurrently, this data is validated by updating the set 1 directory entry for bank A. Specifically, the upper address bits overwrite the previous tag, the tag valid bit is set, the appropriate line valid bit is set, and the other seven line valid bits cleared. Since this data is the most recently used, the LRU bit is turned toward B. No change to bank B occurs.

If the next 386 DX request is line 10 of page two, the result will be a line miss. As the data returns from main memory, it will be written into offset 10 of bank A (tag hit/line miss in bank A), and the appropriate line valid bit will be set. A line miss in one bank will cause the LRU bit to point to the other bank. In this example, however, the LRU bit has already been turned toward B.

2.2.3.3 OTHER OPERATIONS THAT AFFECT THE CACHE AND CACHE DIRECTORY

Other operations that affect the cache and cache directory are write hits, snoop hits, cache flushes, and 82385 resets. A write hit updates the cache along with main memory. If directory A detects the hit, bank A is updated. If directory B detects the hit, bank B is updated. If one bank is updated, the LRU bit is pointed toward the other.

If a snoop hit invalidates an entry, for example, in cache bank A, the corresponding LRU bit is pointed toward A. This ensures that invalid data is the prime candidate for replacement in a read miss. Finally, resets and flushes behave just as they do in a direct mapped cache, clearing all tag valid bits.

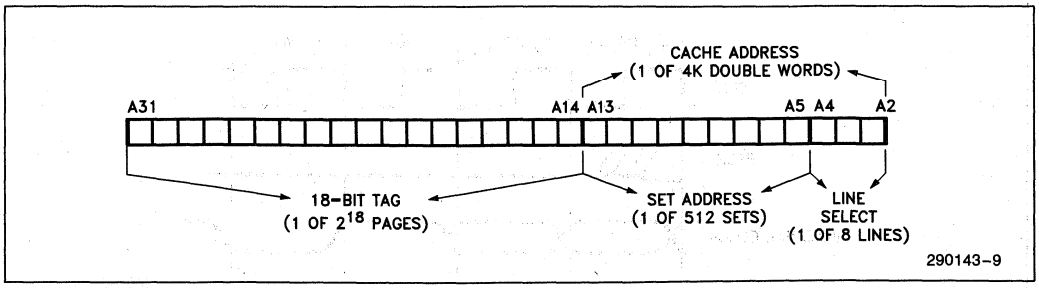


Figure 2-4. 386 DX Address Bus Bit Fields—Two-Way Set Associative Organization

3.0 82385 PIN DESCRIPTION

The 82385 creates the 82385 local bus, which is a functional 386 DX interface. To facilitate understanding, 82385 local bus signals go by the same name as their 386 DX equivalents, except that they are preceded by the letter "B". The 82385 local bus equivalent to ADS# is BADS#, the equivalent to NA# is BNA#, etc. This convention applies to bus states as well. For example, BT1P is the 82385 local bus state equivalent to the 386 DX T1P state.

3.1 386 DX CPU/82385 INTERFACE SIGNALS

These signals form the direct interface between the 386 DX and 82385.

3.1.1 386 DX CPU/82385 Clock (CLK2)

CLK2 provides the fundamental timing for an 386 DX CPU/82385 system, and is driven by the same source that drives the 386 DX CLK2 input. The 82385, like the 386 DX, divides CLK2 by two to generate an internal "phase indication" clock. (See Figure 3-1.) The CLK2 period whose rising edge drives the internal clock low is called PHI1, and the CLK2 period that drives the internal clock high is called PHI2. A PHI1-PHI2 combination (in that order) is known as a "T" state, and is the basis for 386 DX bus cycles.

3.1.2 386 DX CPU/82385 Reset (RESET)

This input resets the 82385, bringing it to an initial known state, and is driven by the same source that drives the 386 DX RESET input. A reset effectively flushes the cache by clearing all cache directory tag valid bits. The falling edge of RESET is synchronized to CLK2, and used by the 82385 to properly establish the phase of its internal clock. (See Figure 3-2.) Specifically, the second internal phase following the falling edge of RESET is PHI2.

3.1.3 386 DX CPU/82385 Address Bus (A2–A31), Byte Enables (BE0#–BE3#), and Cycle Definition Signals (M/IO#, D/C#, W/R#, LOCK#)

The 82385 directly connects to these 386 DX outputs. The 386 DX address bus is used in the cache directory comparison to see if data referenced by 386 DX resides in the cache, and the byte enables inform the 82385 as to which portions of the data bus are involved in an 386 DX cycle. The cycle definition signals are decoded by the 82385 to determine the type of cycle the 386 DX is executing.

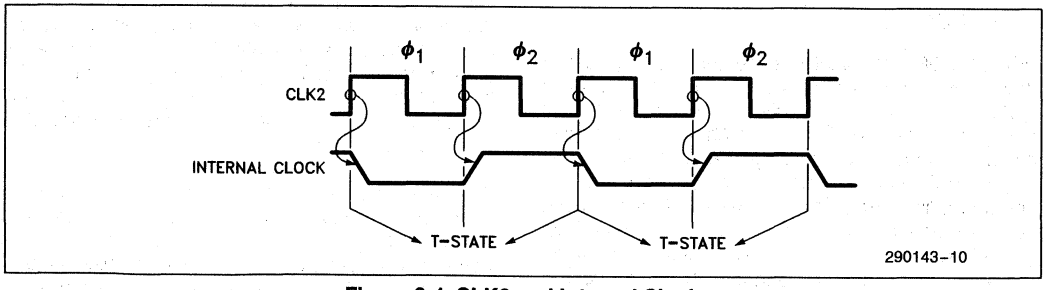


Figure 3-1. CLK2 and Internal Clock

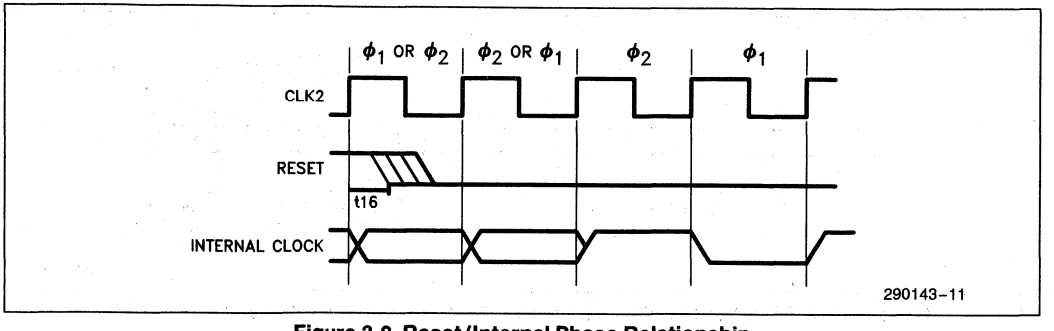


Figure 3-2. Reset/Internal Phase Relationship

3.1.4 386 DX CPU/82385 Address Status (ADS#) and Ready Input (READYI#)

ADS#, a 386 DX output, tells the 82385 that new address and cycle definition information is available. READYI#, an input to both the 386 DX (via the 386 DX READY# input pin) and 82385, indicates the completion of an 386 DX bus cycle. ADS# and READYI# are used to keep track of the 386 DX bus state.

3.1.5 386 DX Next Address Request (NA#)

This 82385 output controls 386 DX pipelining. It can be tied directly to the 386 DX NA# input, or it can be logically "AND"ed with other 386 DX local bus next address requests.

3.1.6 Ready Output (READYO#) and Bus Ready Enable (BRDYEN#)

The 82385 directly terminates all but two types of 386 DX bus cycles with its READYO# output. 386 DX local bus cycles must be terminated by the local device being accessed. This includes devices decoded using the 82385 LBA# signal and 80387 accesses. The other cycles not directly terminated by the 82385 are 82385 local bus reads, specifically cache read misses and non-cacheable reads. (Recall that the 82385 forwards and runs such cycles on the 82385 bus.) In these cycles the signal that terminates the 82385 local bus access is BREADY#, which is gated through to the 386 DX local bus such that the 386 DX and 82385 local bus cycles are concurrently terminated. BRDYEN# is used to gate the BREADY# signal to the 386 DX.

3.2 CACHE CONTROL SIGNALS

These 82385 outputs control the external 32 KB cache data memory.

3.2.1 Cache Address Latch Enable (CALEN)

This signal controls the latch (typically an F or AS series 74373) that resides between the low order 386 DX address bits and the cache SRAM address inputs. (The outputs of this latch are the "cache address" described in the previous chapter.) When CALEN is high the latch is transparent. The falling edge of CALEN latches the current inputs which remain applied to the cache data memory until CALEN returns to an active high state.

3.2.2 Cache Transmit/Receive (CT/R#)

This signal defines the direction of an optional data transceiver (typically an F or AS series 74245) between the cache and 386 DX data bus. When high, the transceiver is pointed towards the 386 DX local data bus (the SRAMs are output enabled). When low, the transceiver points towards the cache data memory. A transceiver is required if the cache is designed with SRAMs that lack an output enable control. A transceiver may also be desirable in a system that has a heavily loaded 386 DX local data bus. These devices are not necessary when using SRAMs which incorporate an output enable.

3.2.3 Cache Chip Selects (CS0# – CS3#)

These active low signals tie to the cache SRAM chip selects, and individually enable the four bytes of the 32-bit wide cache. CS0# enables D0–D7, CS1# enables D8–D15, CS2# enables D16–D23, and CS3# enables D24–D31. During read hits, all four bytes are enabled regardless of whether or not all four 386 DX byte enables are active. (The 386 DX ignores what it did not request.) Also, all four cache bytes are enabled in a read miss so as to update the cache with a complete line (double word). In a write hit, only those cache bytes that correspond to active byte enables are selected. This prevents cache data from being corrupted in a partial doubleword write.

5

3.2.4 Cache Output Enables (COEA#, COEB#) and Write Enables (CWEA#, CWEB#)

COEA# and COEB# are active low signals which tie to the cache SRAM or Transceiver output enables and respectively enable cache bank A or B. The state of DEFOE# (define cache output enable), an 82385 configuration input, determines the functional definition of COEA# and COEB#.

If DEFOE# = V_{IL}, in a two-way set associative cache, either COEA# or COEB# is active during read hit cycles only, depending on which bank is selected. In a direct mapped cache, both are activated during read hits, so the designer is free to use either one. This COEx# definition best suites cache SRAMs with output enables.

If DEFOE# = V_{IH}, COEx# is active during read hit, read miss (cache update) and write hit cycles only. This COEx# definition suites cache SRAMs without output enables. In such systems, transceivers are needed and their output enables must be active for writing, as well as reading, the cache SRAMs.

CWEA# and CWEB# are active low signals which tie to the cache SRAM write enables, and respectively enable cache bank A or B to receive data from the 386 DX data bus (386 DX write hit or read miss update). In a two-way set associative cache, one or the other is enabled in a read miss or write hit. In a direct mapped cache, both are activated, so the designer is free to use either one.

The various cache configurations supported by the 82385 are described in Chapter 4.

3.3 386 DX LOCAL BUS DECODE INPUTS

These 82385 inputs are generated by decoding the 386 DX address and cycle definition lines. These active low inputs are sampled at the end of the first state in which the address of a new 386 DX cycle becomes available (T1 or first T2P).

3.3.1 386 DX Local Bus Access (LBA#)

This input identifies an 386 DX access as directed to a resource (other than the cache) on the 386 DX local bus. (The 387 Numerics Coprocessor is considered a 386 DX local bus resource, but LBA# need not be generated as the 82385 internally decodes 387 accesses.) The 82385 simply ignores these cycles. They are neither forwarded to the system nor do they affect the cache or cache directory. Note that LBA# has priority over all other types of cycles. If LBA# is asserted, the cycle is interpreted as an 386 DX local bus access, regardless of the cycle type or status of NCA# or X16#. This allows any 386 DX cycle (memory, I/O, interrupt acknowledge, etc.) to be kept on the 386 local bus if desired.

3.3.2 Non-Cacheable Access (NCA#)

This active low input identifies a 386 DX cycle as non-cacheable. The 82385 forwards non-cacheable cycles to the 82385 local bus and runs them. The cache and cache directory are unaffected.

NCA# allows a designer to set aside a portion of main memory as non-cacheable. Potential applications include memory-mapped I/O and systems where multiple masters access dual ported memory via different busses. Another possibility makes use of the 386 DX D/C# output. The 82385 by default implements a unified code and data cache, but driving NCA# directly by D/C# creates a data only cache. If D/C# is inverted first, the result is a code only cache.

3.3.3 16-Bit Access (X16#)

X16# is an active low input which identifies 16-bit memory and/or I/O space, and the decoded signal that drives X16# should also drive the 386 DX BS16# input. 16-bit accesses are treated like non-cacheable accesses: they are forwarded to and executed on the 82385 local bus with no impact on the cache or cache directory. In addition, the 82385 locks the two halves of a non-aligned 16-bit transfer from interruption by another master, as does the 386 DX.

3.4 82385 LOCAL BUS INTERFACE SIGNALS

The 82385 presents a "386 DX-like" front end to the system, and the signals discussed in this section are 82385 local bus equivalents to actual 386 DX signals. These signals are named with respect to their 386 DX counterparts, but with the letter "B" appended to the front.

Note that the 82385 itself does not have equivalent output signals to the 386 DX data bus (D0–D31), address bus (A2–A31), and cycle definition signals (M/IO#, D/C#, W/R#). The 82385 data bus (BD0–BD31) is actually the system side of a latching transceiver, and the 82385 address bus and cycle definition signals (BA2–BA31, BM/IO#, BD/C#, BW/R#) are the outputs of an edge-triggered latch. The signals that control this data transceiver and address latch are discussed in Section 3.5.

3.4.1 82385 Bus Byte Enables (BBE0#–BBE3#)

BBE0#–BBE3# are the 82385 local bus equivalents to the 386 DX byte enables. In a cache read miss, the 82385 drives all four signals low, regardless of whether or not all four 386 DX byte enables are active. This ensures that a complete line (doubleword) is fetched from main memory for the cache update. In all other 82385 local bus cycles, the 82385 duplicates the logic levels of the 386 DX byte enables. The 82385 tri-states these outputs when it is not the current bus master.

3.4.2 82385 Bus Lock (BLOCK#)

BLOCK# is the 82385 local bus equivalent to the 386 DX LOCK# output, and distinguishes between locked and unlocked cycles. When the 386 DX runs a locked sequence of cycles (and LBA# is negated), the 82385 forwards and runs the sequence on the 82385 local bus, regardless of whether any locations

referenced in the sequence reside in the cache. A read hit will be run as if it is a read miss, but a write hit will update the cache as well as being completed to system memory. In keeping with 386 DX behavior, the 82385 does not allow another master to interrupt the sequence. BLOCK# is tri-stated when the 82385 is not the current bus master.

3.4.3 82385 Bus Address Status (BADS#)

BADS# is the 82385 local bus equivalent of ADS#, and indicates that a valid address (BA2–BA31, BBE0#–BBE3#) and cycle definition (BM/IO#, BW/R#, BD/C#) is available. It is asserted in BT1 and BT2P states, and is tri-stated when the 82385 does not own the bus.

3.4.4 82385 Bus Ready Input (BREADY#)

82385 local bus cycles are terminated by BREADY#, just as 386 DX cycles are terminated by the 386 DX READY# input. In 82385 local bus read cycles, BREADY# is gated by BRDYEN# onto the 386 DX local bus, such that it terminates both the 386 DX and 82385 local bus cycles.

3.4.5 82385 Bus Next Address Request (BNA#)

BNA# is the 82385 local bus equivalent to the 386 DX NA# input, and indicates that the system is prepared to accept a pipelined address and cycle definition. If BNA# is asserted and the new cycle information is available, the 82385 begins a pipelined cycle on the 82385 local bus.

3.5 82385 BUS DATA TRANSCEIVER AND ADDRESS LATCH CONTROL SIGNALS

The 82385 data bus is the system side of a latching transceiver (typically an F or AS series 74646), and the 82385 address bus and cycle definition signals are the outputs of an edge-triggered latch (F or AS series 74374). The following is a discussion of the 82385 outputs that control these devices. An important characteristic of these signals and the devices they control is that they ensure that BD0–BD31, BA2–BA31, BM/IO#, BD/C#, and BW/R# reproduce the functionality and timing behavior of their 386 DX equivalents.

3.5.1 Local Data Strobe (LDSTB), Data Output Enable (DOE#), and Bus Transmit/Receive (BT/R#)

These signals control the latching data transceiver. BT/R# defines the transceiver direction. When high, the transceiver drives the 82385 data bus in write cycles. When low, the transceiver drives the 386 DX data bus in 82385 local bus read cycles. DOE# enables the transceiver outputs.

The rising edge of LDSTB latches the 386 DX data bus in all write cycles. The interaction of this signal and the latching transceiver is used to perform the 82385's posted write capability.

3.5.2 Bus Address Clock Pulse (BACP) and Bus Address Output Enable (BAOE#)

These signals control the latch that drives BA2–BA31, BM/IO#, BW/R#, and BD/C#. In any 386 DX cycle that is forwarded to the 82385 local bus, the rising edge of BACP latches the 386 DX address and cycle definition signals. BAOE# enables the latch outputs when the 82385 is the current bus master and disables them otherwise.

3.6 STATUS AND CONTROL SIGNALS

5

3.6.1 Cache Miss Indication (MISS#)

This output accompanies cacheable read and write miss cycles. This signal transitions to its active low state when the 82385 determines that a cacheable 386 DX access is a miss. Its timing behavior follows that of the 82385 local bus cycle definition signals (BM/IO#, BD/C#, BW/R#) so that it becomes available with BADS# in BT1 or the first BT2P. MISS# is floated when the 82385 does not own the bus, such that multiple 82385's can share the same node in multi-cache systems. (As discussed in Chapter 7, this signal also serves a reserved function in testing the 82385.)

3.6.2 Write Buffer Status (WBS)

The latching data transceiver is also known as the "posted write buffer." WBS indicates that this buffer contains data that has not yet been written to the system even though the 386 DX may have begun its next cycle. It is activated when 386 DX data is latched, and deactivated when the corresponding

82385 local bus write cycle is completed (BREADY#). (As discussed in Chapter 7, this signal also serves a reserved function in testing the 82385.)

WBS can serve several functions. In multi-processor applications, it can act as a coherency mechanism by informing a bus arbiter that it should let a write cycle run on the system bus so that main memory has the latest data. If any other 82385 cache subsystems are on the bus, they will monitor the cycle via their bus watching mechanisms. Any 82385 that detects a snoop hit will invalidate the corresponding entry in its local cache.

3.6.3 Cache Flush (FLUSH)

When activated, this signal causes the 82385 to clear all of its directory tag valid bits, effectively flushing the cache. (As discussed in Chapter 7, this signal also serves a reserved function in testing the 82385.) The primary use of the FLUSH input is for diagnostics and multi-processor support. The use of this pin as a coherency mechanism may impact software transparency.

The FLUSH input must be held active for at least 4 CLK (8 CLK2) cycles to complete the flush sequence. If FLUSH is still active after 4 CLK cycles, any accesses to the cache will be misses and the cache will not be updated (since FLUSH is active).

3.7 BUS ARBITRATION SIGNALS (BHOLD AND BHLDA)

In master mode, BHOLD is an input that indicates a request by a slave device for bus ownership. The 82385 acknowledges this request via its BHLDA output. (These signals function identically to the 386 DX HOLD and HLDA signals.)

The roles of BHOLD and BHLDA are reversed for an 82385 in slave mode. BHOLD is now an output indicating a request for bus ownership, and BHLDA an input indicating that the request has been granted.

3.8 COHERENCY (BUS WATCHING) SUPPORT SIGNALS (SA2–SA31, SSTB#, SEN)

These signals form the 82385's bus watching interface. The Snoop Address Bus (SA2–SA31) connects to the system address lines if masters reside at both the system and 82385 local bus levels, or the 82385 local bus address lines if masters reside only at the 82385 local bus level. Snoop Strobe (SSTB#) indicates that a valid address is on the snoop address inputs. Snoop Enable (SEN) indicates that the cycle is a write. In a system with masters only at the 82385 local bus level, SA2–SA31, SSTB#, and SEN can be driven respectively by BA2–BA31, BADS#, and BW/R# without any support circuitry.

3.9 CONFIGURATION INPUTS (2W/D#, M/S#, DEFOE#)

These signals select the configurations supported by the 82385. They are hardware strap options and must not be changed dynamically. 2W/D# (2-Way/Direct Mapped Select) selects a two-way set associative cache when tied high, or a direct mapped cache when tied low. M/S# (Master/Slave Select) chooses between master mode (M/S# high) and slave mode (M/S# low). DEFOE# defines the functionality of the 82385 cache output enables (COEA# and COEB#). DEFOE# allows the 82385 to interface to SRAMs with output enables (DEFOE# low) or to SRAMs requiring transceivers (DEFOE# high).

4.0 386 DX LOCAL BUS INTERFACE

The following is a detailed description of how the 82385 interfaces to the 386 DX and to 386 DX local bus resources. Items specifically addressed are the interfaces to the 386 DX, the cache SRAMs, and the 387 Numerics Coprocessor.

The many timing diagrams in this and the next chapter provide insight into the dual pipelined bus structure of a 386 DX CPU/82385 system. It's important to realize, however, that one need not know every possible cycle combination to use the 82385. The interface is simple, and the dual bus operation invisible to the 386 DX and system. To facilitate discussion of the timing diagrams, several conventions have been adopted. Refer to Figure 4-2A, and note that 386 DX bus cycles, 386 DX bus states, and 82385 bus states are identified along the top. All states can be identified by the "frame numbers" along the bottom. The cycles in Figure 4-2A include a cache read hit (CRDH), a cache read miss (CRDM), and a write (WT). WT represents any write, cacheable or not. When necessary to distinguish cacheable writes, a write hit goes by CWTH and a write miss by CWTM. Non-cacheable system reads go by SBRD. Also, it is assumed that system bus pipelining occurs even though the BNA# signal is not shown. When the system pipeline begins is a function of the system bus controller.

386 DX bus cycles can be tracked by ADS# and READYI#, and 82385 cycles by BADS# and BREADY#. These four signals are thus a natural

choice to help track parallel bus activity. Note in the timing diagrams that 386 DX cycles are numbered using ADS# and READYI#, and 82385 cycles using BADS# and BREADY#. For example, when the address of the first 386 DX cycle becomes available, the corresponding assertion of ADS# is marked "1", and the READYI# pulse that terminates the cycle is marked "1" as well. Whenever a 386 DX cycle is forwarded to the system, its number is forwarded as well so that the corresponding 82385 bus cycle can be tracked by BADS# and BREADY#.

The "N" value in the timing diagrams is the assumed number of main memory wait states inserted in a non-pipelined 82385 bus cycle. For example, a non-pipelined access to N=2 memory requires a total of four bus states, while a pipelined access requires three. (The pipeline advantage effectively hides one main memory wait state.)

4.1 PROCESSOR INTERFACE

This section presents the 386 DX CPU /82385 hardware interface and discusses the interaction and timing of this interface. Also addressed is how to decode the 386 DX address bus to generate the 82385 inputs LBA#, NCA#, and X16#. (Recall that LBA# allows memory and/or I/O space to be set aside for 386 DX local bus resources; NCA# allows system memory to be set aside as non-cacheable; and X16# allows system memory and/or I/O space to be reserved for 16-bit resources.) Finally, the 82385's handling of 16-bit space is discussed.

4.1.1 Hardware Interface

Figure 4-1 is a diagram of an 386 DX CPU/82385 system, which can be thought of as three distinct interfaces. The first is the 386 DX CPU/82385 interface (including the Ready Logic). The second is the cache interface, as depicted by the cache control bus in the upper left corner of Figure 4-1. The third is the 82385 bus interface, which includes both direct connects and signals that control the 74374 address/cycle definition latch and 74646 latching data transceiver. (The 82385 bus interface is the subject of the next chapter.)

As seen in Figure 4-1, the 386 DX CPU/82385 interface is a straightforward connection. The only necessary support logic is that required to sum all ready sources.

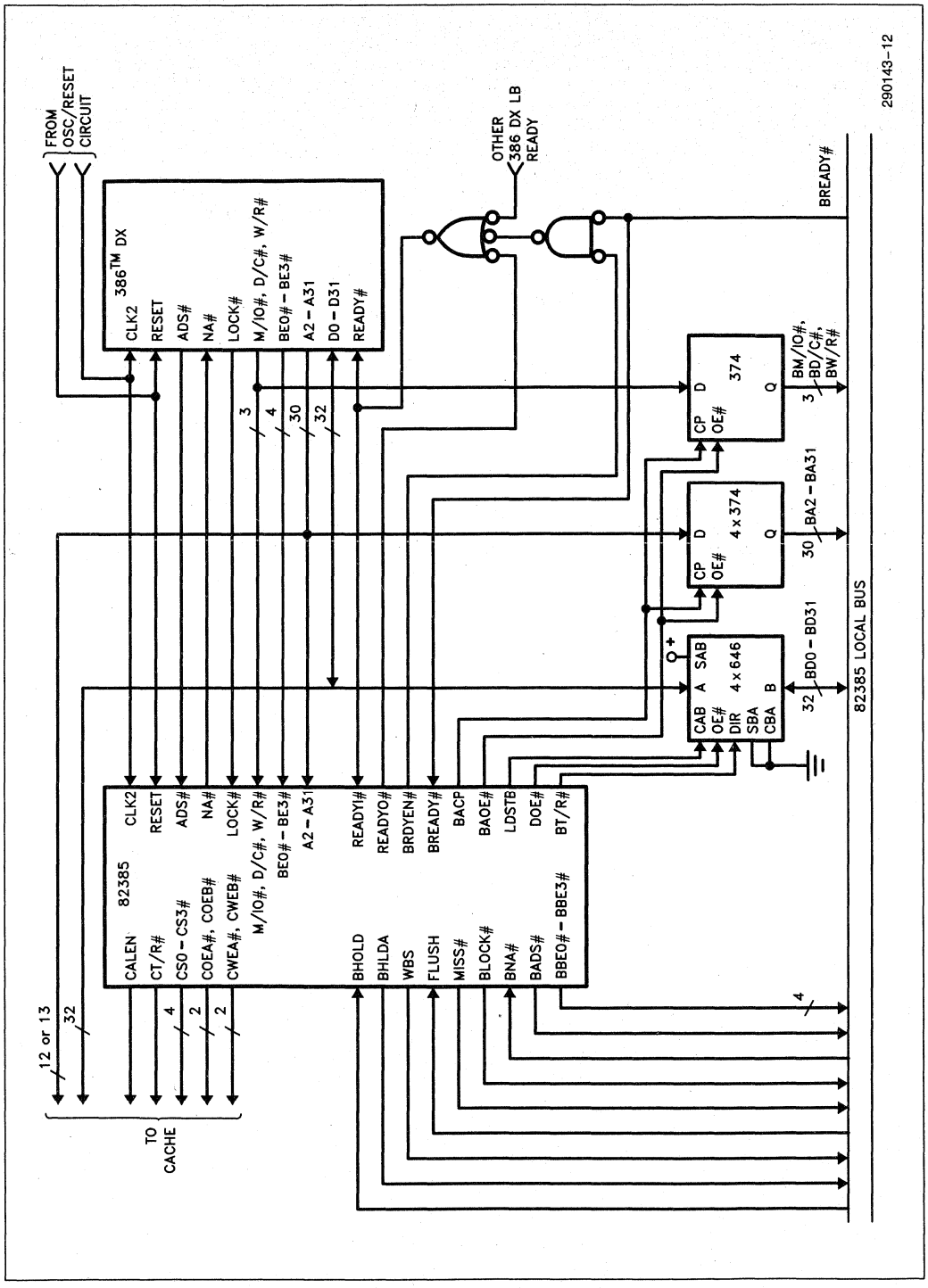


Figure 4-1. 386 DX CPU/82385 Interface

4.1.2 Ready Generation

Note in Figure 4-1 that the ready logic consists of two gates. The upper three-input AND gate (shown as a negative logic OR) sums all 386 DX local bus ready sources. One such source is the 82385 READYO# output, which terminates read hits and posted writes. The output of this gate drives the 386 DX READY# input and is monitored by the 82385 (via READYI#) to track the 386 DX bus state.

When the 82385 forwards a 386 DX read cycle to the 82385 bus (cache read miss or non-cacheable read), it does not directly terminate the cycle via READYO#. Instead, the 386 DX and 82385 bus cycles are concurrently terminated by a system ready source. This is the purpose of the additional two-input OR gate (negative logic AND) in Figure 4-1. When the 82385 forwards a read to the 82385 bus, it asserts BRDYEN# which enables the system ready signal (BREADY#) to directly terminate the 386 DX bus cycle.

Figures 4-2A and 4-2B illustrate the behavior of the signals involved in ready generation. Note in cycle 1 of Figure 4-2A that the 82385 READYO# directly terminates the hit cycle. In cycle 2, READYO# is not activated. Instead the 82385 BRDYEN# is activated in BT2, BT2P, or BT2I states such that BREADY# can concurrently terminate the 386 DX and 82385 bus cycles (frame 6). Cycle 3 is a posted write. The write data becomes available in T1P (frame 7), and

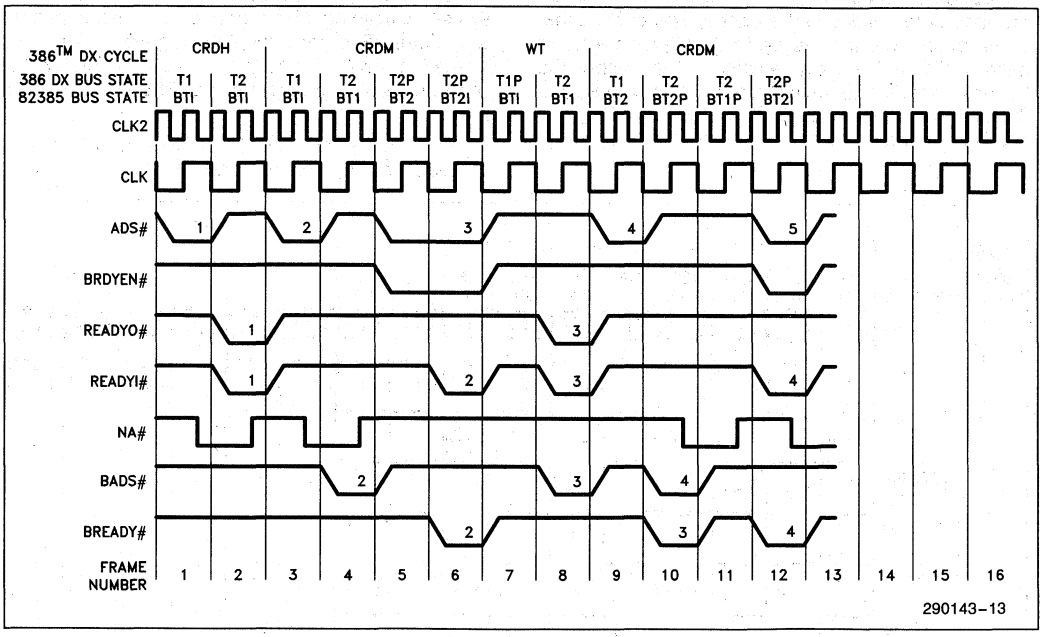


Figure 4-2A. READYO#, BRDYEN#, and NA# (N = 1)

the address, data, and cycle definition of the write are latched in T2 (frame 8). The 386 DX cycle is terminated by READYO# in frame 8 with no wait states. The 82385, however, sees the write cycle through to completion on the 82385 bus where it is terminated in frame 10 by BREADY#. In this case, the BREADY# signal is not gated through to the 386 DX . Refer to Figures 4-2A and 4-2B for clarification.

4.1.3. NA# and 386 DX Local Bus Pipelining

Cycle 1 of Figure 4-2A is a typical cache read hit. The 386 DX address becomes available in T1, and the 82385 uses this address to determine if the referenced data resides in the cache. The cache look-up is completed and the cycle qualified as a hit or miss in T1. If the data resides in the cache, the cache is directed to drive the 386 DX data bus, and the 82385 drives its READYO# output so the cycle can be terminated at the end of the first T2 with no wait states.

Although cycle 2 starts out like cycle 1, at the end of T1 (frame 3), it is qualified as a miss and forwarded to the 82385 bus. The 82385 bus cycle begins one state after the 386 DX bus cycle, implying a one wait state overhead associated with cycle 2 due to the look-up. When the 82385 encounters the miss, it immediately asserts NA#, which puts the 386 DX into pipelined mode. Once in pipelined mode, the 82385 is able to qualify an 386 DX cycle using the 386 DX pipelined address and control signals. The result is that the cache look-up state is hidden in all but the first of a contiguous sequence of read misses. This is shown in the first two cycles, both read misses, of Figure 4-2B. The CPU sees the look-up state in the first cycle, but not in the second. In fact, the second miss requires a total of only two states, as not only does 386 DX pipelining hide the look-up state, but system pipelining hides one of the main memory wait states. (System level pipelining via BNA# is discussed in the next chapter.) Several characteristics of the 82385's pipelining of the 386 DX are as follows:

— The above discussion applies to all system reads, not just cache read misses.

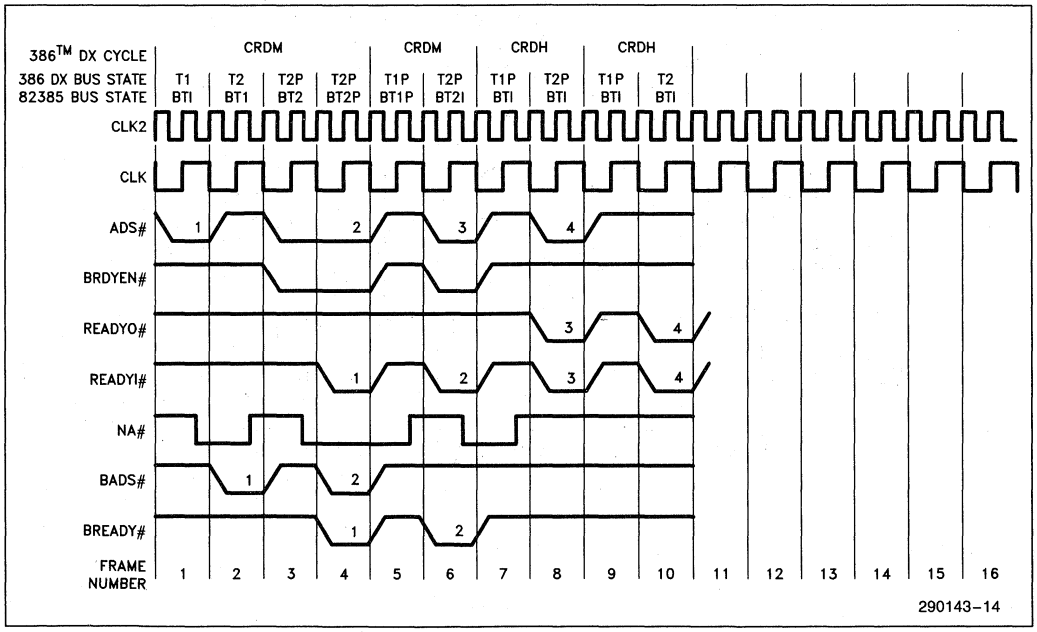


Figure 4-2B. READYO#, BRDYEN#, and NA# (N = 1)

— The 82385 provides the fastest possible switch to pipelining, T1-T2-T2P. The exception to this is when a system read follows a posted write. In this case, the sequence is T1-T2-T2-T2P. (Refer to cycle 4 of Figure 4-2A.) The number of T2 states is dependent on the number of main memory wait states.

— Refer to the read hit in Figure 4-2A (cycle 1), and note that NA# is actually asserted before the end of T1, before the hit/miss decision is made. This is of no consequence since even though NA# is sampled active in T2, the activation of READYO# in the same T2 renders NA# a "don't care". NA# is asserted in this manner to meet 386 DX timing requirements and to ensure the fastest possible switch to pipelined mode.

— All read hits and the majority of writes can be serviced by the 82385 with zero wait states in non-pipelined mode, and the 82385 accordingly attempts to run all such cycles in non-pipelined mode. An exception is seen in the hit cycles (cycles 3 and 4) of Figure 4-2B. The 82385 does not know soon enough that cycle 3 is a hit, and thus sustains the pipeline. The result is that three sequential hits are required before the 386 DX is totally out of pipelined mode. (The three hits look like T1P-T2P, T1P-T2, T1-T2.) Note that this does not occur if the number of main memory wait states is equal to or greater than two.

As far as the design is concerned, NA# is generally tied directly to the 386 DX NA# input. However, other local NA# sources may be logically "AND"ed with the 82385 NA# output if desired. It is essential, however, that no device other than the 82385 drive the 386 DX NA# input unless that device resides on the 386 DX local bus in space decoded via LBA#. If desired, the 82385 NA# output can be ignored and the 386 DX NA# input tied high. The 386 DX NA# input should never be tied low, which would always keep it active.

4.1.4 LBA#, NCA#, and X16# Generation

The 82385 input signals LBA#, NCA# and X16# are generated by decoding the 386 DX address (A2–A31) and cycle definition (W/R#, D/C#, M/IO#) lines. The 82385 samples them at the end of the first state in which they become available, which is either T1 or the first T2P cycle. The decode configuration and timings are illustrated respectively in Figures 4-3A and 4-3B.

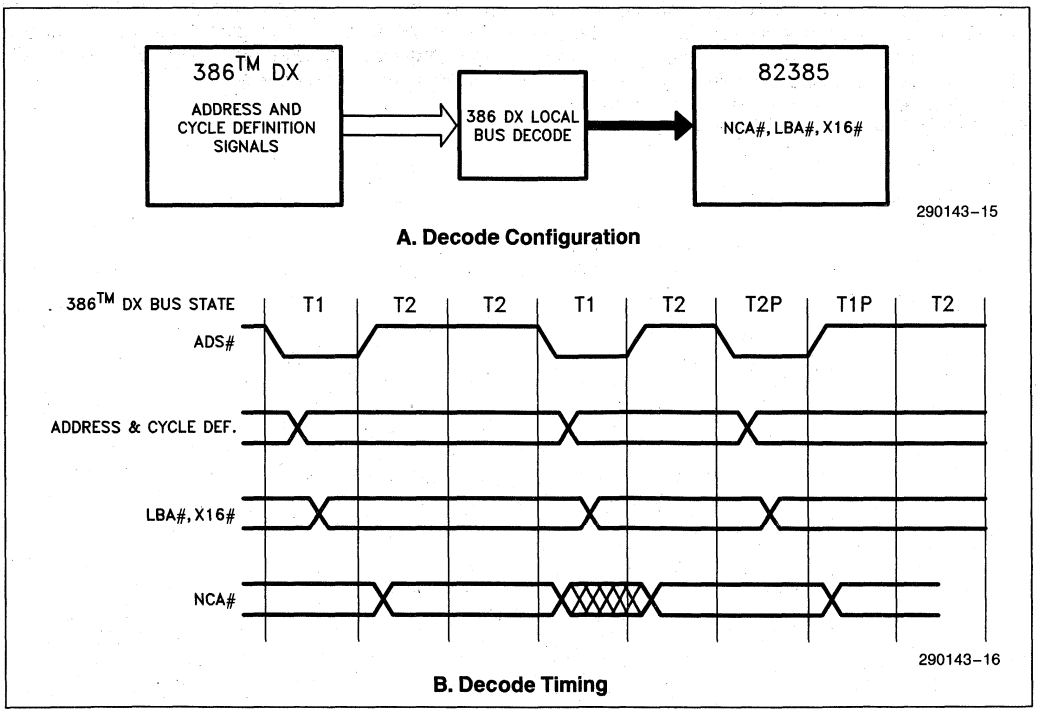


Figure 4-3. NCA#, LBA#, X16# Generation

4.1.5 82385 Handling of 16-Bit Space

As discussed previously, the 82385 does not cache devices decoded as 16-bit. Instead it makes provision to accommodate 16-bit space as non-cacheable via the X16# input. X16# is generated when the user decodes the 386 DX address and cycle definition lines for the BS16# input of the 386 DX (Figure 4-3). The decode output now drives both the 386 DX BS16# input and the 82385 X16# input. Cycles decoded this way are treated as non-cacheable. They are forwarded to and executed on the 82385 bus, but have no impact on the cache or cache directory. The 82385 also monitors the 386 DX byte enables in a 16-bit cycle to see if an additional cycle is required to complete the transfer. Specifically, a second cycle is required if (BE0# OR BE1#) AND (BE2# OR BE3#) is asserted in the current cycle. The 82385, like the 386 DX , will not allow the two halves of a 16-bit transfer to be interrupted by another master. There is an important distinction between the handling of 16-bit space in a 386 DX system with an 82385 as compared to a system without an 82385. The 386 DX BS16# input need not be asserted until the last state of a 16-bit cycle for the 386 DX to recognize it as such. The 82385, however, needs the information earlier, specifically at the end of the first 386 DX bus state (T1 or first T2P) in which the address of the 16-bit cycle becomes available. The result is that in a system without an 82385, 16-bit devices can define themselves as 16-bit devices "on the fly", while in a system with an 82385, 16-bit devices should be located in space set aside for 16-bit devices via the X16# decode.

4.2 CACHE INTERFACE

The following is a description of the external data cache and 82385 cache interface.

4.2.1 Cache Configurations

The 82385 controls the cache memory via the control signals shown in Figure 4-1. These signals drive one of four possible cache configurations, as depicted in Figures 4-4A through 4-4D. Figure 4-4A shows a direct mapped cache organized as 8K doublewords. The likely design choice is four 8K x 8 SRAMs. Figure 4-4B depicts the same cache memory but with a data transceiver between the cache and 386 DX data bus. In this configuration, CT/R# controls the transceiver direction, COEA# drives the transceiver output enable. (COEB# could also be used, and DEFOE# is strapped high.) A data buffer is required if the chosen SRAM does not have a separate output enable. Additionally, buffers may be used to ease SRAM timing requirements or in a system with a heavily loaded data bus. (Guidelines for SRAM selection are included in Chapter 6.)

Figure 4-4C depicts a two-way set associative cache organized as two banks (A and B) of 4K doublewords each. The likely design choice is sixteen 4K x 4 SRAM's. Finally, Figure 4-4D depicts the two-way organization with data buffers between the cache memory and data bus.

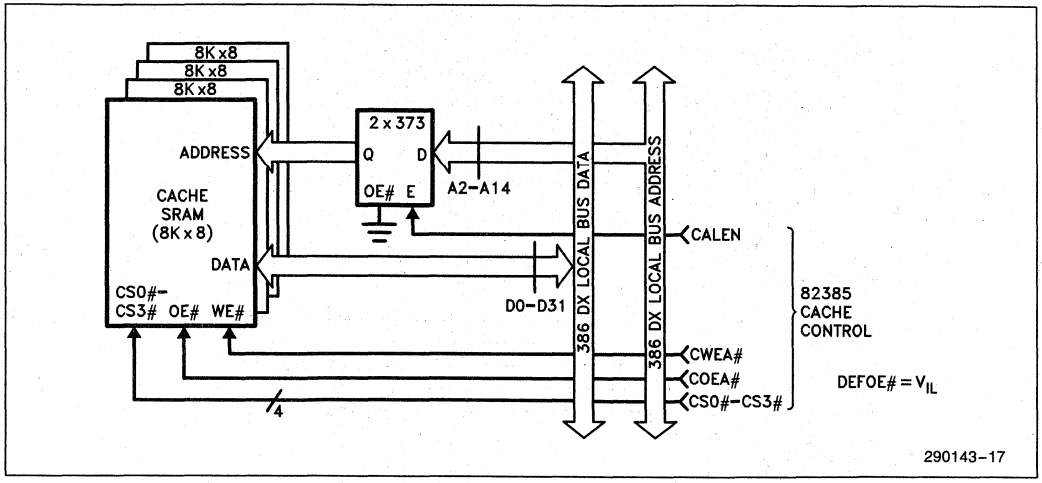


Figure 4-4A. Direct Mapped Cache without Data Buffers

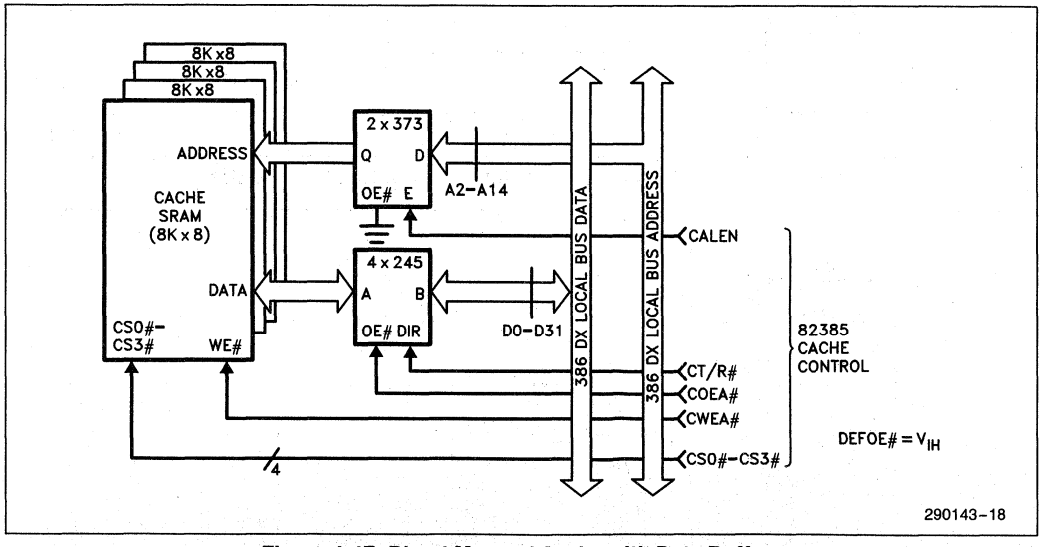


Figure 4-4B. Direct Mapped Cache with Data Buffers

5

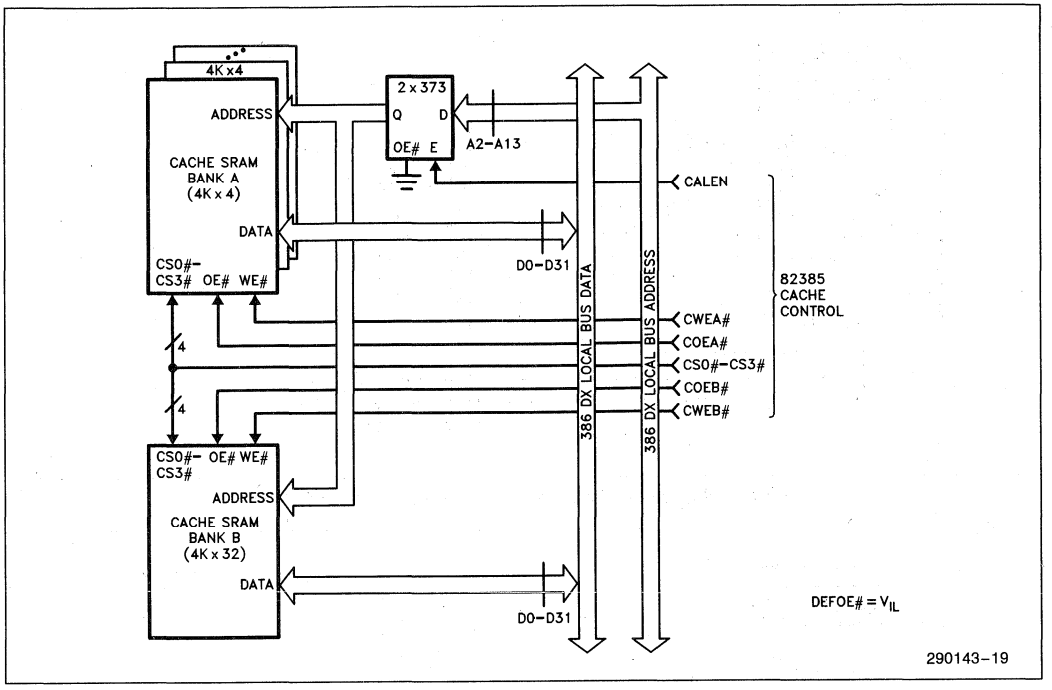


Figure 4-4C. Two-Way Set Associative Cache without Data Buffers

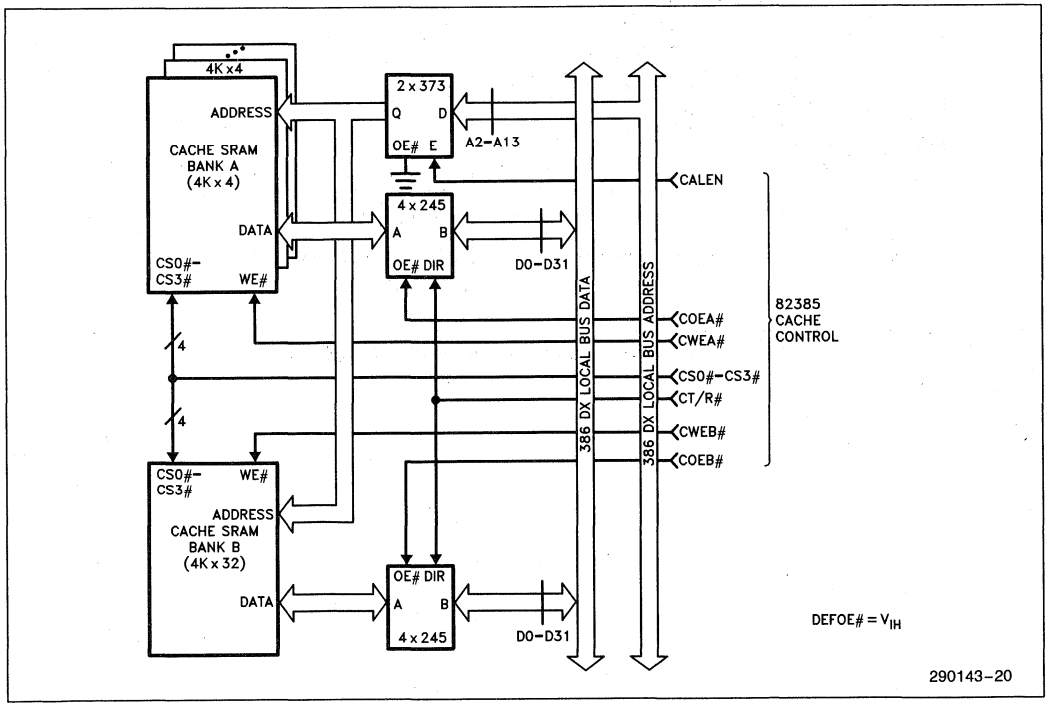


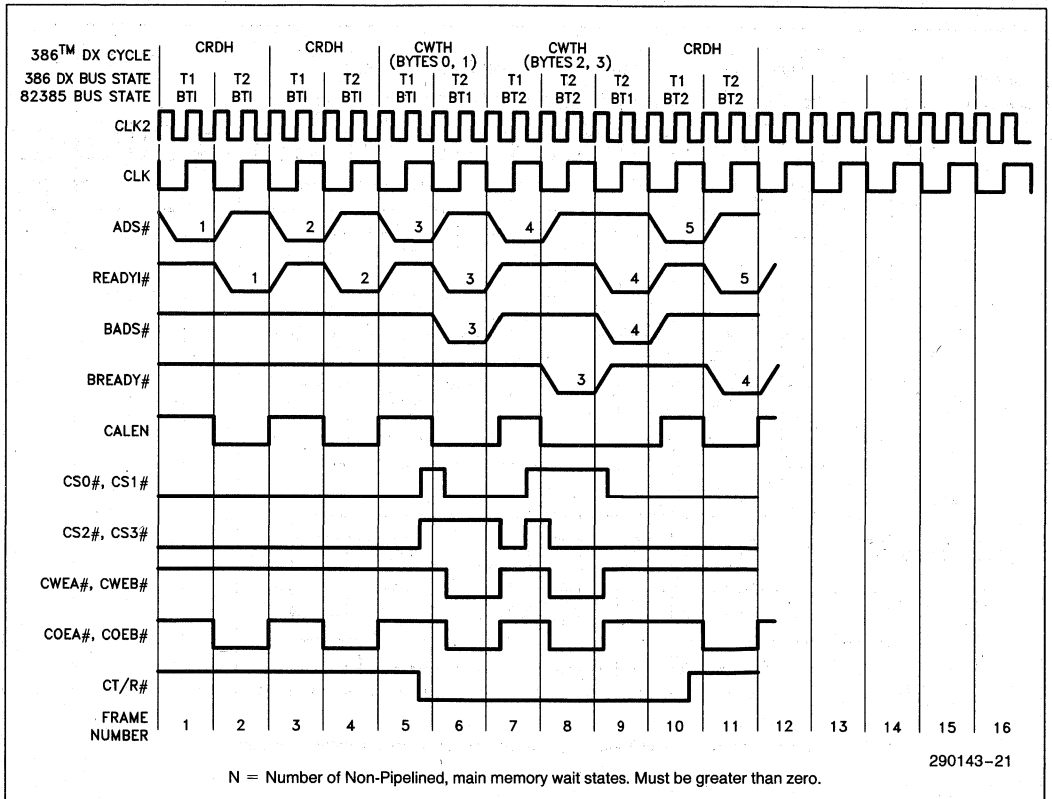
Figure 4-4D. Two-Way Set Associative Cache with Data Buffers

4.2.2 Cache Control—Direct Mapped

Figure 4-5A illustrates the timing of cache read and write hits, while Figure 4-5B illustrates cache updates. In a read hit, the cache output enables are driven from the beginning of T2 (cycle 1 of Figure 4-5A). If at the end of T1 the cycle is qualified as a cacheable read, the output enables are asserted on the assumption that the cycle will be a hit. (Driving the output enables before the actual hit/miss decision is made eases SRAM timing requirements.)

Cycle 1 of Figure 4-5B illustrates what happens when the assumption of a hit turns out to be wrong. Note that the output enables are asserted at the beginning of T2, but then disabled at the end of T2. Once the output enables are inactive, the 82385 turns the transceiver around (via CT/R#) and drives the write enables to begin the cache update cycle. Note in Figure 4-5B that once the 386 DX is in pipelined mode, the output enables need not be driven prior to a hit/miss decision, since the decision is made earlier via the pipelined address information.

One consequence of driving the output enables low in a miss before the hit/miss decision is made is that since the cache starts driving the 386 DX data bus,



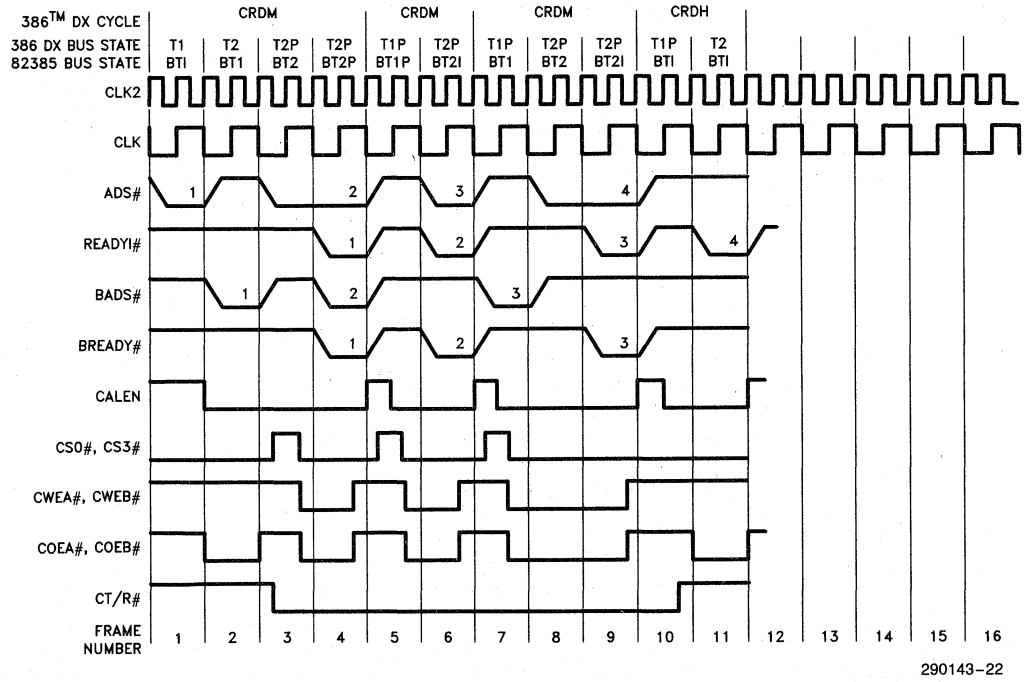
N = Number of Non-Pipelined, main memory wait states. Must be greater than zero.

NOTES:
CRDH = Cache Read Hit
CWTH = Cache Write Hit

Figure 4-5A. Cache Read and Write Cycles—Direct Mapped (N = 1)

the 82385 cannot enable the 74646 transceiver (Figure 4-1) until after the cache outputs are disabled. (The timing of the 74646 control signals is described in the next chapter.) The result is that the 74646 cannot be enabled soon enough to support N=0 main memory ("N" was defined in section 4.0 as the number of non-pipelined main memory wait states). This means that memory which can run with zero wait states in a non-pipelined cycle should not be mapped into cacheable memory. This should not present a problem, however, as a main memory system built with N=0 memory has no need of a cache. (The main memory is as fast as the cache.) Zero wait state memory can be supported if it is decoded as non-cacheable. The 82385 knows that a cycle is non-cacheable in time not to drive the cache output enables, and can thus enable the 74646 sooner.

In a write hit, the 82385 only updates the cache bytes that are meant to be updated as directed by the 386 DX byte enables. This prevents corrupting cache data in partial doubleword writes. Note in Figure 4-5A that the appropriate bytes are selected via the cache byte select lines CS0#–CS3#. In a read hit, all four select lines are driven as the 386 DX will simply ignore data it does not need. Also, in a cache update (read miss), all four selects are active in order to update the cache with a complete line (doubleword).



N = Number of Non-Pipelined, main memory wait states. Must be greater than zero.

NOTE:
CRDM = Cache Read Miss

Figure 4-5B. Cache Update Cycles—Direct Mapped (N=1)

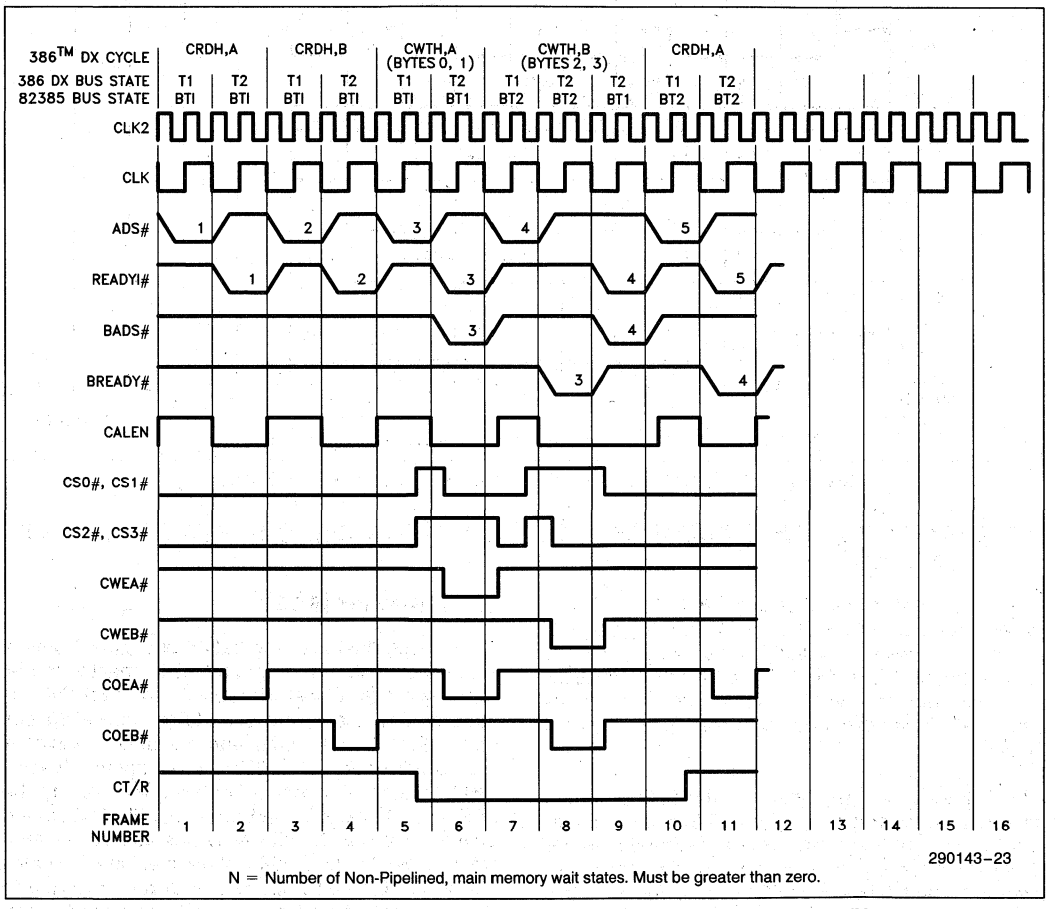
4.2.3 Cache Control—Two-Way Set Associative

Figures 4-6A and 4-6B illustrate the timing of cache read hits, write hits, and updates for a two-way set associative cache. (Note that the cycle sequences are the same as those in Figures 4-5A and 4-5B.) In a cache read hit, only one bank on the other is enabled to drive the 386 DX data bus, so unlike the control of a direct mapped cache, the appropriate cache output enable cannot be driven until the outcome of the hit/miss decision is known. (This implies stricter SRAM timing requirements for a two-way set associative cache.) In write hits and read misses, only one bank or the other is updated.

4.3 387™ DX INTERFACE

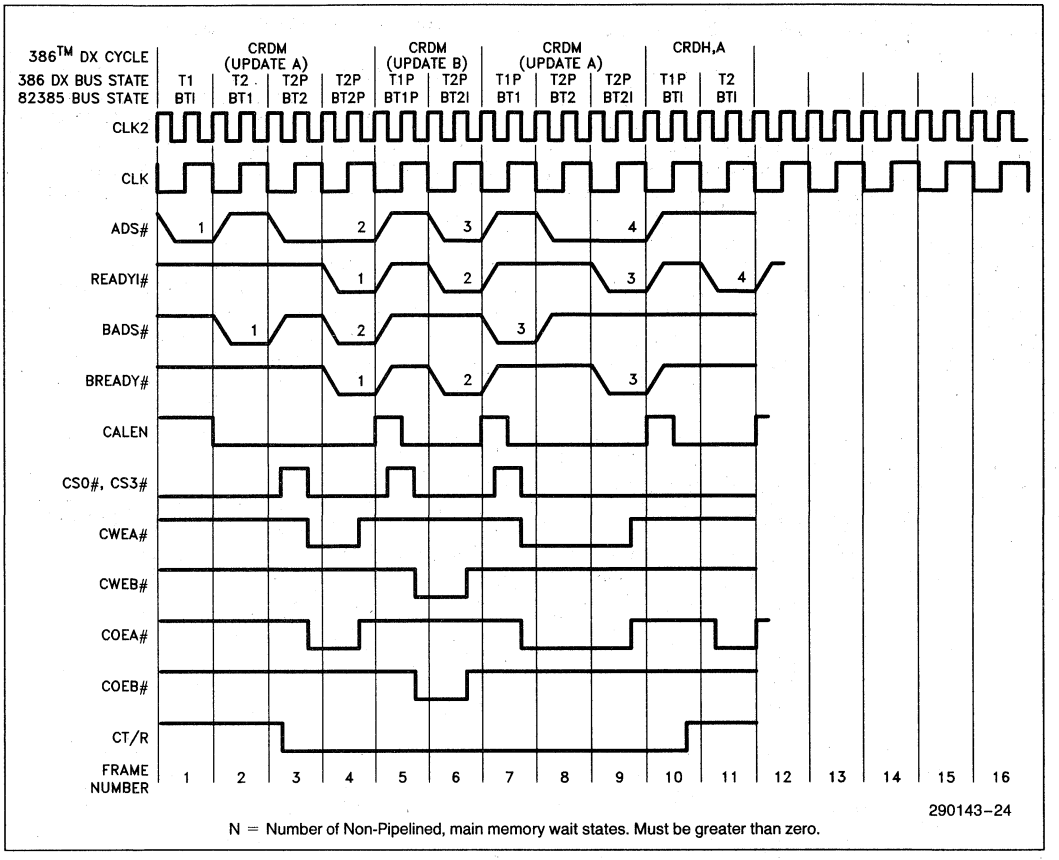
The 387 DX Math Coprocessor interfaces to the 386 DX just as it would in a system without an 82385. The 387 DX READYO# output is logically "AND"ed along with all other 386 DX local bus ready sources (Figure 4-1), and the output is fed to the 387 DX READY#, 82385 READYI#, and 386 DX READY# inputs.

The 386 DX uniquely addresses the 387 DX by driving M/IO# low and A31 high. The 82385 decodes this internally and treats 387 DX accesses in the same way it treats 386 DX cycles in which LBA# is asserted, it ignores them.



N = Number of Non-Pipelined, main memory wait states. Must be greater than zero.

Figure 4-6A. Cache Read and Write Cycles—Two Way Set Associative (N = 1)



N = Number of Non-Pipelined, main memory wait states. Must be greater than zero.

Figure 4-6B. Cache Update Cycles—Two Way Set Associative (N = 1)

5.0 82385 LOCAL BUS AND SYSTEM INTERFACE

The 82385 system interface is the 82385 Local Bus, which presents a "386 DX -like" front end to the system. The system ties to it just as it would to a 386 DX . Although this 386 DX -like front end is functionally equivalent to a 386 DX , there are timing differences which can easily be accounted for in a system design.

The following is a description of the 82385 system interface. After presenting the 82385 bus state machine, the 82385 bus signals are described, as are techniques for accommodating any differences between the 82385 bus and 386 DX bus. Following this is a discussion of the 82385's condition upon reset.

5.1 THE 82385 BUS STATE MACHINE

5.1.1 Master Mode

Figure 5-1A illustrates the 82385 bus state machine when the 82385 is programmed in master mode. Note that it is almost identical to the 386 DX bus state machine, only the bus states are 82385 bus states (BT1P, BTH, etc.) and the state transitions are conditioned by 82385 bus inputs (BNA#, BHOLD, etc.). Whereas a "pending request" to the 386 DX state machine indicates that the 386 DX execution or prefetch unit needs bus access, a pending request to the 82385 state machine indicates that a 386 DX bus cycle needs to be forwarded to the system (read miss, non-cacheable read, write,

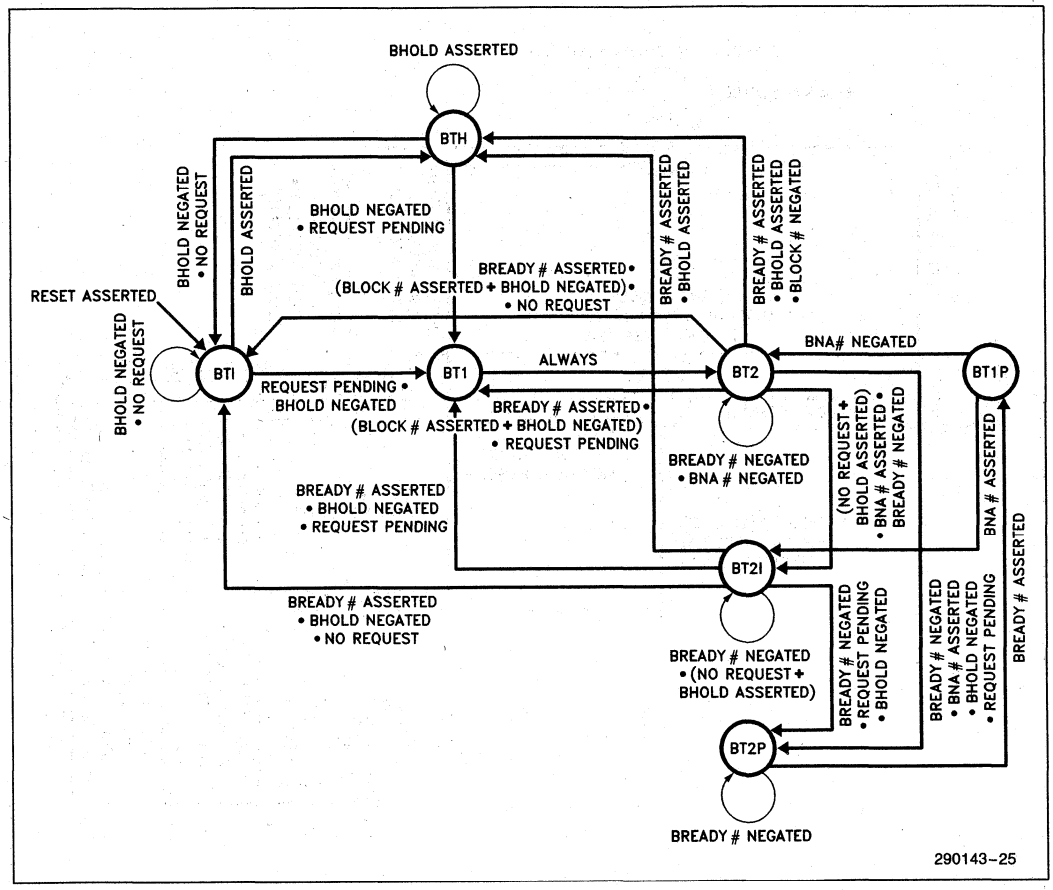


Figure 5-1A. 82385 Local Bus State Machine—Master Mode

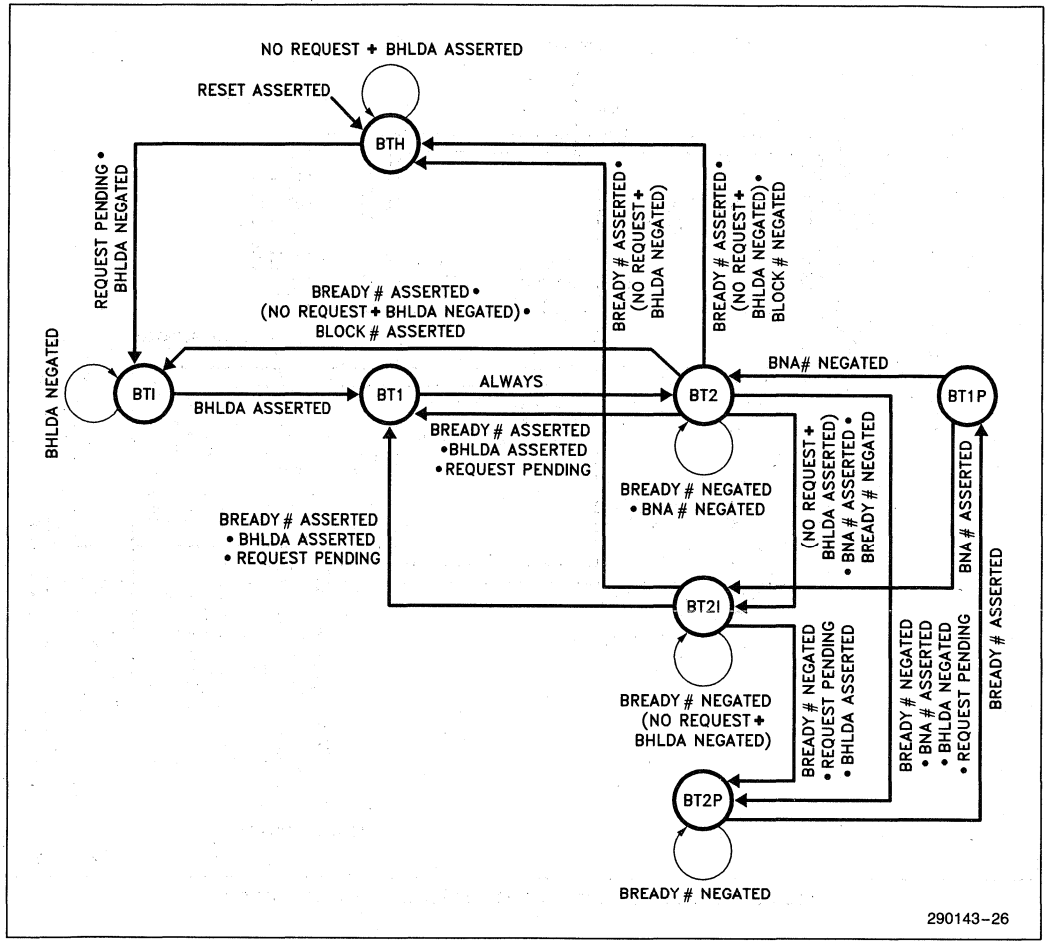


Figure 5-1B. 82385 Local Bus State Machine—Slave Mode

etc.). The only difference between the state machines is that the 82385 does not implement a direct BT1P–BT2P transition. If BNA# is asserted in BT1P, the resulting state sequence is BT1P-BT2I-BT2P. The 82385's ability to sustain a pipeline is not affected by the lack of this state transition.

5.1.2 Slave Mode

The 82385's slave mode state machine (Figure 5-1B) is similar to the master mode machine except that now transitions are conditioned by BHLDA rather than BHOLD. (Recall that in slave mode, the roles of BHOLD and BHLDA are reversed from their master mode roles.) Figure 5-2 clarifies slave mode state machine operation. Upon reset, a slave mode 82385 enters the BTH state. When the 386 DX of the slave 82385 subsystem has a cycle that needs to be forwarded to the system, the 82385 moves to BTI and issues a hold request via BHOLD. It is important to note that a slave mode 82385 does not drive the bus in a BTI state. When the master or bus arbiter returns BHLDA, the slave 82385 enters BT1 and runs the cycle. When the cycle is completed, and if no additional requests are pending, the 82385 moves back to BTH and disables BHOLD.

If, while a slave 82385 is running a cycle, the master or arbiter drops BHLDA (Figure 5-2B), the 82385 will complete the current cycle, move to BTH and remove the BHOLD request. If the 82385 still had cycles to run when it was kicked off the bus, it will immediately assert a new BHOLD and move to BTI to await bus acknowledgement. Note, however, that it will only move to BTI if BHLDA is negated, ensuring that the handshake sequence is completed.

There are several cases in which a slave 82385 will not immediately release the bus if BHLDA is dropped. For example, if BHLDA is dropped during a BT2P state, the 82385 has already committed to the next system bus pipelined cycle and will execute it before releasing the bus. Also, the 82385 will complete the second half of a two-cycle 16-bit transfer, or will complete a sequence of locked cycles before releasing the bus. This should not present any problems, as a properly designed arbiter will not assume that the 82385 has released the bus until it sees BHOLD become inactive.

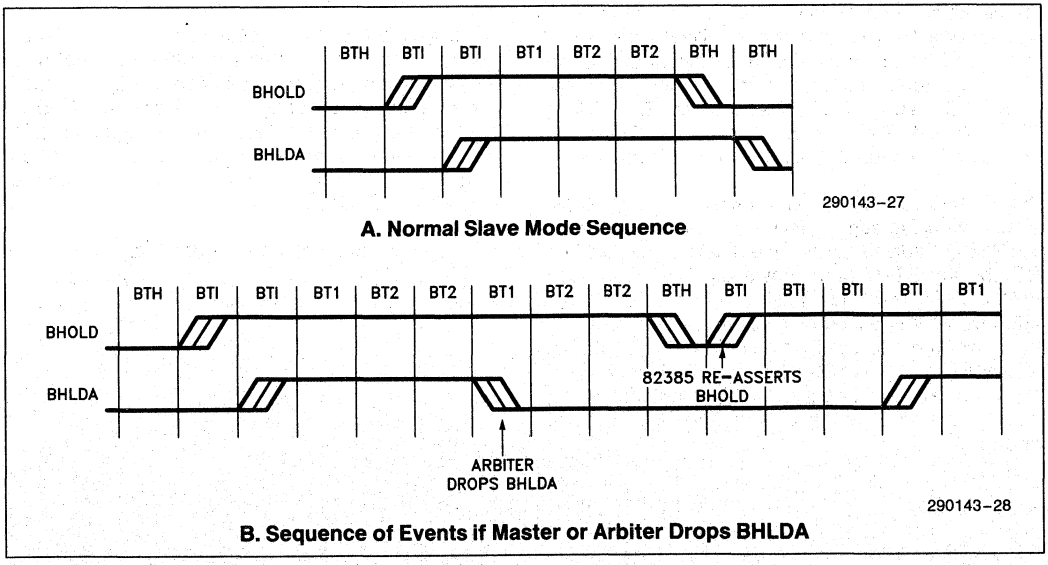


Figure 5-2. BHOLD/BHLDA—Slave Mode

5.2 The 82385 Local Bus

The 82385 bus can be broken up into two groups of signals: those which have direct 386 DX counterparts, and additional status and control signals provided by the 82385. The operation and interaction of all 82385 bus signals are depicted in Figures 5-3A through 5-3L for a wide variety of cycle sequences. These diagrams serve as a reference for the 82385 bus discussion and provide insight into the dual bus operation of the 82385.

5.2.1 82385 Bus Counterparts to 386 DX Signals

The following sections discuss the signals presented on the 82385 local bus which are functional equivalents to the signals present at the 386 DX local bus.

5.2.1.1 ADDRESS BUS (BA2–BA31) AND CYCLE DEFINITION SIGNALS (BM/IO#, BD/C#, BW/R#)

These signals are not driven directly by the 82385, but rather are the outputs of the 74374 address/cycle definition latch. (Refer to Figure 4-1 for the hardware interface.) This latch is controlled by the 82385 BACP and BAOE# outputs. The behavior and timing of these outputs and the latch they control (typically F or AS series TTL) ensure that BA2–BA31, BM/IO#, BW/R#, and BD/C# are compatible in timing and function to their 386 DX counterparts.

The behavior of BACP can be seen in Figure 5-3B, where the rising edge of BACP latches and forwards the 386 DX address and cycle definition signals in a BT1 or first BT2P state. However, the 82385 need not be the current bus master to latch the 386 DX address, as evidenced by cycle 4 of Figure 5-3A. In this case, the address is latched in frame 8, but not forwarded to the system (via BAOE#) until frame 10. (The latch and output enable functions of the 74374 are independent and invisible to one another.)

Note that in frames 2 and 6 the BACP pulses are marked "False." The reason is that BACP is issued and the address latched before the hit/miss determination is made. This ensures that should the cycle be a miss, the 82385 bus can move directly into BT1 without delay. In the case of a hit, the latched address is simply never qualified by the assertion of BADS#. The 82385 bus stays in BTI if there is no access pending (new cycle is a hit) and no bus activity. It will move to and stay in BT2I if the system has requested a pipelined cycle and the 82385 does not have a pending bus access (new cycle is a hit).

5.2.1.2 DATA BUS (BD0–BD31)

The 82385 data bus is the system side of the 74646 latching transceiver. (See Figure 4-1.) This device is controlled by the 82385 outputs LDSTB, DOE#, and BT/R#. LDSTB latches data in write cycles, DOE# enables the transceiver outputs, and BT/R# controls the transceiver direction. The interaction of these signals and the transceiver is such that BD0–BD31 behave just like their 386 DX counterparts. The transceiver is configured such that data flow in write cycles (A to B) is latched, and data flow in read cycles (B to A) is flow-through.

Although BD0–BD31 function just like their 386 DX counterparts, there is a timing difference that must be accommodated for in a system design. As mentioned above, the transceiver is transparent during read cycles, so the transceiver propagation delay must be added to the 386 DX data setup. In addition, the cache SRAM setup must be accommodated for in cache read miss cycles.

For non-cacheable reads the data setup is given by:

Min BD0–BD31 Read Data Setup	=	386 DX Min Data Setup	+	74646 B-to-A Max Propagation Delay

The required BD0–BD31 setup in a cache read miss is given by:

Min BD0–BD31 Read Data Setup	=	74646 B-to-A Max Propagation Delay	+	Cache SRAM Min Write Setup		
	+	One CLK2 Period	–	82385 CWEA# or CWEB# Min Delay		

If a data buffer is located between the 386 DX data bus and the cache SRAMs, then its maximum propagation delay must be added to the above formula as well. A design analysis should be completed for every new design to determine actual margins.

A design can accommodate the increased data setup by choosing appropriately fast main memory DRAMs and data buffers. Alternatively, a designer may deal with the longer setup by inserting an extra wait state into cache read miss cycles. If an additional state is to be inserted, the system bus controller should sample the 82385 MISS# output to distinguish read misses from cycles that do not require the longer setup. Tips on using the 82385 MISS# signal are presented later in this chapter.

The behavior of LDSTB, DOE#, and BT/R# can be understood via Figures 5-3A through 5-3L. Note that in cycle 1 of Figure 5-3A (a non-cacheable system read), DOE# is activated midway through BT1, but in cycle 1 of Figure 5-3B (a cache read miss), DOE# is not activated until midway through BT2. The reason is that in a cacheable read cycle, the cache SRAMs are enabled to drive the 386 DX data bus before the outcome of the hit/miss decision (in anticipation of a hit). In cycle 1 of Figure 5-3B, the assertion of DOE# must be delayed until after the 82385 has disabled the cache output buffers. The result is that N=0 main memory should not be mapped into the cache.

5.2.1.3 BYTE ENABLES (BBE0#–BBE3#)

These outputs are driven directly by the 82385, and are completely compatible in timing and function with their 386 DX counterparts. When a 386 DX cycle is forwarded to the 82385 bus, the 386 DX byte enables are duplicated on BBE0#–BBE3#. The one exception is a cache read miss, during which BBE0#–BBE3# are all active regardless of the status of the 386 DX byte enables. This ensures that the cache is updated with a valid 32-bit entry.

5.2.1.4 ADDRESS STATUS (BADS#)

BADS# is identical in function and timing to its 386 DX counterpart. It is asserted in BT1 and BT2P states, and indicates that valid address and cycle definition (BA2–BA31, BBE0#–BBE3#, BM/IO#, BW/R#, BD/C#) information is available on the 82385 bus.

5.2.1.5 READY (BREADY#)

The 82385 BREADY# input terminates 82385 bus cycles just as the 386 DX READY# input terminates 386 DX bus cycles. The behavior of BREADY# is the same as that of READY#, but note in the A.C. timing specifications that a cache read miss requires a longer BREADY# setup than do other cycles. This must be accommodated for in ready logic design.

5.2.1.6 NEXT ADDRESS (BNA#)

BNA# is identical in function and timing to its 386 DX counterpart. Note that in Figures 5-3A through 5-3L, BNA# is assumed asserted in every BT1P or first BT2 state. Along with the 82385's pipelining of the 386 DX , this ensures that the timing diagrams accurately reflect the full pipelined nature of the dual bus structure.

5.2.1.7 BUS LOCK (BLOCK#)

The 386 DX flags a locked sequence of cycles by asserting LOCK#. During a locked sequence, the 386 DX does not acknowledge hold requests, so the sequence executes without interruption by another master. The 82385 forces all locked 386 DX cycles to run on the 82385 bus (unless LBA# is active), regardless of whether or not the referenced location resides in the cache. In addition, a locked sequence of 386 DX cycles is run as a locked sequence on the 82385 bus; BLOCK# is asserted and the 82385 does not allow the sequence to be interrupted. Locked writes (hit or miss) and locked read misses affect the cache and cache directory just as their unlocked counterparts do. A locked read hit, however, is handled differently. The read is necessarily

forced to run on the 82385 local bus, and as the data returns from main memory, it is "re-copied" into the cache. (See Figure 5-3L.) The directory is not changed as it already indicates that this location exists in the cache. This activity is invisible to the system and ensures that semaphores are properly handled.

BLOCK# is asserted during locked 82385 bus cycles just as LOCK# is asserted during locked 386 DX cycles. The BLOCK# maximum valid delay, however, differs from that of LOCK#, and this must be accounted for in any circuitry that makes use of BLOCK#. The difference is due to the fact that LOCK#, unlike the other 386 DX cycle definition signals, is not pipelined. The situation is clarified in Figure 5-3K. In cycle 2 the state of LOCK# is not known before the corresponding system read starts (Frames 4 and 5). In this case, LOCK# is asserted at the beginning of T1P, and the delay for BLOCK# to become active is the delay of LOCK# from the 386 DX plus the propagation delay through the 82385. This occurs because T1P and the corresponding BT1P are concurrent (Frame 5). The result is that BLOCK# should not be sampled at the end of BT1P. The first appropriate sampling point is midway through the next state, as shown in Frame 6. In Figure 5-3L, the maximum delay for BLOCK# to become valid in Frame 4 is the same as the maximum delay for LOCK# to become valid from the 386 DX. This is true since the pipelining issue discussed above does not occur.

The 82385 should negate BLOCK# after BREADY# of the last 82385 Locked Cycle was asserted and Lock turns inactive. This means that in a sequence of cycles which begins with a 82385 Locked Cycle and goes on with all the possible Locked Cycles (other 82385 cycles, idles, and local cycles), while LOCK# is continuously active, the 82385 will maintain BLOCK# active continuously. Another implication is that in a Locked Posted Write Cycle followed by non-locked sequence, BLOCK# is negated one CLK after BREADY# of the write cycle. In other 82385 Locked Cycles, followed by non-locked sequences, BLOCK# is negated one CLK after LOCK# is negated, which occurs two CLKs after BREADY# is asserted. In the last case BLOCK# active moves by one CLK to the non-locked sequence.

The arbitration rules of Locked Cycles are:

MASTER MODE:

BHOLD input signal is ignored when BLOCK# or internal lock (16-bit non-aligned cycle) are active. BHLDA output signal remains inactive, and BAOE# output signal remains active at that time interval.

SLAVE MODE:

The 82385 does not relinquish the system bus if BLOCK# or internal lock are active. The BHOLD output signal remains active when BLOCK# or internal lock is active plus one CLK. The BHLDA input signal is ignored when BLOCK# or the internal lock is active plus one CLK. This means the 82385 slave does not respond to BHLDA inactivation. The BAOE# output signal remains active during the same time interval.

5.2.2 Additional 82385 Bus Signals

The 82385 bus provides two status outputs and one control input that are unique to cache operation and thus have no 386 DX counterparts. The outputs are MISS#, and WBS, and the input is FLUSH.

5.2.2.1 CACHE READ/WRITE MISS INDICATION (MISS#)

MISS# can be thought of as an extra 82385 bus cycle definition signal similar to BM/IO#, BW/R#, and BD/C#, that distinguishes cacheable read and write misses from other cycles. MISS#, like the other definition signals, becomes valid with BADS# (BT1 or first BT2P). The behavior of MISS# is illustrated in Figures 5-3B, 5-3C, and 5-3J. The 82385 floats MISS# when another master owns the bus, allowing multiple 82385s to share the same node in multi-cache systems. MISS# should thus be lightly pulled up (~ 20 KΩ) to keep it negated during hold (BTH) states.

MISS# can serve several purposes. As discussed previously, the BD0–BD31 and BREADY# setup times in a cache read miss are longer than in other cycles. A bus controller can distinguish these cycles by gating MISS# with BW/R#. MISS# may also prove useful in gathering 82385 system performance data.

5.2.2.2 WRITE BUFFER STATUS (WBS)

WBS is activated when 386 DX write cycle data is latched into the 74646 latching transceiver (via LDSTB). It is deactivated upon completion of the write cycle on the 82385 bus when the 82385 sees the BREADY# signal. WBS behavior is illustrated in Figures 5-3F through 5-3J, and potential applications are discussed in Chapter 3.

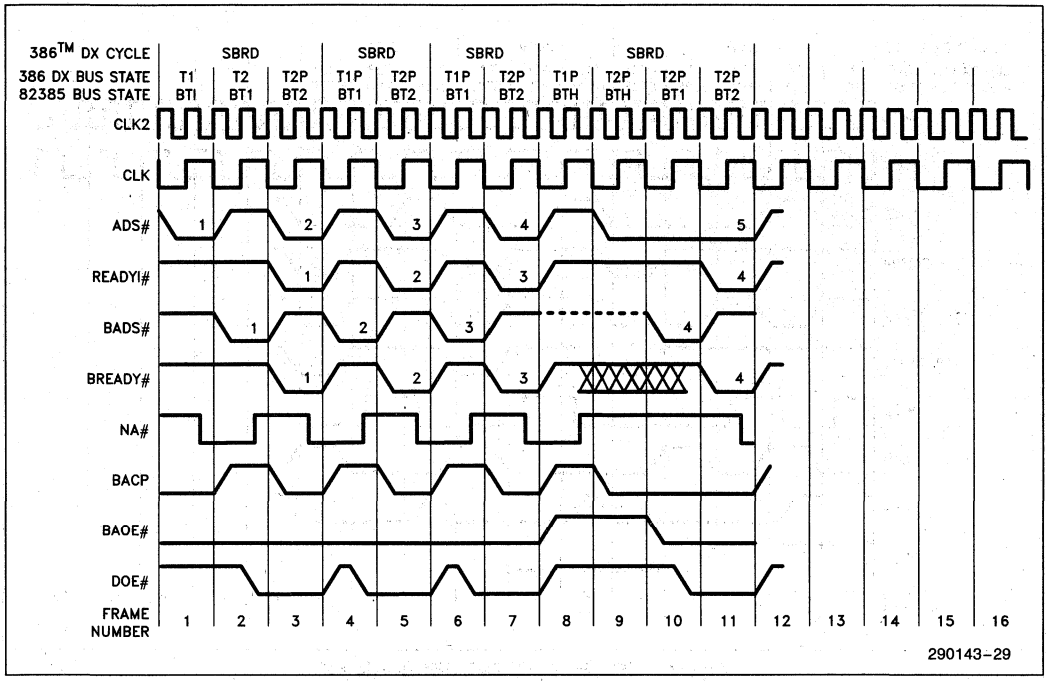


Figure 5-3A. Consecutive SBRD Cycles—(N = 0)

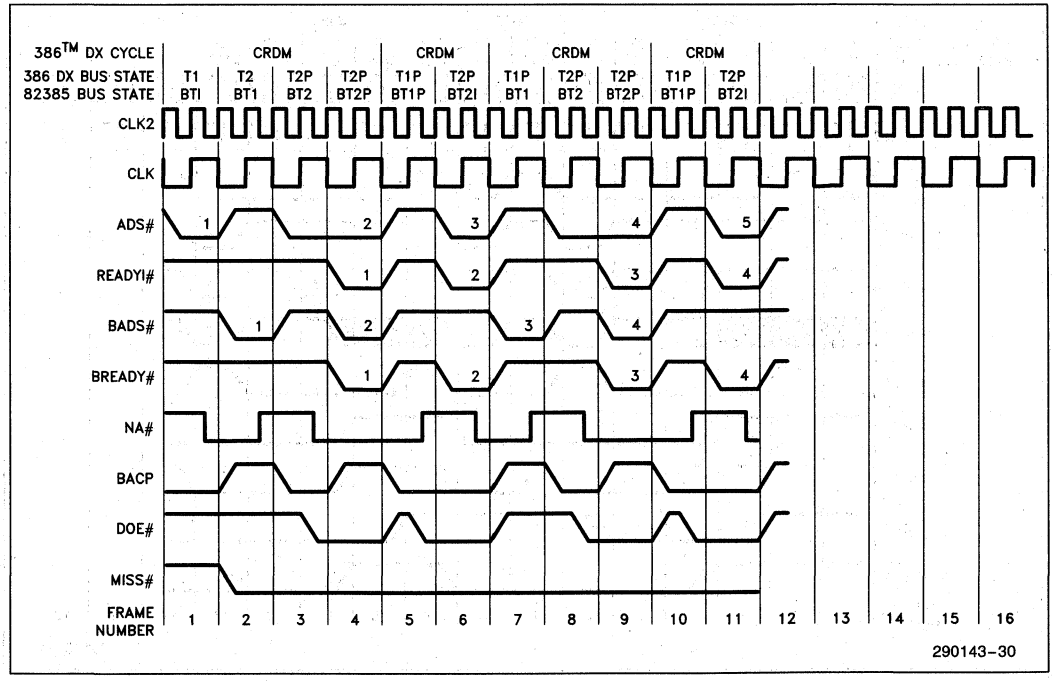


Figure 5-3B. Consecutive CRDM Cycles—(N = 1)

5

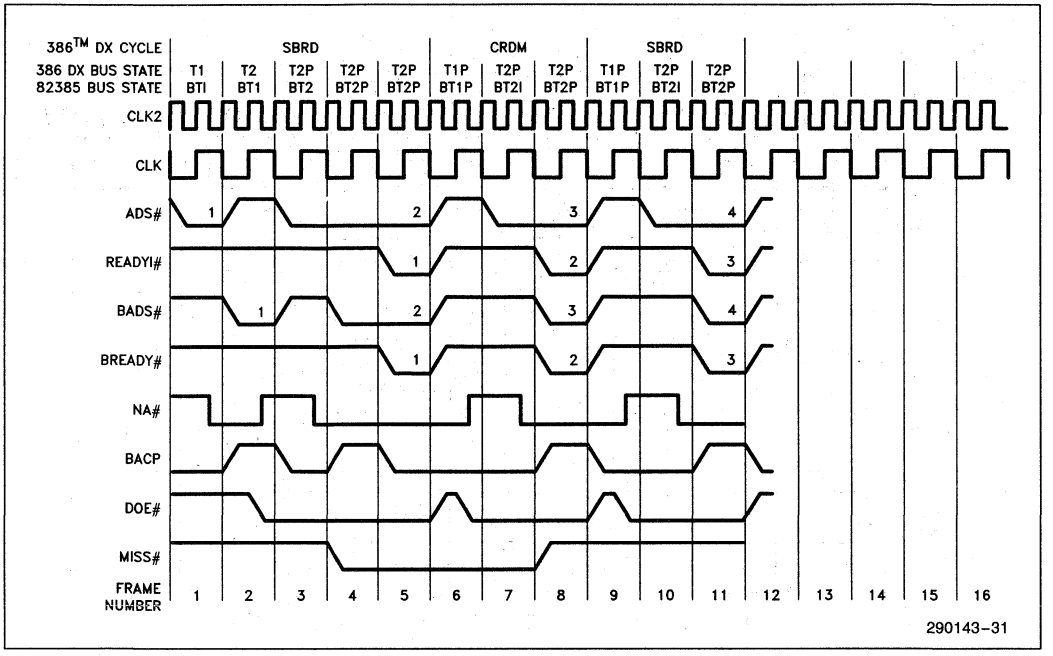


Figure 5-3C. SBRD, CRDM, SBRD—(N = 2)

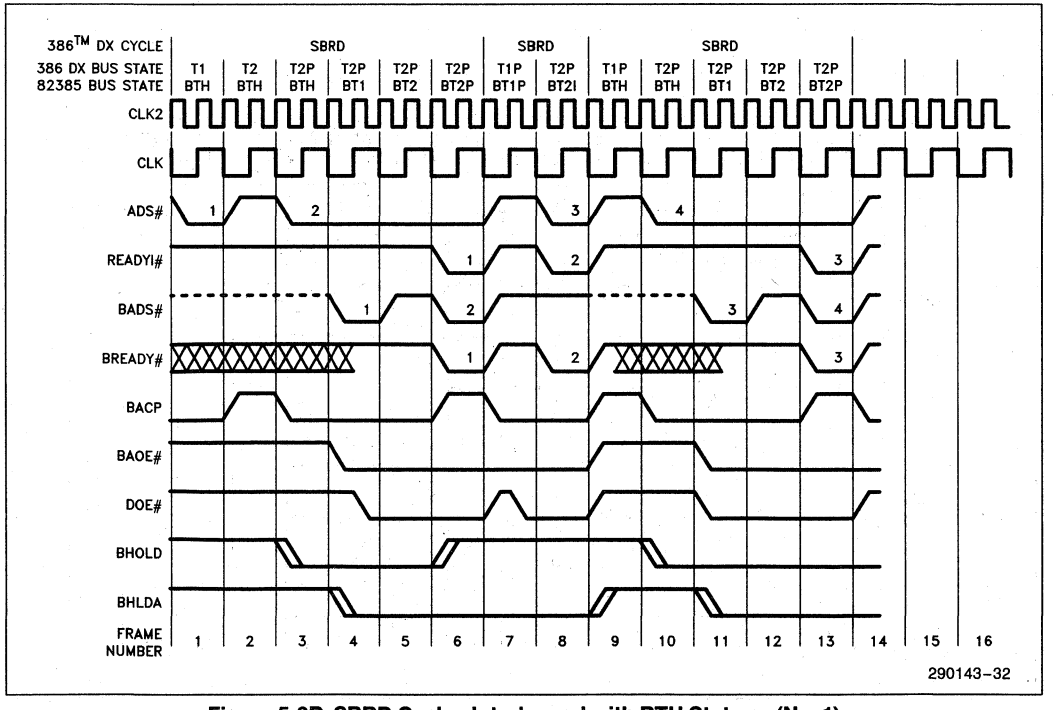


Figure 5-3D. SBRD Cycles Interleaved with BTH States—(N = 1)

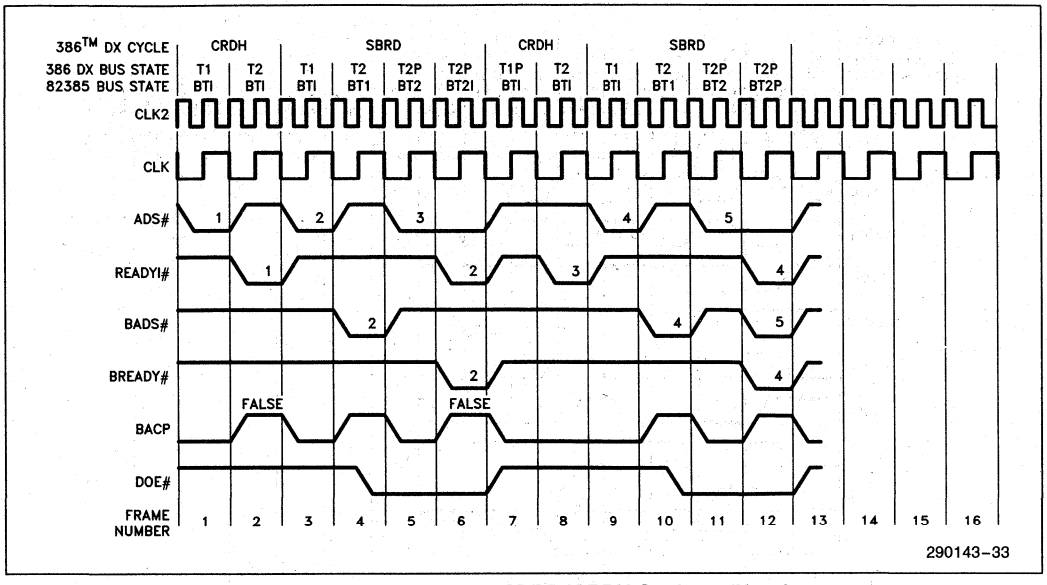


Figure 5-3E. Interleaved SBRD/CRDH Cycles—(N = 1)

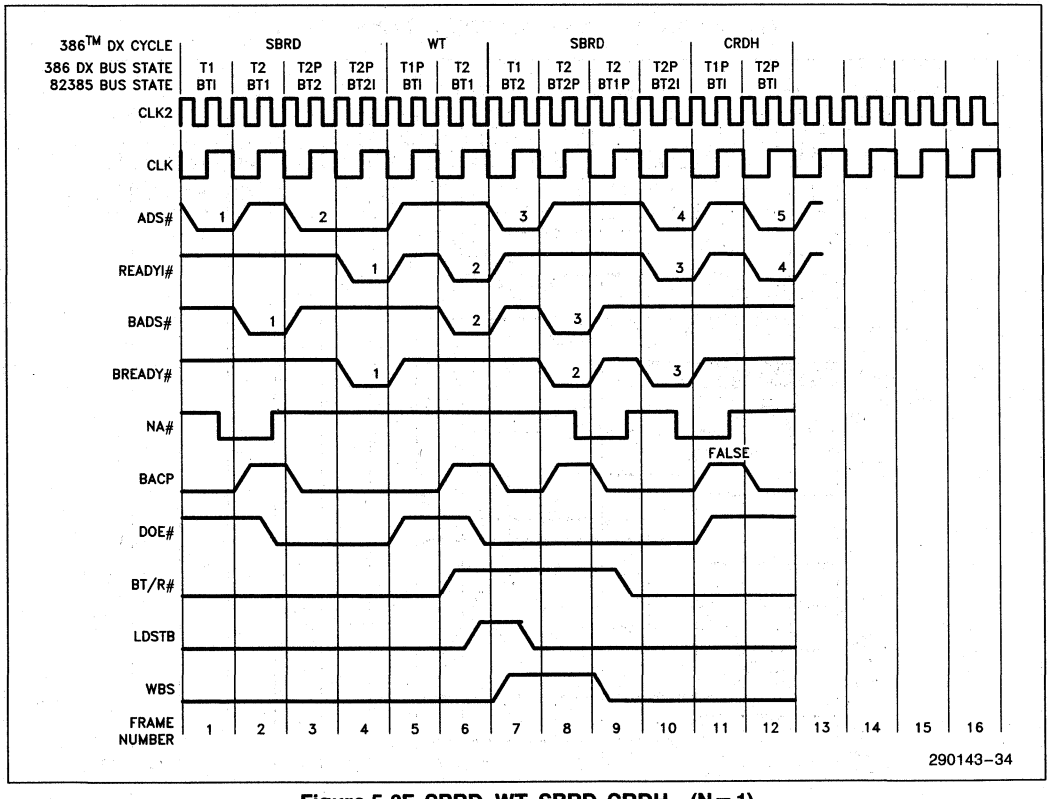


Figure 5-3F. SBRD, WT, SBRD, CRDH—(N = 1)

5

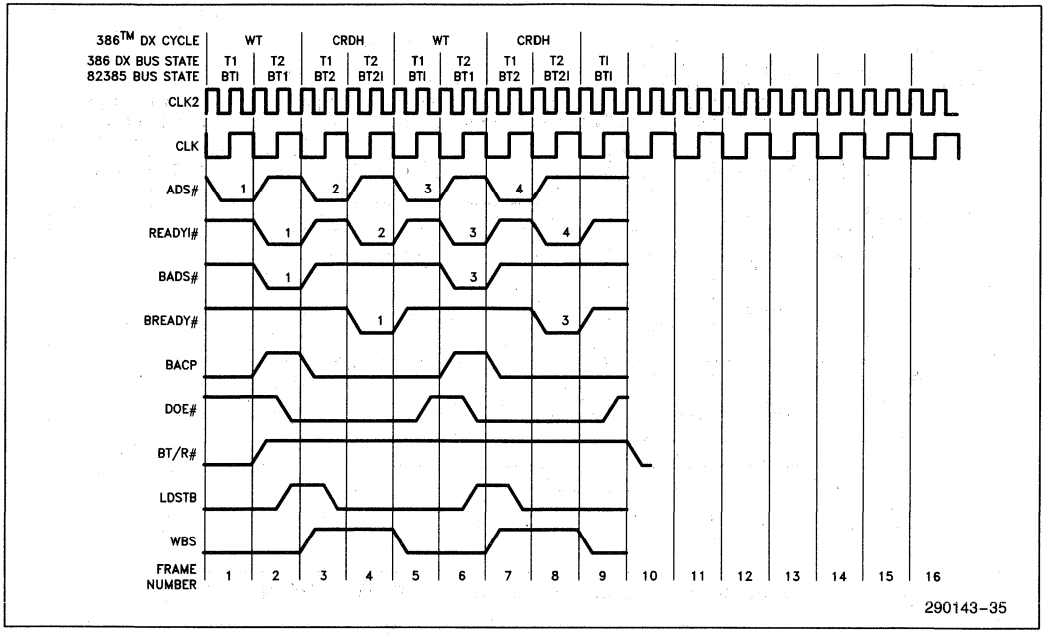


Figure 5-3G. Interleaved WT/CRDH Cycles—(N = 1)

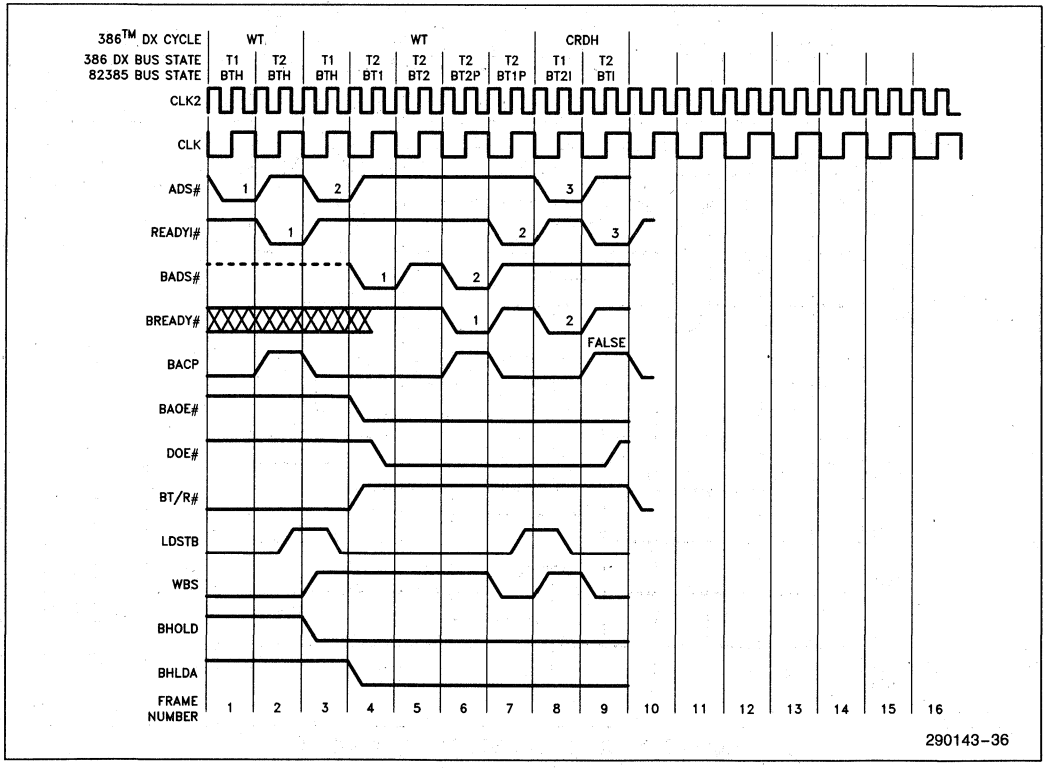


Figure 5-3H. WT, WT, CRDH—(N = 1)

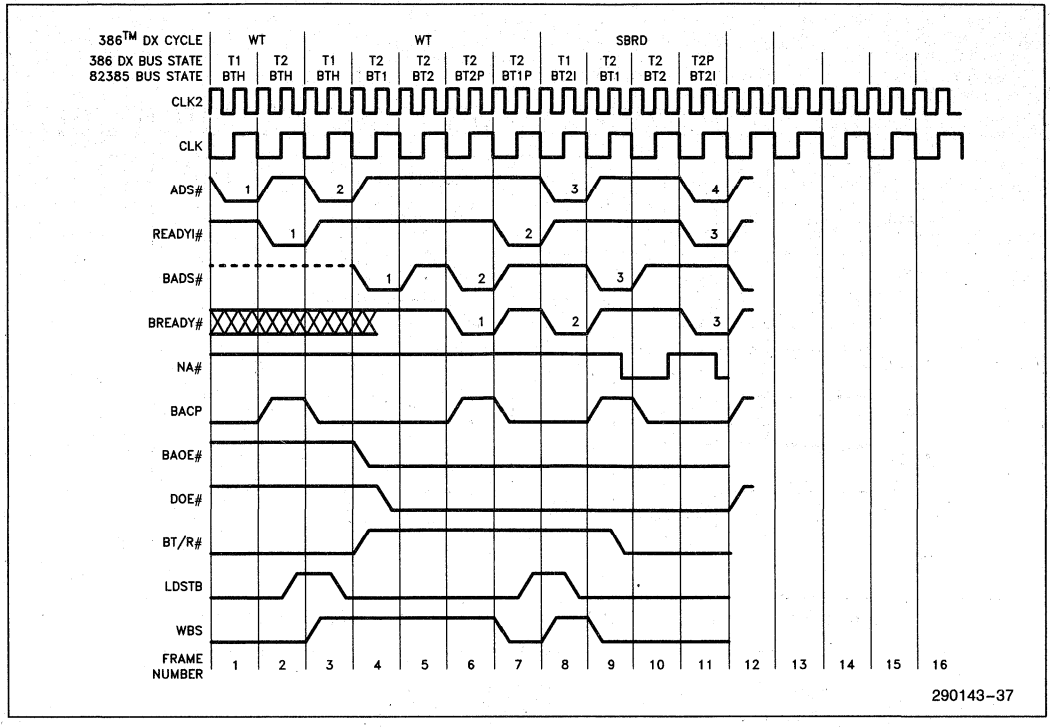


Figure 5-3I. WT, WT, SBRD—(N = 1)

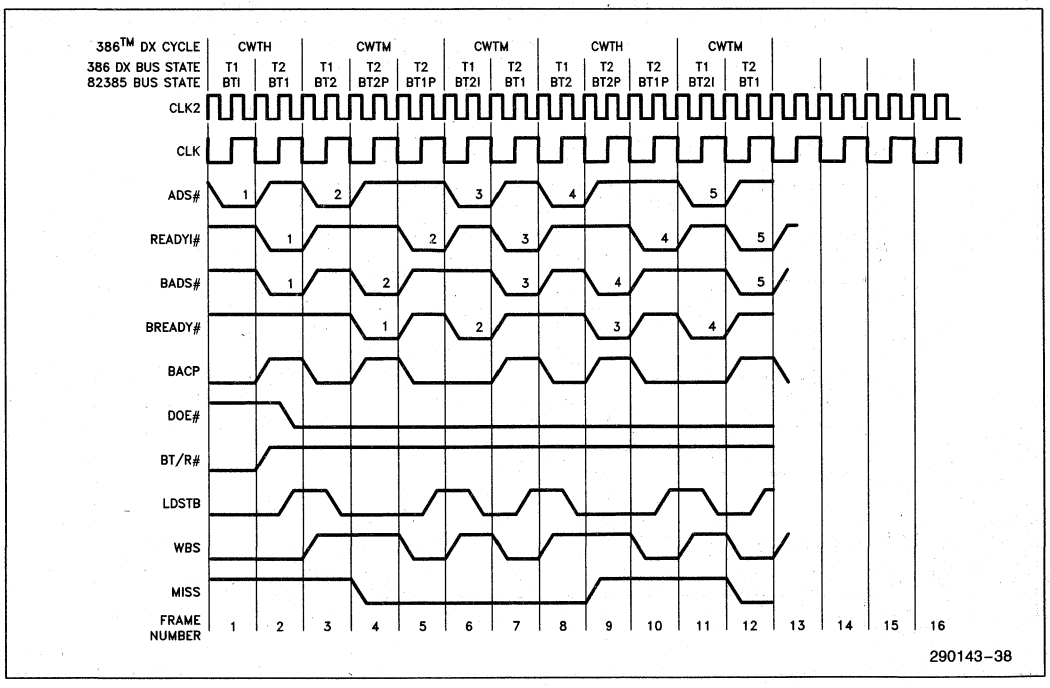


Figure 5-3J. Consecutive Write Cycles—(N = 1)

5

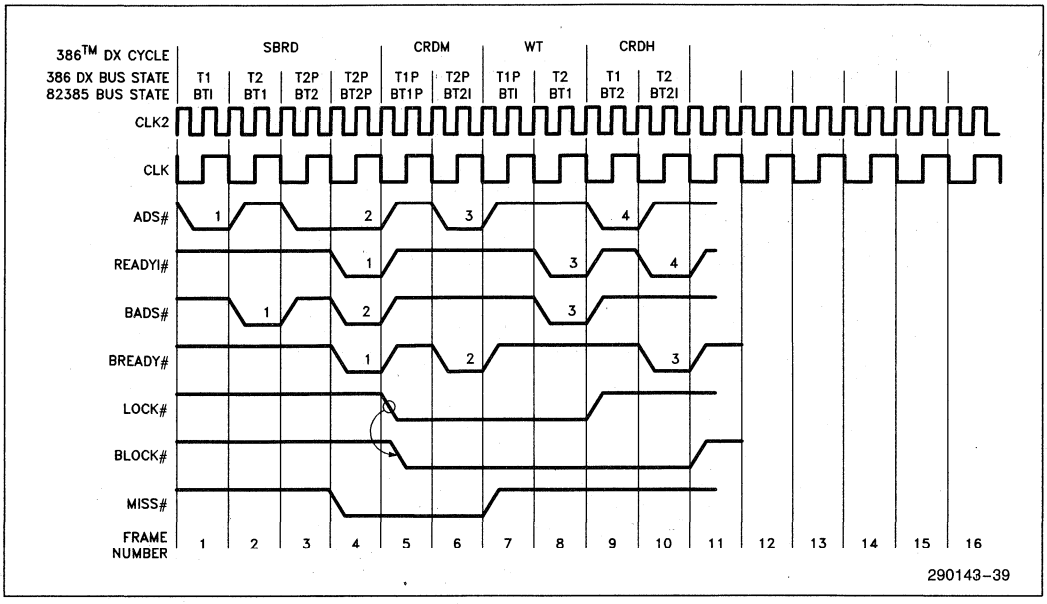


Figure 5-3K. LOCK# /BLOCK# in Non-Cacheable or Miss Cycles—(N = 1)

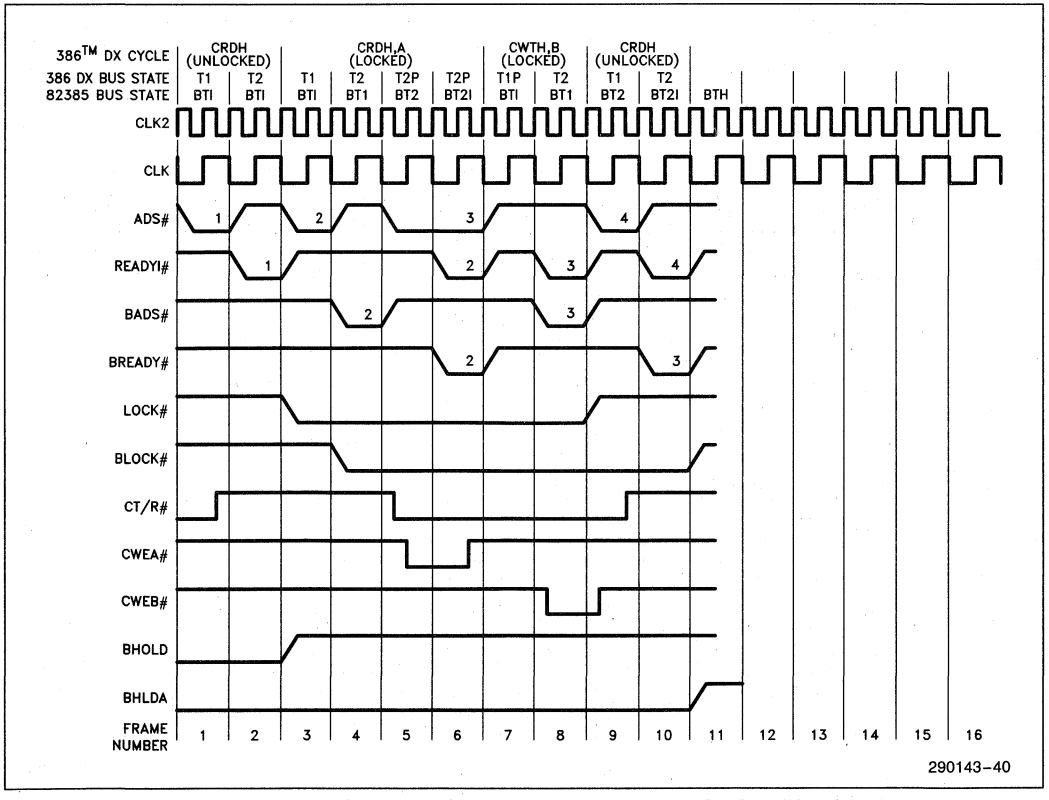


Figure 5-3L. LOCK# /BLOCK# in Cache Read Hit Cycle—(N = 1)

5.2.2.3 CACHE FLUSH (FLUSH)

FLUSH is an 82385 input which is used to reset all tag valid bits within the cache directory. The FLUSH input must be kept active for at least 4 CLK (8 CLK2) periods to complete the directory flush. Flush is generally used in diagnostics but can also be used in applications where snooping cannot guarantee coherency.

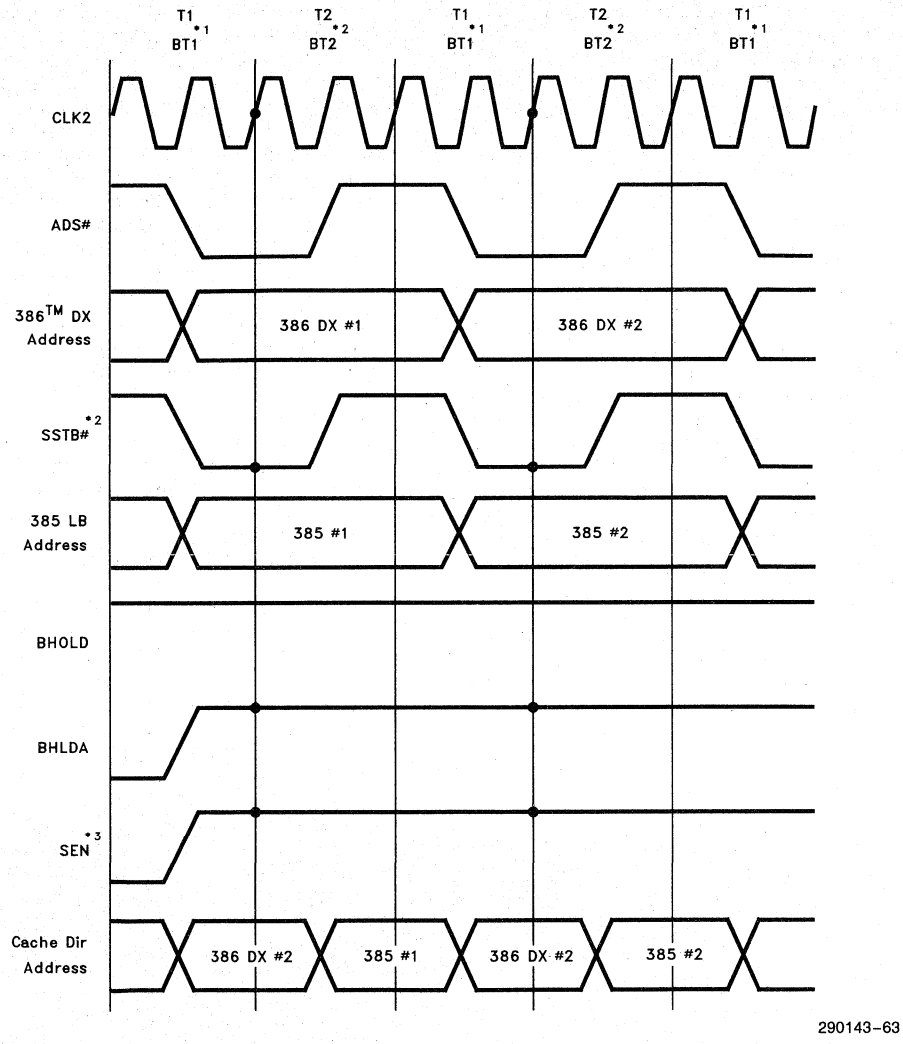
5.3 BUS WATCHING (SNOOP) INTERFACE

The 82385's bus watching interface consists of the snoop address (SA2–SA31), snoop strobe (SSTB#), and snoop enable (SEN) inputs. If masters reside at the system bus level, then the SA2–SA31 inputs are connected to the system address lines and SEN the system bus memory write command. SSTB# indicates that a valid address is present on the system bus. Note that the snoop bus inputs are synchronous, so care must be taken to ensure that they are stable during their sample windows. If no master resides beyond the 82385 bus level, then the 82385 inputs SA2–SA31, SEN, and SSTB# can respectively tie directly to BA2–BA31, BW/R#, and BADS# of the other system bus master (see Figure 5.5). However, it is recommended that SEN be driven by the logical "AND" of BW/R# and BM/IO# so as to prevent I/O writes from unnecessarily invalidating cache data.

When the 82385 detects a system write by another master and the conditions in Figure 5.4 are met: CLK2 PHI1 rising (CLK falling), BHLDA asserted, SEN asserted, SSTB# asserted, it internally latches SA2–SA31 and runs a cache look-up to see if the altered main memory location is duplicated in the cache. If yes (a snoop hit), the line valid bit associated with that cache entry is cleared. An important feature of the 82385 is that even if the 386 DX is running zero wait state hits out of the cache, all snoops are serviced. This is accomplished by time multiplexing the cache directory between the 386 DX address and the latched system address. If the SSTB# signal occurs during an 82385 comparison cycle (for the 386 DX), the 386 DX cycle has the highest priority in accessing the cache directory. This takes the first of the two 386 DX states. The other state is then used for the snoop comparison. This worst case example, depicted in Figure 5-4, shows the 386 DX running zero wait state hits on the 386 DX local bus, and another master running zero wait state writes on the 82385 bus. No snoops are missed, and no performance penalty incurred.

5.4 RESET DEFINITION

Table 5-1 summarizes the states of all 82385 outputs during reset and initialization. A slave mode 82385 tri-states its "386 DX-like" front end. A master mode 82385 emits a pulse stream on its BACP output. As the 386 DX address and cycle definition lines reach their reset values, this stream will latch the reset values through to the 82385 bus.



NOTES:
*1. These states are induced by another System Bus master.
*2. SSTB# on the 82385 is tied directly to BADS# of the System Bus master.
*3. SEN on the 82385 is tied directly to BW/R# of the System Bus master.

Figure 5.4. Interleaved Snoop and 386 DX Accesses to the Cache Directory

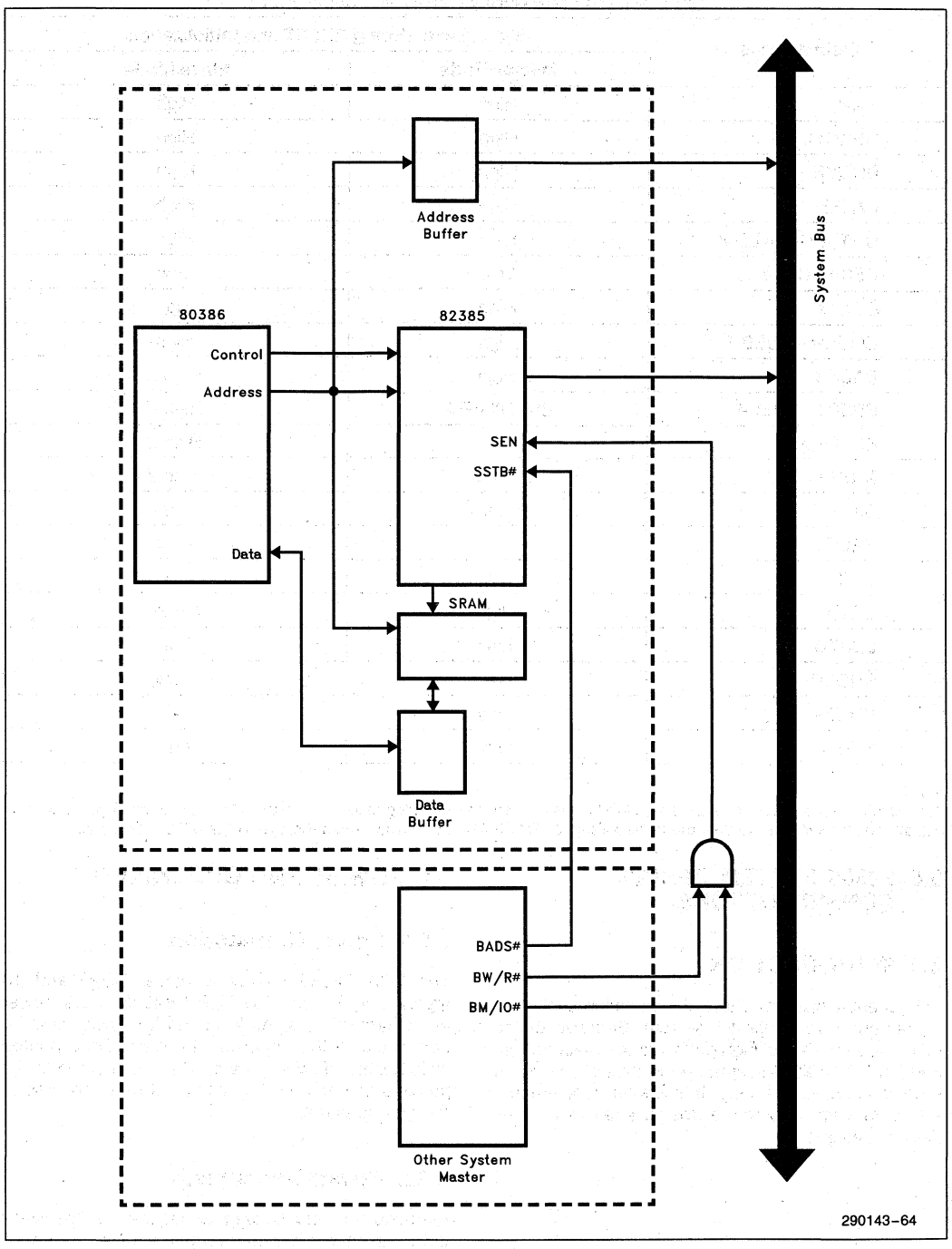


Figure 5.5. Snooping Connections in a Multi Master Environment

Table 5-1. Pin State During RESET and Initialization

Output Name	Signal Level During RESET and Initialization	
	Master Mode	Slave Mode
NA#	High	High
READY0#	High	High
BRDYEN#	High	High
CALEN	High	High
CWEA# – CWEB#	High	High
CS0# – CS3#	Low	Low
CT/R#	High	High
COEA# – COEB#	High	High
BADS#	High	High Z
BBE0# – BBE3#	386 DX BE#	High Z
BLOCK#	High	High Z
MISS#	High	High Z
BACP	Pulse(1)	Pulse
BAOE#	Low	High
BT/R#	Low	Low
DOE#	High	High
LDSTB	Low	Low
BHOLD	—	Low
BHLDA	Low	—
WBS	Low	Low

NOTE:
1. In Master Mode, BAOE# is low and BACP emits a pulse stream during reset. As the 386 DX address and cycle definition signals reach their reset values, the pulse stream on BACP will latch these values through to the 82385 local bus.

6.0 82385 SYSTEM DESIGN CONSIDERATIONS

6.1 INTRODUCTION

This chapter discusses techniques which should be implemented in an 82385 system. Because of the high frequencies and high performance nature of the 386 DX CPU/82385 system, good design and layout techniques are necessary. It is always recommended to perform a complete design analysis on new system designs.

6.2 POWER AND GROUNDING

6.2.1 Power Connections

The PGA 82385 utilizes 8 power (V_{CC}) and 10 ground (V_{SS}) pins. The PQFP 82385 has 9 power and 9 ground pins. All V_{CC} and V_{SS} pins must be connected to their appropriate plane. On a printed circuit board, all V_{CC} pins must be connected to the power plane and all V_{SS} pins must be connected to the ground plane.

6.2.2 Power Decoupling

Although the 82385 itself is generally a "passive" device in that it has few output signals, the cache

subsystem as a whole is quite active. Therefore, many decoupling capacitors should be placed around the 82385 cache subsystem.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the decoupling capacitors and their respective devices as much as possible. Capacitors specifically for PGA packages are also commercially available, for the lowest possible inductance.

6.2.3 Resistor Recommendations

Because of the dual bus structure of the 82385 subsystem (386 DX Local Bus and 82385 Local Bus), any signals which are recommended to be pulled up will be respective to one of the busses. The following sections will discuss signals for both busses.

6.2.3.1 386 DX LOCAL BUS

For typical designs, the pullup resistors shown in Table 6-1 are recommended. This table correlates to Chapter 7 of the 386 DX Data Sheet. However, particular designs may have a need to differ from the listed values. Design analysis is recommended to determine specific requirements.

6.2.3.2 82385 LOCAL BUS

Pullup resistor recommendations for the 82385 Local Bus signals are shown in Table 6-2. Design analysis is necessary to determine if deviations to the typical values given is needed.

Table 6-1. Recommended Resistor Pullups to V_{CC} (386 DX Local Bus)

Pin and Signal	Pullup Value	Purpose
ADS# PGA E13 PQFP 123	20 KΩ ±10%	Lightly Pull ADS# Negated for 386 DX Hold States
LOCK# PGA F13 PQFP 118	20 KΩ ±10%	Lightly Pull LOCK# Negated for 386 DX Hold States

Table 6-2. Recommended Resistor Pullups to V_{CC} (82385 Local Bus)

Signal and Pin	Pullup Value	Purpose
BADS# PGA N9 PQFP 89	20 KΩ ±10%	Lightly Pull BADS# Negated for 82385 Hold States
BLOCK# PGA P9 PQFP 86	20 KΩ ±10%	Lightly Pull BLOCK# Negated for 82385 Hold States
MISS# PGA N8 PQFP 85	20 KΩ ±10%	Lightly Pull MISS# Negated for 82385 Hold States

6.3 82385 SIGNAL CONNECTIONS

6.3.1 Configuration Inputs

The 82385 configuration signals (M/S#, 2W/D#, DEFOE#) must be connected (pulled up) to the appropriate logic level for the system design. There is also a reserved 82385 input which must be tied to the appropriate level. Refer to Table 6-3 for the signals and their required logic level.

Table 6-3. 82385 Configuration Inputs Logic Levels

Pin and Signal	Logic Level	Purpose
M/S# PGA B13 PQFP 129	High	Master Mode Operation
	Low	Slave Mode Operation
2W/D# PGA D12 PQFP 127	High	2-Way Set Associative
	Low	Direct Mapped
Resrved PGA L14 PQFP 102	High	Must be tied to V_{CC} via a pull-up for proper functionality
DEFOE# PGA A14 PQFP 128	N/A	Define Cache Output Enables. Allows use of any SRAM.

NOTE:
The listed 82385 pins which need to be tied high should use a pull-up resistor in the range of 5 KΩ to 20 KΩ.

6.3.2 CLK2 and RESET

The 82385 has two inputs to which the 386 DX CLK2 signal must be connected. One is labeled CLK2 (82385 PGA pin C13, PQFP lead 126) and the other is labeled BCLK2 (82385 PGA pin L13, PQFP lead 103). These two inputs must be tied together on the printed circuit board.

The 82385 also has two reset inputs. RESET (82385 PGA pin D13, PQFP lead 125) and BRESET (82385 PGA pin K12, PQFP lead 104) must be connected on the printed circuit board.

6.4 UNUSED PIN REQUIREMENTS

For reliable operation, ALWAYS connect unused inputs to a valid logic level. As is the case with most other CMOS processes, a floating input will increase the current consumption of the component and give an indeterminate state to the component.

6.5 CACHE SRAM REQUIREMENTS

The 82385 offers the option of using SRAMs with or without an output enable pin. This is possible by inserting a transceiver between the SRAMs and the 386 DX local data bus and strapping DEFOE# to the appropriate logic level for a given system configuration. This transceiver may also be desirable in a system which has a very heavily loaded 386 DX local data bus. The following sections discuss the SRAM requirements for all cache configurations.

6.5.1 Cache Memory without Transceivers

As discussed in Section 3.2, the 82385 presents all of the control signals necessary to access the cache memory. The SRAM chip selects, write enables, and output enables are driven directly by the 82385. Table 6-4 lists the required SRAM specifications. These specifications allow for zero margins. They should be used as guides for the actual system design.

6.5.2 Cache Memory With Transceivers

To implement an 82385 subsystem using cache memory transceivers, COEA# or COEB# must be used as output enable signals for the transceivers and DEFOE# must be appropriately strapped for proper COEx# functionality (since the cache SRAM transceivers must be enabled for writes as well as reads). DEFOE# must be tied high when using cache SRAM transceivers. In a 2-way set associative organization, COEA# enables the transceiver for bank A and COEB# enables the bank B transceiver. A direct mapped cache may use either COEA# or COEB# to enable the transceiver. Table 6-5 lists the required SRAM specifications. These specifications allow for zero margin. They should be used as guides for the actual system design.

Table 6-4. SRAM Specs for Non-Buffered Cache Memory

SRAM Spec Requirements						
	Direct Mapped			2-Way Set Associative		
	20	25	33	20	25	33
Read Cycle Requirements						
Address Access (MAX)	44	36	27	42	34	27
Chip Select Access (MAX)	56	44	35	56	41	35
OE# to Data Valid (MAX)	19	13	10	14	13	10
OE# to Data Float (MAX)	20	15	10	20	15	10
Write Cycle Requirements						
Chip Select to End of Write (MIN)	30	25	20	30	25	20
Address Valid to End of Write (MIN)	42	37	29	40	37	29
Write Pulse Width (MIN)	30	25	20	30	25	20
Data Setup (MAX)	—	—	—	—	—	—
Data Hold (MIN)	4	4	2	4	4	2

Table 6-5. SRAM Specs for Buffered Cache Memory

SRAM Spec Requirements						
	Direct Mapped			2-Way Set Associative		
	20	25	33	20	25	33
Read Cycle Requirements						
Address Access (MAX)	37	29	20	35	29	20
Chip Select Access (MAX)	48	36	27	48	36	27
OE# to Data Valid (MAX)	N/A	N/A	N/A	N/A	N/A	N/A
OE# to Data Float (MAX)	N/A	N/A	N/A	N/A	N/A	N/A
Write Cycle Requirements						
Chip Select to End of Write (MIN)	30	25	20	30	23	20
Address Valid to End of Write (MIN)	42	37	29	40	36	27
Write Pulse Width (MIN)	30	25	20	30	25	20
Data Setup (MAX)	15	10	10	15	10	10
Data Hold (MIN)	3	3	3	3	3	3

7.0 SYSTEM TEST CONSIDERATIONS

7.1 INTRODUCTION

Power On Self Testing (POST) is performed by most systems after a reset. This chapter discusses the requirements for properly testing an 82385 based system after power up.

7.2 MAIN MEMORY (DRAM) TESTING

Most systems perform a memory test by writing a data pattern and then reading and comparing the data. This test may also be used to determine the total available memory within the system. Without properly taking into account the 82385 cache memory, the memory test can give erroneous results. This will occur if the cache responds with read hits during the memory test routine.

7.2.1 Memory Testing Routine

In order to properly test main memory, the test routine must not read from the same block consecutively. For instance, if the test routine writes a data pattern to the first 32 kbytes of memory (0000–7FFFH), reads from the same block, writes a new pattern to the same locations (0000–7FFFH), and reads the new pattern, the second pattern tested would have had data returned from the 82385 cache memory. Therefore, it is recommended that the test routine work with a memory block of at least 64 kbytes. This will guarantee that no 32 kbyte block will be read twice consecutively.

7.3 82385 CACHE MEMORY TESTING

With the addition of SRAMs for the cache memory, it may be desirable for the system to be able to test the cache SRAMs during system diagnostics. This requires the test routine to access only the cache memory. The requirements for this routine are based on where it resides within the memory map. This can be broken into two areas: the routine residing in cacheable memory space or the routine residing in either non-cacheable memory or on the 386 DX local bus (using the LBA# input).

5

7.3.1 Test Routine in the NCA# or LBA# Memory Map

In this configuration, the test routine will never be cached. The recommended method is code which will access a single 32 kbyte block during the test. Initially, a 32 kbyte read (assume 0000–7FFFH) must be executed. This will fill the cache directory with the address information which will be used in the diagnostic procedure. Then, a 32 kbyte write to the same address locations (0000–7FFFH) will load the cache with the desired test pattern (due to write hits). The comparison can be made by completing another 32 kbyte read (same locations, 0000–7FFFH), which will be cache read hits. Subsequent writes and reads to the same addresses will enable various patterns to be tested.

7.3.2 Test Routine in Cacheable Memory

In this case, it must be understood that the diagnostic routine must reside in the cache memory before the actual data testing can begin. Otherwise, when the 386 DX performs a code fetch, a location within the cache memory which is to be tested will be altered due to the read miss (code fetch) update.

The first task is to load the diagnostic routine into the top of the cache memory. It must be known how much memory is required for the code as the rest of the cache memory will be tested as in the earlier method. Once the diagnostics have been cached (via read updates), the code will perform the same type of read/write/read/compare as in the routine explained in the above section. The difference is that now the amount of cache memory to be tested is 32 kbytes minus the length of the test routine.

7.4 82385 CACHE DIRECTORY TESTING

Since the 82385 does not directly access the data bus, it is not possible to easily complete a comparison of the cache directory. (The 82385 can serially transmit its directory contents. See Section 7.5.) However, the cache memory tests described in Section 7.3 will indicate if the directory is working properly. Otherwise, the data comparison within the diagnostics will show locations which fail.

There is a slight possibility that the cache memory comparison could pass even if locations within the directory gave false hit/miss results. This could cause the comparison to always be performed to main memory instead of the cache and give a proper comparison to the 386 DX . The solution here is to use the MISS# output of the 82385 as an indicator to a diagnostic port which can be read by the 386 DX . It could also be used to flag an interrupt if a failure occurs.

The implementation of these techniques in the diagnostics will assure proper functionality of the 82385 subsystem.

7.5 SPECIAL FUNCTION PINS

As mentioned in Chapter 3, there are three 82385 pins which have reserved functions in addition to their normal operational functions. These pins are MISS#, WBS, and FLUSH.

As discussed previously, the 82385 performs a directory flush when the FLUSH input is held active for at least 4 CLK (8 CLK2) cycles. However, the FLUSH pin also serves as a diagnostic input to the 82385. The 82385 will enter a reserved mode if the FLUSH pin is high at the falling edge of RESET.

If, during normal operation, the FLUSH input is active for only one CLK (2 CLK2) cycle/s, the 82385 will enter another reserved mode. Therefore it must be guaranteed that FLUSH is active for at least the 4 CLK (8 CLK2) cycle specification.

WBS and MISS# serve as outputs in the 82385 reserved modes.

8.0 MECHANICAL DATA

8.1 INTRODUCTION

This chapter discusses the physical package and its connections in detail.

8.2 PIN ASSIGNMENT

The 82385 pinout as viewed from the top side of the component is shown by Figure 8-1. Its pinout as viewed from the Pin side of the component is shown in Figure 8-2.

V_{CC} and V_{SS} connections must be made to multiple V_{CC} and V_{SS} (GND) pins. Each V_{CC} and V_{SS} must be connected to the appropriate voltage level. The circuit board should include V_{CC} and GND planes for power distribution and all V_{CC} and V_{SS} pins must be connected to the appropriate plane.

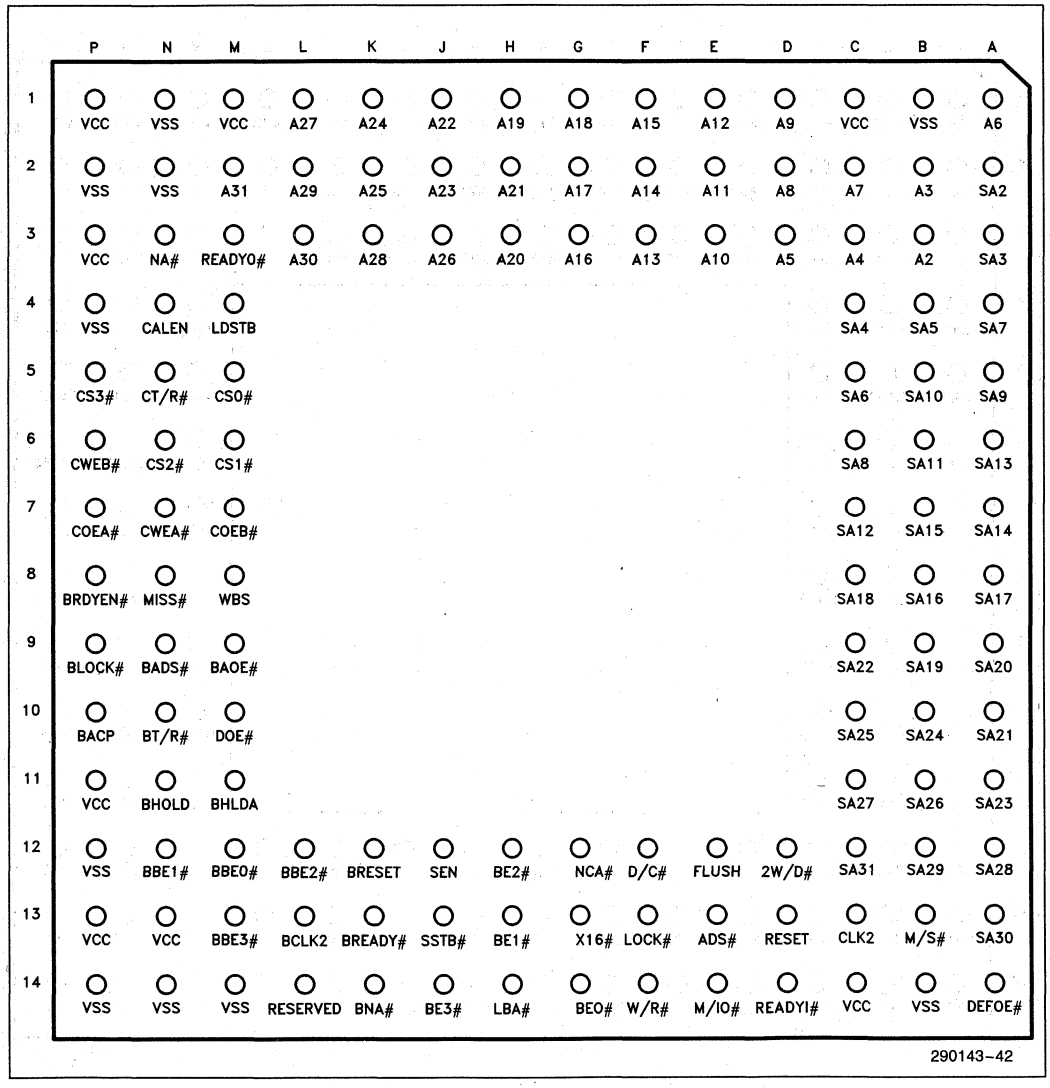


Figure 8-1. 82385 PGA Pinout—View from TOP Side

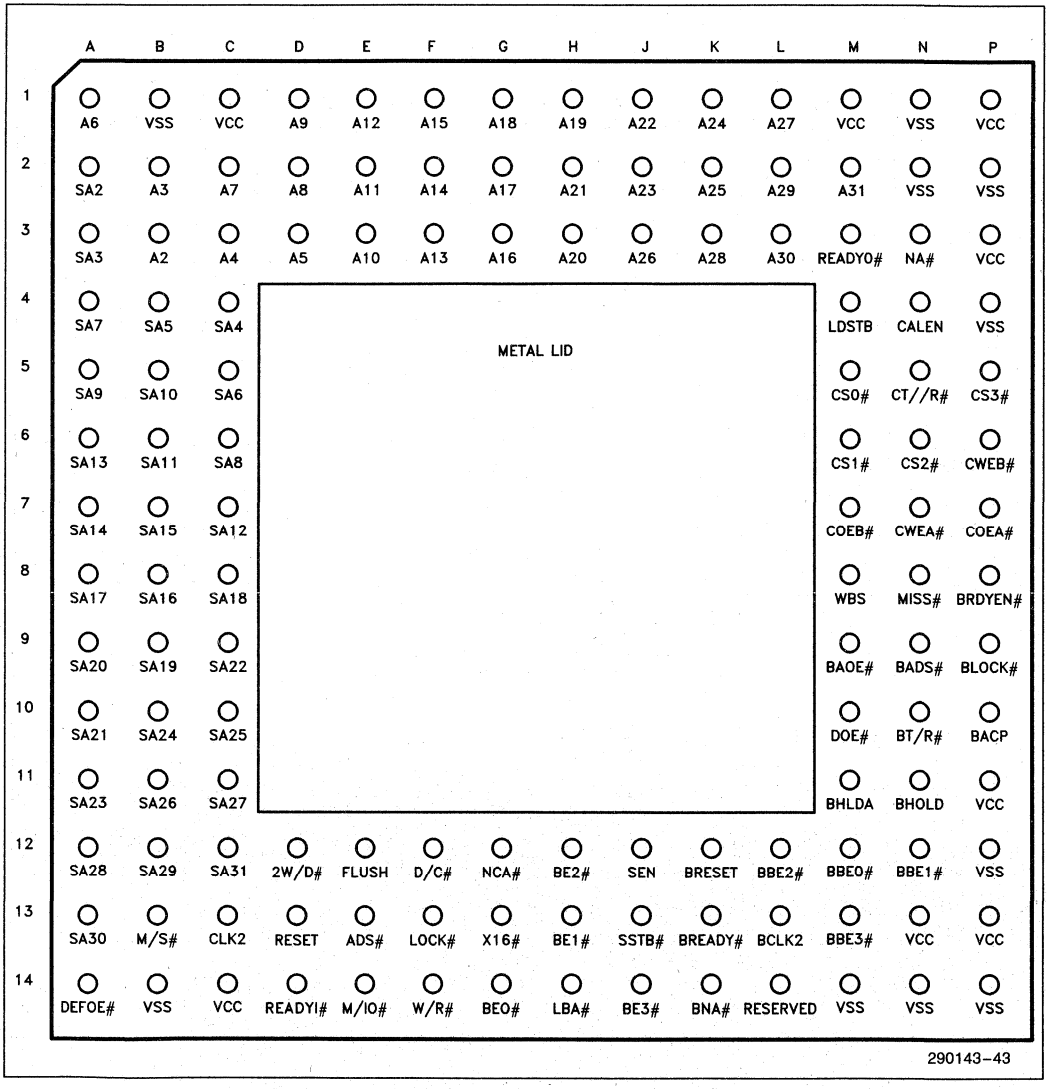


Figure 8-2. 82385 PGA Pinout—View from PIN Side

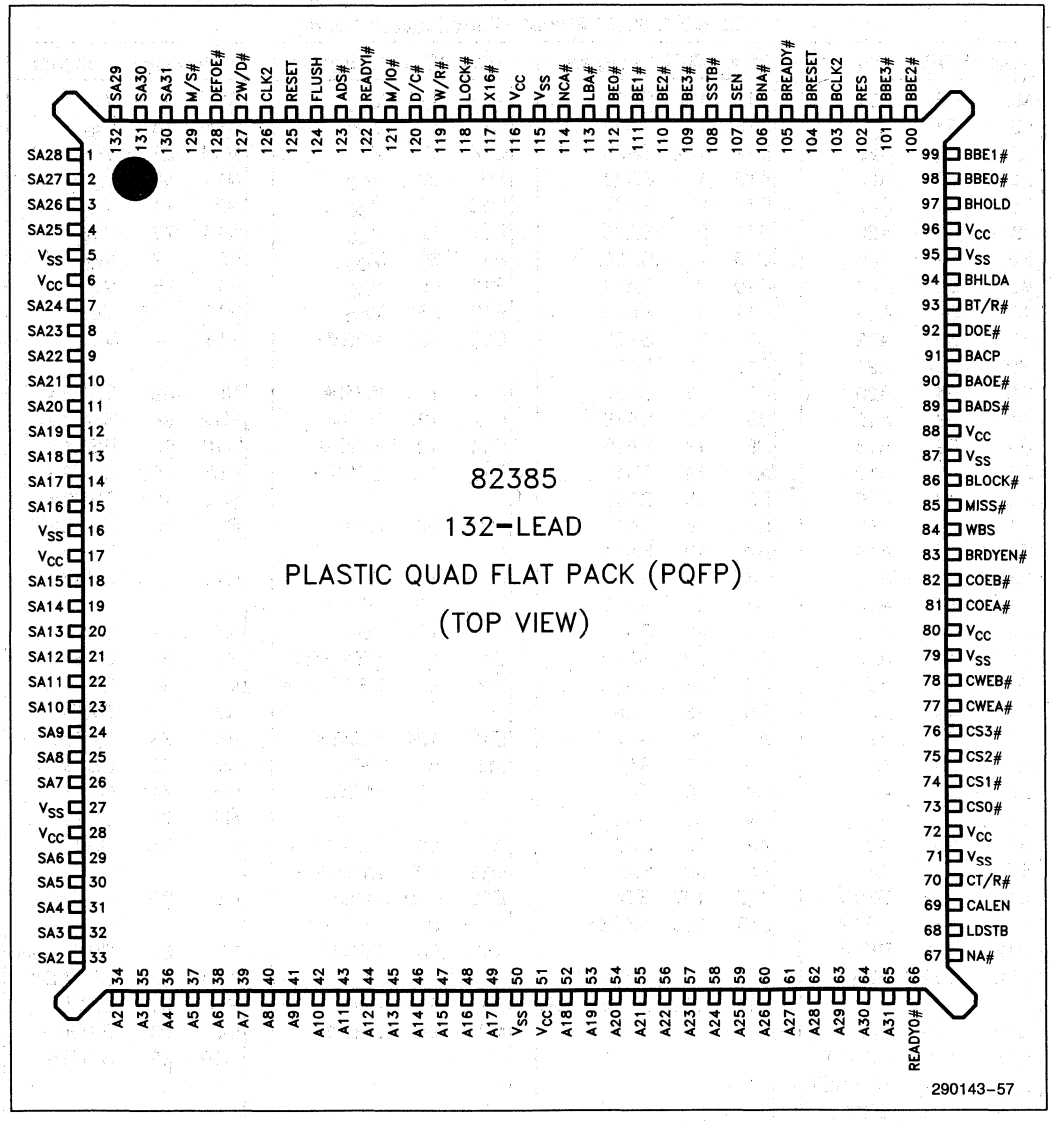


Figure 8-3. 82385 PQFP Pinout—View from TOP Side

Table 8-1. 82385 Pinout—Functional Grouping

PGA	PQFP	Signal
M2	65	A31
L3	64	A30
L2	63	A29
K3	62	A28
L1	61	A27
J3	60	A26
K2	59	A25
K1	58	A24
J2	57	A23
J1	56	A22
H2	55	A21
H3	54	A20
H1	53	A19
G1	52	A18
G2	49	A17
G3	48	A16
F1	47	A15
F2	46	A14
F3	45	A13
E1	44	A12
E2	43	A11
E3	42	A10
D1	41	A9
D2	40	A8
C2	39	A7
A1	38	A6
D3	37	A5
C3	36	A4
B2	35	A3
B3	34	A2
G14	112	BE0#
H13	111	BE1#
H12	110	BE2#
J14	109	BE3#
C13	126	CLK2
D13	125	RESET
K12	104	BRESET
L13	103	BCLK2

PGA	PQFP	Signal
C12	130	SA31
A13	131	SA30
B12	132	SA29
A12	1	SA28
C11	2	SA27
B11	3	SA26
C10	4	SA25
B10	7	SA24
A11	8	SA23
C9	9	SA22
A10	10	SA21
A9	11	SA20
B9	12	SA19
C8	13	SA18
A8	14	SA17
B8	15	SA16
B7	18	SA15
A7	19	SA14
A6	20	SA13
C7	21	SA12
B6	22	SA11
B5	23	SA10
A5	24	SA9
C6	25	SA8
A4	26	SA7
C5	29	SA6
B4	30	SA5
C4	31	SA4
A3	32	SA3
A2	33	SA2
J12	107	SEN
J13	108	SSTB#
L14	102	RESERVED

PGA	PQFP	Signal
—	116	V_{CC}
C1	6	V_{CC}
C14	17	V_{CC}
M1	28	V_{CC}
N13	51	V_{CC}
P1	72	V_{CC}
P3	80	V_{CC}
P11	88	V_{CC}
P13	96	V_{CC}
E13	123	ADS#
F14	119	W/R#
F12	120	D/C#
E14	121	M/IO#
F13	118	LOCK#
N3	67	NA#
G13	117	X16#
G12	114	NCA#
H14	113	LBA#
D14	122	READYI#
M3	66	READYO#
E12	124	FLUSH
M8	84	WBS
N8	85	MISS#
A14	128	DEFOE#
D12	127	2W/D#
B13	129	M/S#
M10	92	DOE#
M4	68	LDSTB
N11	97	BHOLD
M11	94	BHLDA

PGA	PQFP	Signal
B1	5	V_{SS}
B14	16	V_{SS}
M14	27	V_{SS}
N1	50	V_{SS}
N2	71	V_{SS}
N14	79	V_{SS}
P2	87	V_{SS}
P4	95	V_{SS}
P12	115	V_{SS}
P14	—	V_{SS}
N9	89	BADS#
M12	98	BBE0#
N12	99	BBE1#
L12	100	BBE2#
M13	101	BBE3#
P9	86	BLOCK#
K14	106	BNA#
N4	69	CALEN
P7	81	COEA#
M7	82	COEB#
N7	77	CWEA#
P6	78	CWEB#
M5	73	CS0#
M6	74	CS1#
N6	75	CS2#
P5	76	CS3#
N5	70	CT/R#
P8	83	BRDYEN#
K13	105	BREADY#
P10	91	BACP
M9	90	BAOE#
N10	93	BT/R#

8.3 PACKAGE DIMENSIONS AND MOUNTING

The 82385 package is a 132-pin ceramic Pin Grid Array (PGA). The pins are arranged 0.100 inch (2.5 mm) center-to-center, in a 14 x 14 matrix, three rows around (Figure 8-3).

A wide variety of available sockets allow low insertion force or zero insertion force mounting. These come in a choice of terminals such as soldertail, surface mount, or wire wrap.

8.4 PACKAGE THERMAL SPECIFICATION

The PGA case temperature should be measured at the center of the top surface opposite the pins, as in Figure 8-4. The case temperature may be measured in any environment to determine whether or not the 82385 is within the specified operating range.

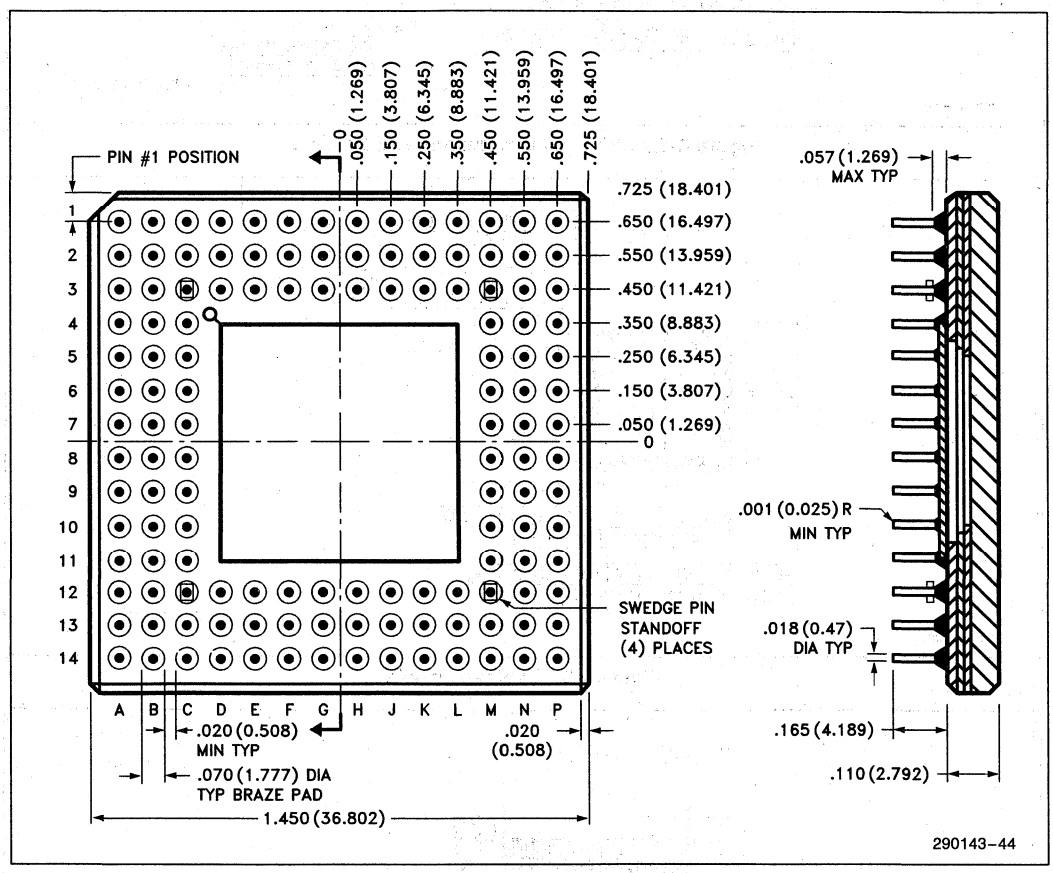


Figure 8-3.1. 132-Pin PGA Package Dimensions

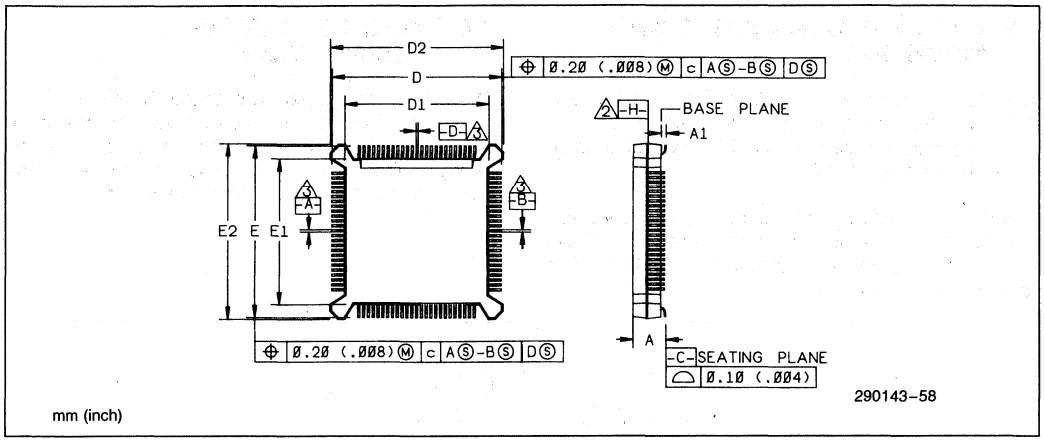


Figure 8-3.2. Principal Dimensions and Datums

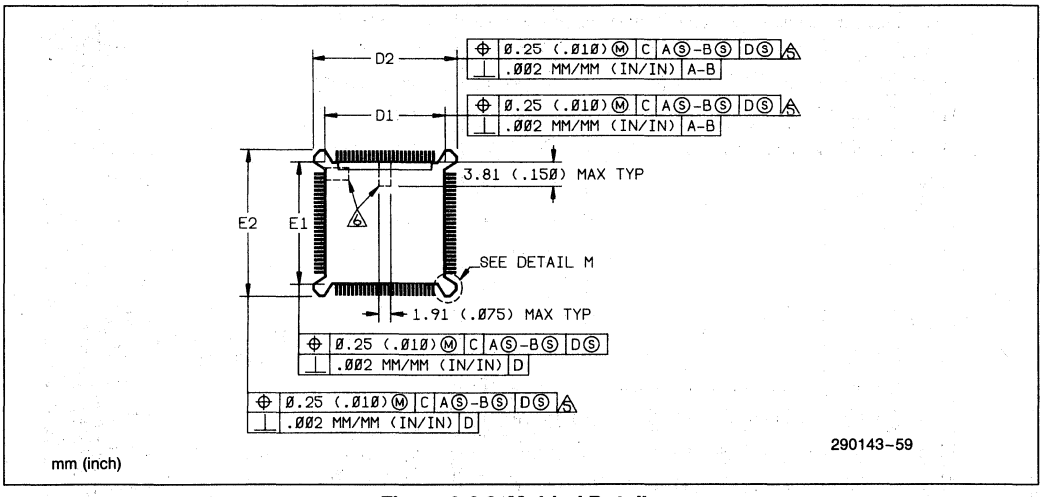


Figure 8-3.3. Molded Details

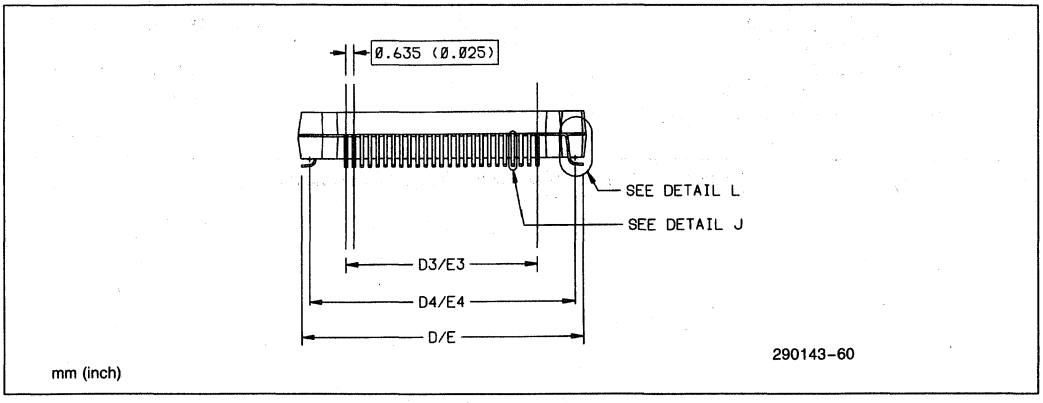


Figure 8-3.4. Terminal Details

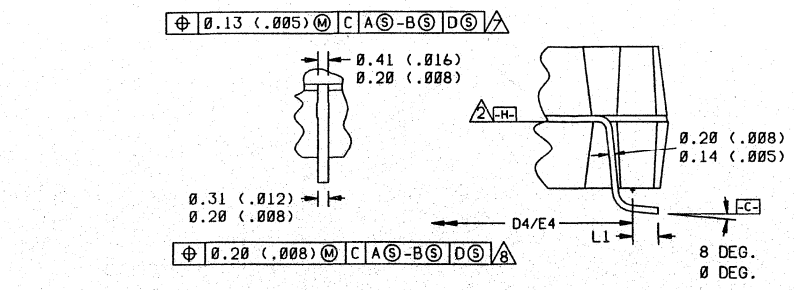


Figure 8-3.5. Typical Lead

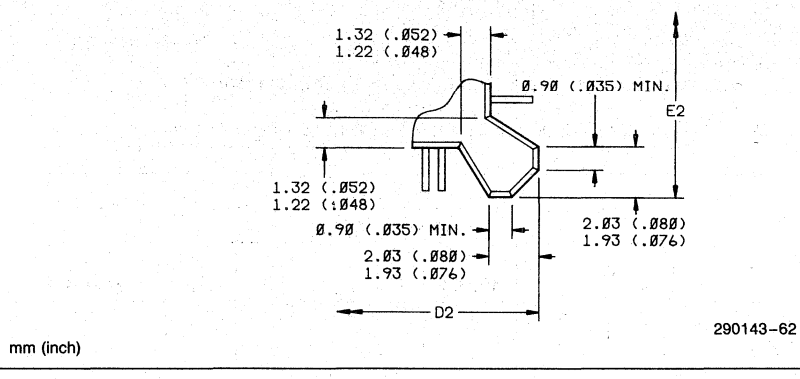


Figure 8-3.6. Detail M

PLASTIC QUAD FLAT PACK

Table 8-2. Symbol List for Plastic Quad Flat Pack

Letter or Symbol	Description of Dimensions
A	Package height: distance from seating plane to highest point of body
A1	Standoff: Distance from seating plane to base plane
D/E	Overall package dimension: lead tip to lead tip
D1/E1	Plastic body dimension
D2/E2	Bumper Distance
D3/E3	Footprint
L1	Foot length
N	Total number of leads

NOTES:
1. All dimensions and tolerances conform to ANSI Y14.5M-1982.
2. Datum plane -H- located at the mold parting line and coincident with the bottom of the lead where lead exits plastic body.
3. Datums A–B and -D- to be determined where center leads exit plastic body at datum plane -H-.
4. Controlling Dimension, Inch.
5. Dimensions D1, D2, E1 and E2 are measured at the mold parting line and do not include mode protrusion. Allowable mold protrusion of 0.18mm (0.007 in.) per side.
6. Pin 1 identifier is located within one of the two zones indicated.
7. Measured at datum plane -H-.
8. Measured at seating plane datum -C-.

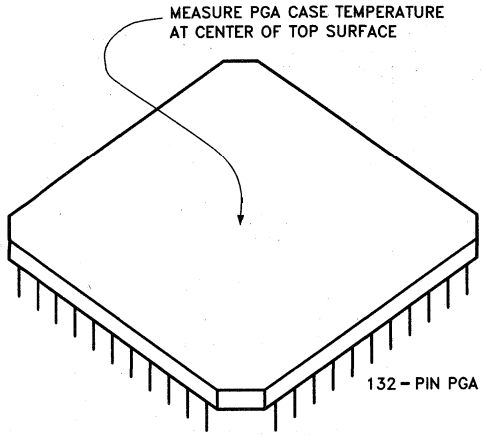


Figure 8-4. Measuring 82385 PGA Case Temperature

Table 8-3. 82385 PGA Package Typical Thermal Characteristics.

Parameter	Thermal Resistance—°C/Watt						
	Airflow—f^3/min (m^3/sec)						
	0 (0)	50 (0.25)	100 (0.50)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)
θ Junction-to-Case (Case Measured as Figure 8.4)	2	2	2	2	2	2	2
θ Case-to-Ambient (No Heatsink)	19	18	17	15	12	10	9
θ Case-to-Ambient (with Omnidirectional Heatsink)	16	15	14	12	9	7	6
θ Case-to-Ambient (with Unidirectional Heatsink)	15	14	13	11	8	6	5

NOTES:
1. Table 8-3 applies to 82385 PGA plugged into socket or soldered directly onto board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.
3. $\theta_{J\text{-}CAP}$ = 4°C/W (approx.)
   $\theta_{J\text{-}PIN}$ = 4°C/W (inner pins) (approx.)
   $\theta_{J\text{-}PIN}$ = 8°C/W (outer pins) (approx.)

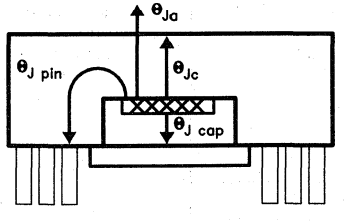


Table 8-4. 82385 132-Lead PQFP Package Typical Thermal Characteristics

Thermal Resistance—°C/Watt							
Parameter	**Airflow—lfm**						
	0	**50**	**100**	**200**	**400**	**600**	**800**
θ Junction-to-Case (Case Measured as Figure 8.4)	5	5	5	5	5	5	5
θ Case-to-Ambient (No Heatsink)	23.5	22.0	20.5	17.5	14.0	11.5	9.5
θ Case-to-Ambient (with Omnidirectional Heatsink)	**TO BE DEFINED**						
θ Case-to-Ambient (with Unidirectional Heatsink)							

NOTES:
1. Table 8-4 applies to 82385 PQFP plugged into socket or soldered directly onto board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.
3. $\theta_{J\text{-}CAP} = 4°C/W$ (approx.)
   $\theta_{J\text{-}PIN} = 4°C/W$ (inner pins) (approx.)
   $\theta_{J\text{-}PIN} = 8°C/W$ (outer pins) (approx.)

5

9.0 ELECTRICAL DATA

9.1 INTRODUCTION

This chapter presents the A.C. and D.C. specifications for the 82385.

9.2 MAXIMUM RATINGS

Storage Temperature ..... −65°C to +150°C

Case Temperature under Bias ... −65°C to +110°C

Supply Voltage with Respect to V_{SS} ..... −0.5V to +6.5V

Voltage on Any Other Pin .... −0.5V to V_{CC} + 0.5V

NOTE:
Stress above those listed may cause permanent damage to the device. This is a stress rating only and functional operation at these or any other conditions above those listed in the operational sections of this specification is not implied.

Exposure to absolute maximum rating conditions for extended periods may affect device reliability. Although the 82385 contains protective circuitry to resist damage from static electrical discharges, always take precautions against high static voltages or electric fields.

9.3 D.C. SPECIFICATIONS V_{CC} = 5V ±5%; V_{SS} = 0V

Table 9-1. D.C. Specifications

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{IL}	Input Low Voltage	−0.3	0.8	V	(Note 1)
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{CL}	CLK2, BCLK2 Input Low	−0.3	0.8	V	(Note 1)
V_{CH}	CLK2, BCLK2 Input High	3.7	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 4 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −1 mA
I_{CC}	Supply Current		300	mA	(Note 2) (Note 4)
I_{LI}	Input Leakage Current		±15	μA	$0V < V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current		±15	μA	$0.45 < V_{OUT} < V_{CC}$
C_{IN}	Input Capacitance		10	pF	(Note 3)
C_{OUT}	Output Capacitance		10	pF	(Note 3)
C_{CLK}	CLK2 Input Capacitance		15	pF	(Note 3)

NOTES:
1. Minimum value is not 100% tested.
2. I_{CC} is specified with inputs driven to CMOS levels. I_{CC} may be higher if driven to TTL levels.
3. Not 100% tested. Test conditions f_C = 1 MHz, Inputs = 0V, T_{CASE} = Room.
4. 300 mA is the maximum I_{CC} at 33 MHz.
   275 mA is the maximum I_{CC} at 25 MHz.
   250 mA is the maximum I_{CC} at 20 MHz.

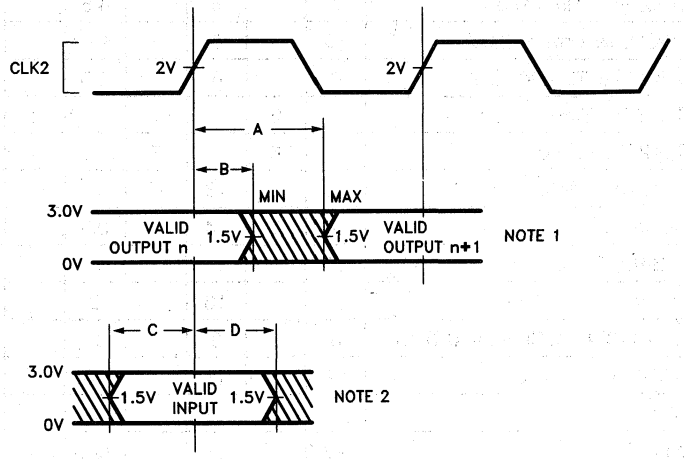
9.4 A.C. SPECIFICATIONS

The A.C. specifications given in the following tables consist of output delays and input setup requirements. The A.C. diagram's purpose is to illustrate the clock edges from which the timing parameters are measured. The reader should not infer any other timing relationships from them. For specific information on timing relationships between signals, refer to the appropriate functional section.

A.C. spec measurement is defined in Figure 9-1. Inputs must be driven to the levels shown when A.C. specifications are measured. 82385 output delays are specified with minimum and maximum limits, which are measured as shown. 82385 input setup and hold times are specified as minimums and define the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct 82385 operation.

9.4.1 Frequency Dependent Signals

The 82385 has signals whose output valid delays are dependent on the clock frequency. These signals are marked in the A.C. Specification Tables with a Note 1.



LEGEND:
A—Maximum output delay specification
B—Minimum output delay specification
C—Minimum input setup specification
B—Minimum input hold specification

NOTES:
1. Under rated loading 82385 output (t_r and t_f) is typically ≤ 4.0 ns from 0.8V to 2.0V.
2. Input waveforms have t_r ≤ 2.0 ns from 0.8V to 2.0V.

Figure 9-1. Drive Levels and Measurement Points for A.C. Specification

A.C. SPECIFICATION TABLES

Many of the A.C. Timing parameters are frequency dependent. The frequency dependent A.C. Timing parameters are guaranteed only at the maximum specified operating frequency.

Table 9-2. 82385 A.C. Timing Specifications

V_{CC} = 5.0 ±5%

Symbol	Parameter	20 MHz		25 MHz		33 MHz		Units	Notes
		Min	Max	Min	Max	Min	Max		
T_{CASE}	Case Temperature	0	85	0	75	0	75	°C	
t1	Operating Frequency	15.40	20.00	15.40	25.00	15.40	33.33	MHz	
t2	CLK2, BCLK2 Clock Period	25.00	32.50	20.00	32.50	15.00	32.50	ns	
t3a	CLK2, BCLK2 High Time @ 2.0V	10		8		6.25		ns	
t3b	CLK2, BCLK2 High Time @ 3.7V	7		5		4.5		ns	(Note 8)
t4a	CLK2, BCLK2 Low Time @ 2.0V	10		8		6.25		ns	
t4b	CLK2, BCLK2 Low Time @ 0.8V	8		6		4.5		ns	(Note 8)
t5	CLK2, BCLK2 Fall Time		8		7		4	ns	(Notes 8, 9)
t6	CLK2, BCLK2 Rise Time		8		7		4	ns	(Notes 8, 9)
t7a	A2–A19, A21–A31 Setup Time	19		18		13		ns	(Note 1)
t7b	LOCK# Setup Time	16		14		9.5		ns	(Note 1)
t7c	BE(0–3)# Setup Time	19		14		10		ns	(Note 1)
t7d	A20 Setup Time	13		13		9		ns	(Note 1)
t8	A2–A31, BE(0–3)# LOCK# Hold Time	3		3		3		ns	
t9a	M/IO#, D/C# Setup Time	22		17		13		ns	(Note 1)
t9b	W/R# Setup Time	22		18		13		ns	(Note 1)
t9c	ADS# Setup Time	22		18		13.5		ns	(Note 1)
t10	ADS#, D/C#, M/IO#, W/R# Hold Time	5		3		3		ns	
t11	READYI# Setup Time	12		8		7		ns	(Note 1)
t12	READYI# Hold Time	4		4		3		ns	
t13a1	NCA# Setup Time (See t55b2)	21		18		13		ns	(Note 6)
t13a2	NCA# Setup Time (See t55b3)	16		13		9		ns	(Note 6)
t13b	LBA# Setup Time	10		8		5.75		ns	
t13c	X16# Setup Time	10		7		5.5		ns	
t14a	NCA# Hold Time	4		3		3		ns	
t14b	LBA#, X16# Hold Time	4		3		3		ns	
t15	RESET, BRESET Setup Time	12		10		8		ns	
t16	RESET, BRESET Hold Time	4		3		2		ns	
t17	NA# Valid Delay	15	34	4	27	4	19.2	ns	(25 pF Load) (Note 1)
t18	READYO# Valid Delay	4	28	4	22	3	15	ns	(25 pF Load) (Note 1)
t19	BRDYEN# Valid Delay	4	28	4	21	3	13	ns	

Table 9-2. 82385 A.C. Timing Specifications (Continued)
V_{CC} = 5.0 ±5%

Symbol	Parameter	20 MHz		25 MHz		33 MHz		Units	Notes
		Min	Max	Min	Max	Min	Max		
t21a1	CALEN Rising, PHI1	3	24	4	21	3	15	ns	
t21a2	CALEN Falling, PHI1	3	24	4	21	3	15	ns	
t21a3	CALEN Falling in T1P, PHI2	3	24	4	21	3	15	ns	
t21b	CALEN Rising Following CWTH Cycle	3	34	4	27	3	20	ns	(Note 1)
t21c	CALEN Pulse Width	10		10		10		ns	
t21d	CALEN Rising to CS# Falling	13		13		13		ns	
t22a1	CWEx# Falling, PHI1 (CWTH)	4	25	4	23	3	18	ns	(Note 1)
t22a2	CWEx# Falling, PHI2 (CRDM)	4	25	4	23	3	18	ns	(Note 1)
t22b	CWEx# Pulse Width	30		25		20		ns	(Notes 1, 2)
t22c1	CWEx# Rising, PHI1 (CWTH)	4	25	4	21	3	16	ns	(Note 1)
t22c2	CWEx# Rising, PHI2 (CRDM)	12	25	8	21	6	16	ns	(Note 1)
t23a	CS(0–3)# Rising	12	37	9	29	3	25	ns	(Note 1)
t23b	COEx# Falling to CS(0–3)# Falling	0		0		0		ns	(Note 1)
t24	CT/R# Valid Delay	12	38	9	30	3	22	ns	(Note 1)
t25a	COEx# Falling (Direct)	1	22	4	19.5	3	15	ns	(25 pF Load)
t25b	COEx# Falling (2-Way)	1	24.5	4	19.5	3	15	ns	(25 pF Load) (Note 1)
t25c1	COEx# Rising Delay @ T_{CASE} = Min	5	17	4	17.5	3	12	ns	(25 pF Load)
t25c2	COEx# Rising Delay @ T_{CASE} = Max	5	19	4	19.5	3	12	ns	(25 pF Load)
t25d	CWEx# Falling to COEx# Falling or CWEx# Rising to COEX# Rising when DEFOE# = V_{CC}	0	5	0	5	0	5	ns	(25 pF Load)
t26	CS(0–3)# Falling to CWEx# Rising	30		25		20		ns	(Notes 1, 2)
t27	CWEx# Falling to CS(0–3)# Falling	0		0		0		ns	
t28a	CWEx# Rising to CALEN Rising	0		0		2		ns	
t28b	CWEx# Rising to CS(0–3)# Falling	0		0		2		ns	
t31	SA(2–31) Setup Time	19		10		8		ns	
t32	SA(2–31) Hold Time	3		3		3		ns	
t33	BADS# Valid Delay	6	28	4	21	3	16	ns	(Note 1)
t34	BADS# Float Delay	6	30	4	30	4	25	ns	(Note 3)
t35	BNA# Setup Time	9		7		7		ns	
t36	BNA# Hold Time	15		4		2		ns	
t37	BREADY# Setup Time	26		18		13		ns	(Note 1)
t38	BREADY# Hold Time	4		3		2		ns	
t40a	BACP Rising Delay	4	20	4	16	2	12	ns	
t40b	BACP Falling Delay	4	22	4	20	2	18	ns	

Table 9-2. 82385 A.C. Timing Specifications (Continued)
V_{CC} = 5.0 ±5%

Symbol	Parameter	20 MHz		25 MHz		33 MHz		Units	Notes
		Min	Max	Min	Max	Min	Max		
t41	BAOE# Valid Delay	4	18	4	15	2	12	ns	
t43a	BT/R# Valid Delay	2	19	4	16	2	14	ns	
t43b1	DOE# Falling Delay	2	23	4	20	2	16	ns	
t43b2	DOE# Rising Delay @ T_{CASE} = Min	4	17	4	17	2	12	ns	
t43b3	DOE# Rising Delay @ T_{CASE} = Max	4	19	4	19	2	14	ns	
t43c	LDSTB Valid Delay	2	26	2	21	2	16	ns	
t44a	SEN Setup Time	11		9		7		ns	
t44b	SSTB# Setup Time	11		5		5		ns	
t45	SEN, SSTB# Hold Time	5		5		2		ns	
M/S# = V_{CC} (Master Mode)									
t_{46}	BHOLD Setup Time	17		15		11		ns	
t47	BHOLD Hold Time	5		3		2		ns	
t48	BHLDA Valid Delay	5	28	4	23	3	16	ns	
M/S# = V_{SS} (Slave Mode)									
t49	BHLDA Setup Time	17		15		11		ns	
t50	BHLDA Hold Delay	5		3		2		ns	
t51	BHOLD Valid Delay	5	28	4	23	3	18	ns	
t55a	BLOCK# Valid Delay	4	30	4	26	3	20	ns	(Notes 1,5)
t55b1	BBE(0–3)# Valid Delay	4	30	4	26	3	20	ns	(Notes 1, 7)
t55b2	BBE(0–3#) Valid Delay	4	30	4	26	3	20	ns	(Notes 1, 7)
t55b3	BBE(0–3)# Valid Delay	4	36	4	32	3	23	ns	(Notes 1, 7)
t55c	LOCK# Valid to BLOCK# Valid	0	30	0	26	0	20	ns	(Notes 1, 5)
t56	MISS# Valid Delay	4	35	4	30	3	22	ns	(Note 1)
t57	MISS#, BBE(0–3)#, BLOCK# Float Delay	4	32	4	30	4	25	ns	(Note 3)
t58	WBS Valid Delay	4	37	4	25	3	16	ns	(Note 1)
t59	FLUSH Setup Time	16		12		10		ns	
t60	FLUSH Hold Time	5		5		3		ns	
t61	FLUSH Setup to RESET Falling	26		21		16		ns	(Note 4)
t62	FLUSH Hold to RESET Falling	26		21		16		ns	(Note 4)

NOTES:
1. Frequency dependent specification.
2. Used for cache data memory (SRAM) specifications.
3. Float times not 100% tested.
4. This feature is tested only at 16 MHz.
5. BLOCK# delay is either from BPHI1 or from 386 LOCK#. Refer to Figure 5-3K and 5-3L in the 82385 data sheet.
6. NCA# setup time is now specified to the rising edge of PHI2 in the state after 386 DX addresses become valid (either the first T2 or the state after the first T2P).
7. BBE# Valid delay is a function of NCA# setup.
8. Not 100% tested.
9. t5 is measured from 0.8V to 3.7V.
   t6 is measured from 3.7V to 0.8V
   This parameter is not 100% tested and is guaranteed by Intel's test methodology.

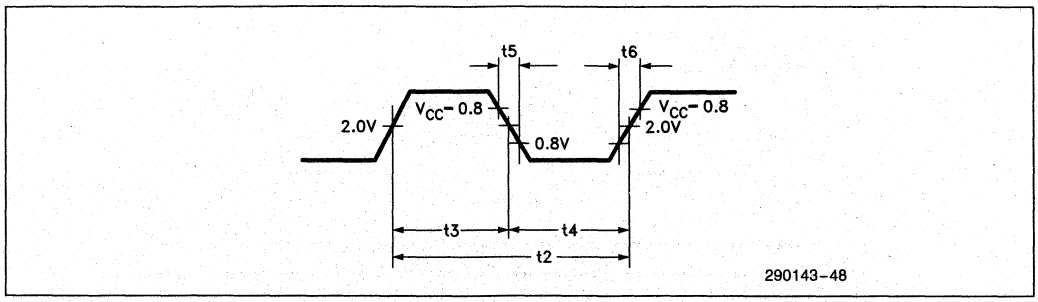
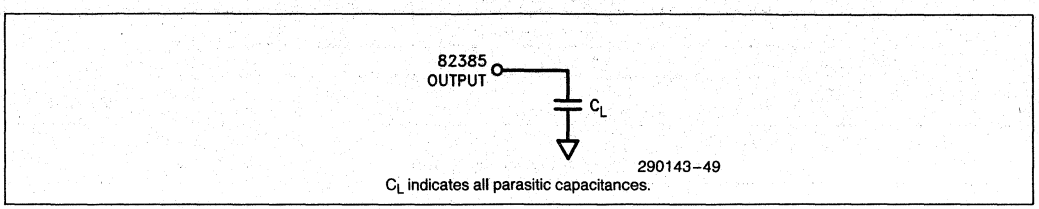


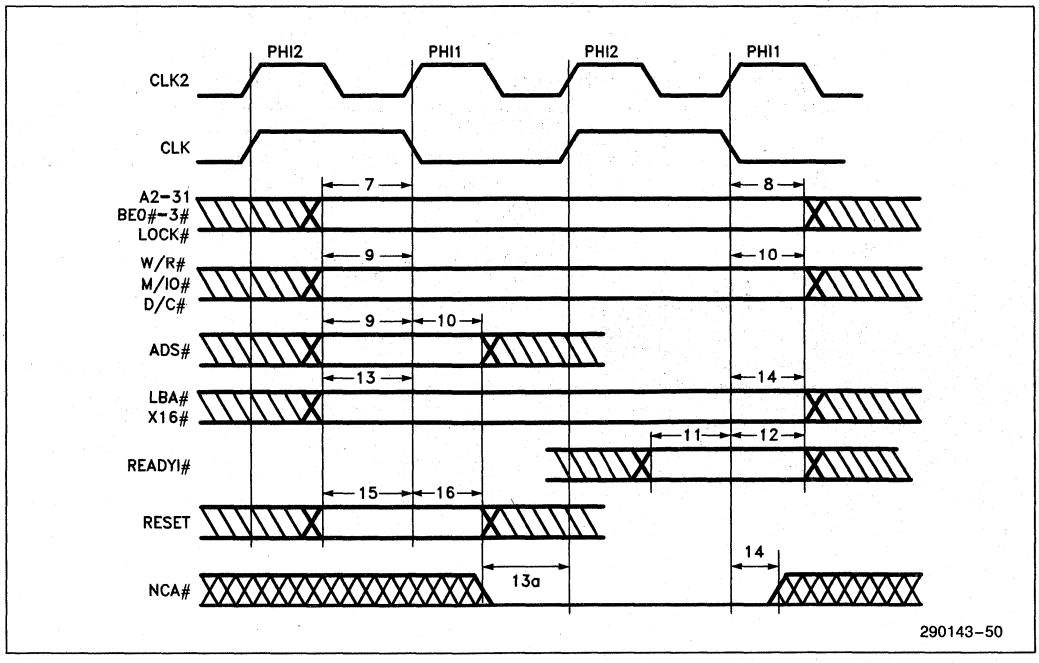
Figure 9-2. CLK2, BCLK2 Timing



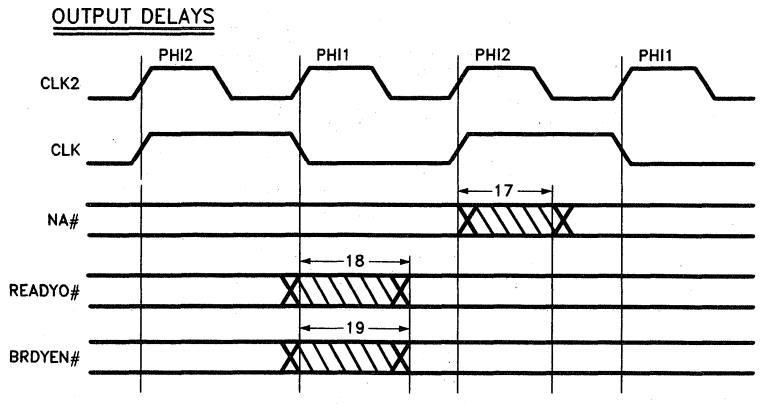
C_L indicates all parasitic capacitances.

Figure 9-3. A.C. Test Load

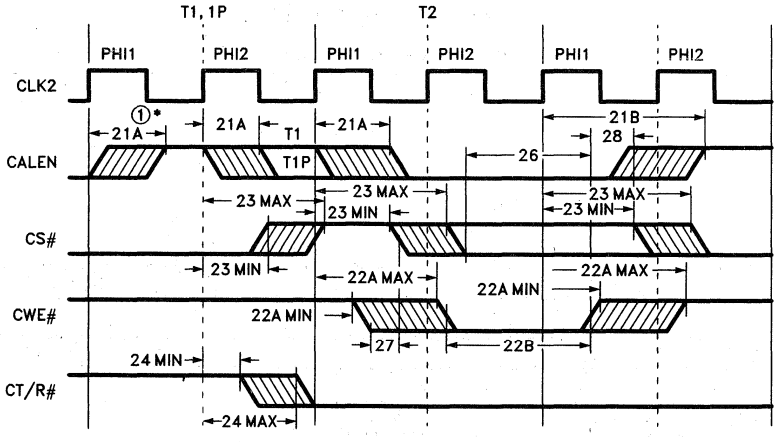
386 DX Interface Parameters



5

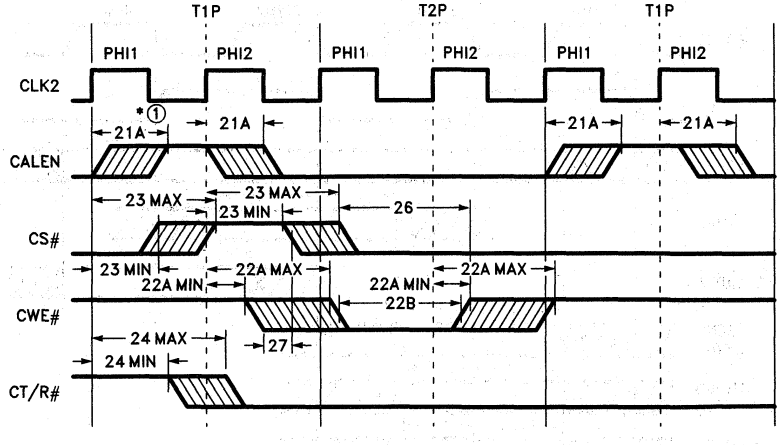


Cache Write Hit Cycle



① *. This would be 21B if previous bus cycle was Cache Write Hit cycle.

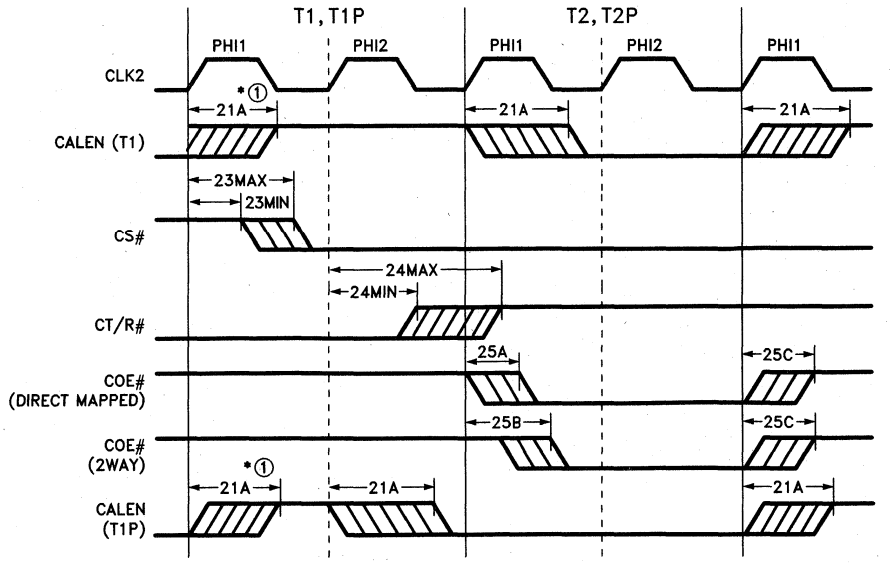
Cache Read Miss (Cache Update Cycle)



290143-53

*①. This would be 21B if previous bus cycle was Cache Write Hit cycle.

Cache Read Cycle

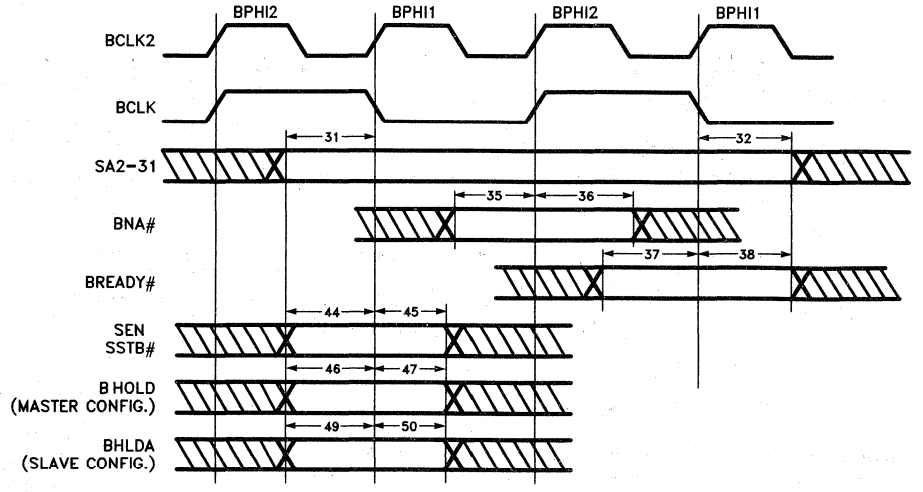


290143-54

*①. This would be 21B if previous bus cycle was Cache Write Hit cycle.

5

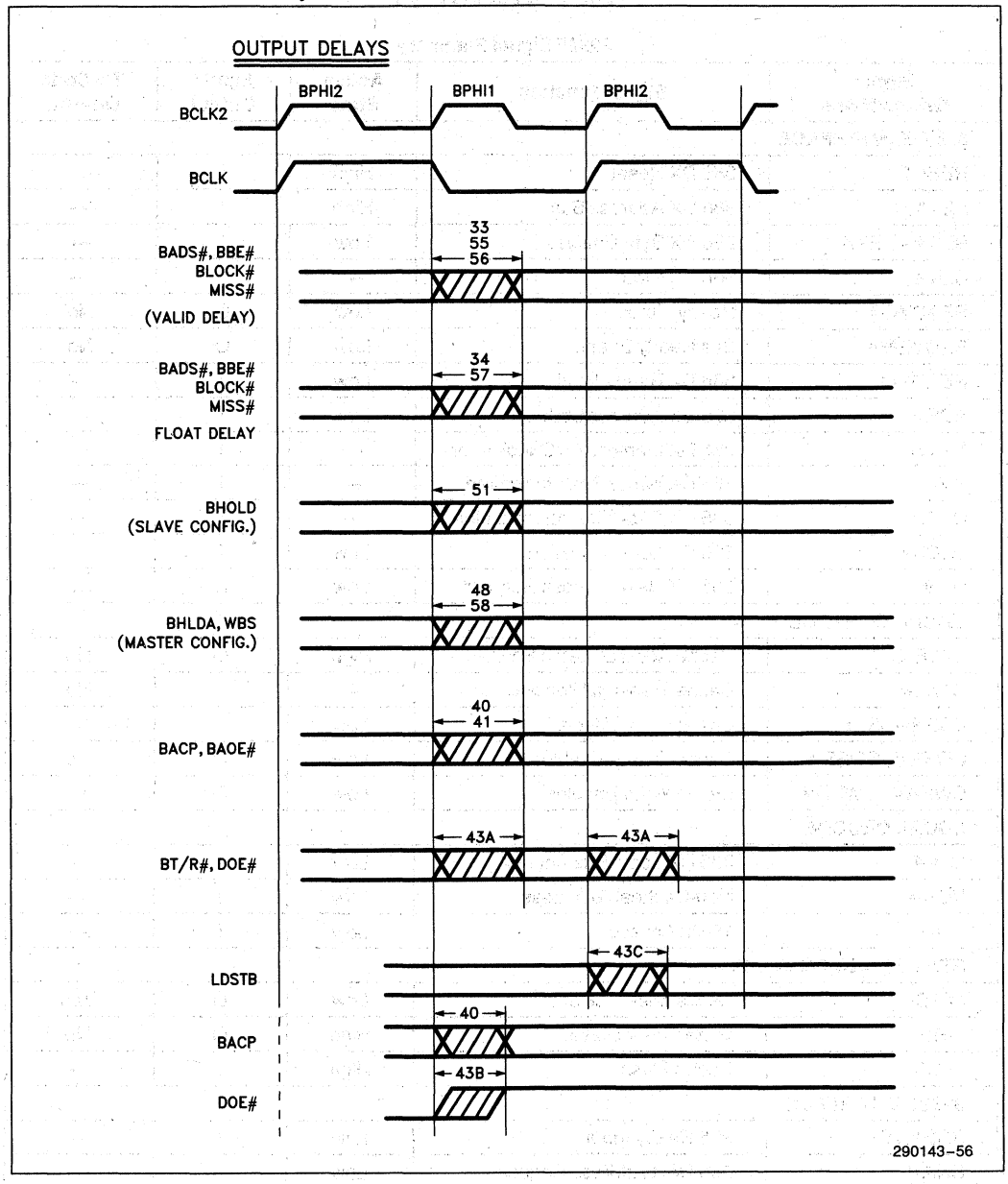
System Bus Interface Parameters



290143–55

*This would be 21B if previous cycle was Cache Write Hit.

System Bus Interface Parameters (Continued)



APPENDIX A

82385 Signal Summary

Signal Group/Name	Signal Function	Active State	Input/ Output	Tri-State Output?
386 DX INTERFACE				
RESET	386 DX Reset	High	I	—
A2–A31	386 DX Address Bus	High	I	—
BE0#–BE3#	386 DX Byte Enables	Low	I	—
CLK2	386 DX Clock	—	I	—
READYO#	Ready Output	Low	O	No
BRDYEN#	Bus Ready Enable	Low	O	No
READYI#	386 DX Ready Input	Low	I	—
ADS#	386 DX Address Status	Low	I	—
M/IO#	386 DX Memory / I/O Indication	—	I	—
W/R#	386 DX Write/Read Indication	—	I	—
D/C#	386 DX Data/Control Indication	—	I	—
LOCK#	386 DX Lock Indication	Low	I	—
NA#	386 DX Next Address Request	Low	O	No
CACHE CONTROL				
CALEN	Cache Address Latch Enable	High	O	No
CT/R#	Cache Transmit/Receive	—	O	No
CS0#–CS3#	Cache Chip Selects	Low	O	No
COEA#, COEB#	Cache Output Enables	Low	O	No
CWEA#, CWEB#	Cache Write Enables	Low	O	No
LOCAL DECODE				
LBA#	386 DX Local Bus Access	Low	I	—
NCA#	Non-Cacheable Access	Low	I	—
X16#	16-Bit Access	Low	I	—
STATUS AND CONTROL				
MISS#	Cache Miss Indication	Low	O	Yes
WBS	Write Buffer Status	High	O	No
FLUSH	Cache Flush	High	I	—
82385 INTERFACE				
BREADY#	385 Ready Input	Low	I	—
BNA#	385 Next Address Request	Low	I	—
BLOCK#	385 Lock Indication	Low	O	Yes
BADS#	385 Address Status	Low	O	Yes
BBE0#–BBE3#	385 Byte Enables	Low	O	Yes

82385 Signal Summary (Continued)

Signal Group/Name	Signal Function	Active State	Input/ Output	Tri-State Output?
DATA/ADDR CONTROL				
LDSTB	Local Data Strobe	Pos. Edge	O	No
DOE#	Data Output Enable	Low	O	No
BT/R#	Bus Transmit/Receive	—	O	No
BACP	Bus Address Clock Pulse	Pos. Edge	O	No
BAOE#	Bus Address Output Enable	Low	O	No
CONFIGURATION				
2W/D#	2-Way/Direct Map Select	—	I	—
M/S#	Master/Slave Select	—	I	—
DEFOE#	Define Cache Output Enable	—	1	—
COHERENCY				
SA2–SA31	Snoop Address Bus	High	I	—
SSTB#	Snoop Strobe	Low	I	—
SEN	Snoop Enable	High	I	—
ARBITRATION				
BHOLD	Hold	High	I/O	No
BHLDA	Hold Acknowledge	High	I/O	No

5

10.0 REVISION HISTORY

DOCUMENT: ADVANCE INFORMATION DATA SHEET
PRIOR REV: 290143-003 September 1988
NEW REV: 290143-004 September 1989

Change #	Page #	Para. #	Change
1.	Throughout	Fig. 8-3	PQFP Package added
2.	Throughout	Tables 8-2, 8-3	PQFP Info
3.	Throughout	Table 8-4	PQFP Thermal Resistance
4.	Throughout		A.C. Specifications Unified (20 MHz, 25 MHz, 33 MHz)
5.	Throughout		DEFOE# Specifications added to device

intel®

APPLICATION NOTE

AP-442

June 1990

33 MHz 386™ System Design Considerations

SHAHZAD BAQAI

KIYOSHI NISHIDE

Order Number: 240725-001

33 MHz 386™ SYSTEM DESIGN CONSIDERATIONS

CONTENTS

PAGE

1.0 INTRODUCTION 5-622

2.0 HIGH SPEED SYSTEM DESIGN CONSIDERATIONS 5-624

2.1 Overview of high speed effects 5-624

2.2 Transmission Line Effects 5-627

2.3 Reflection 5-628

2.4 Cosstalk 5-633

2.5 Skew 5-633

2.6 D.C. Loading 5-634

2.7 A.C. (Capacitive) Loading 5-634

2.8 Derating Curve 5-634

2.9 High Speed Clock Circuits 5-635

CONTENTS

PAGE

3.0 DESIGN EXAMPLE 5-637

3.1 System Architecture for High Speeds 5-637

3.2 CPU Subsystem 5-637

3.3 DRAM Subsystem 5-637

3.4 Cache Subsystem 5-639

3.5 I/O, EPROM Subsystem 5-645

APPENDIX A
Schematics 5-649

APPENDIX B
State Diagrams and Palcodes 5-660

APPENDIX C
Timing Diagrams 5-707

APPENDIX D
Timing Equations 5-720

APPENDIX E
References 5-730

RELATED DOCUMENTATION

This Application Note should be used in conjunction with the 386™ DX microprocessor Data Sheet (Order Number 231630-007) and the 386™ DX Hardware Reference Manual (Order Number 231732-004). A list of related references is provided in the appendix for getting more information on high speed design issues.

INTRODUCTION

The 386™ DX Microprocessor is an advanced 32-bit microprocessor designed using Intel's CHMOS IV process for applications which require very high performance. It is optimized for multitasking operating systems. The 32-bit register and data paths support 32-bit address and data types allowing up to four gigabytes of physical memory and 64 terabytes of virtual memory to be addressed. The integrated memory management and protection architecture includes address translation registers, advanced multitasking hardware and a protection mechanism to support operating systems. In addition, the 386 DX microprocessor allows the simultaneous running of DOS with other operating systems. Instruction pipelining, on chip address translation and high bus bandwidth ensure short average instruction execution times and high system throughput. To facilitate high performance system hardware designs, the 386 DX microprocessor bus interface offers address pipelining, dynamic data bus sizing and direct byte enable signals for each byte of the data bus.

This Application Note is intended to show how to complete a successful design of a 'Core' system using the 386 DX-33, the 33 MHz clock version. A Core system is a minimum system configuration, in this case comprising the CPU, the 82385 32-bit Cache controller, Dynamic and Static RAM and an I/O mechanism with which to communicate with the CPU.

The Application Note examines the design techniques necessary when executing a design at this frequency. Many of the methods used at lower frequencies, such as 16 MHz and 20 MHz, are no longer valid at this higher frequency. Phenomena, whose effects are negligible at the lower frequencies, must be taken into account in the design. The physical positioning of components relative to each other plays a significant part in the success of the design, since transmission line effects (reflection, radiation) are no longer negligible.

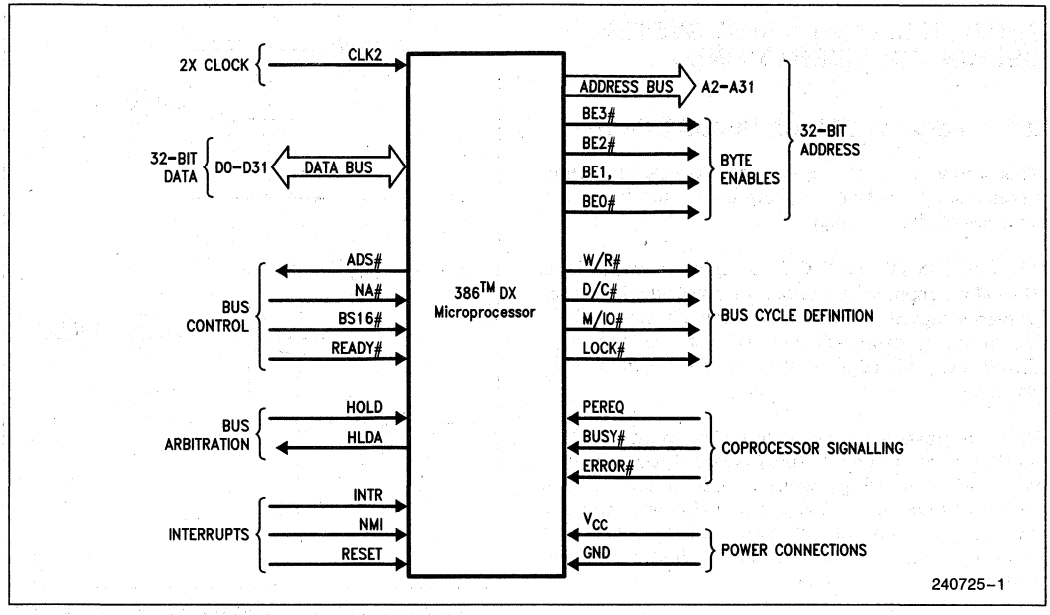


Figure 1-1. Functional Signal Groups

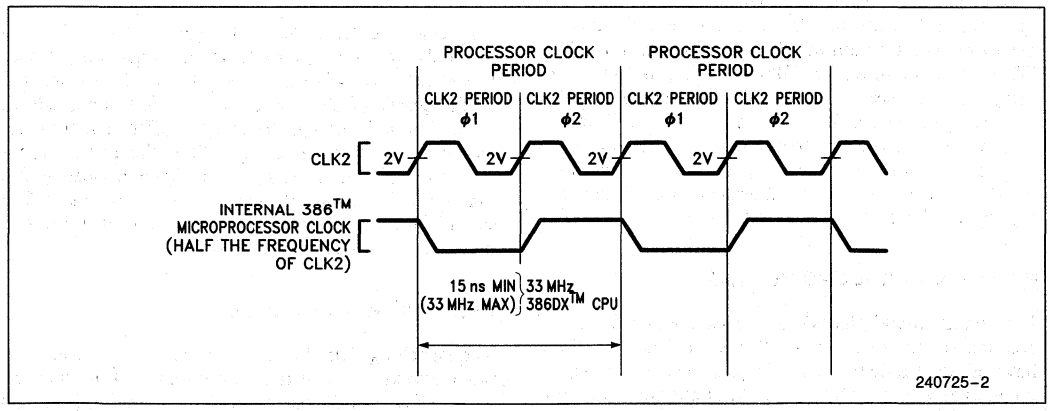


Figure 1-2. CLK2 Signal and Internal Processor Clock

5

SECTION II. HIGH SPEED SYSTEM DESIGN CONSIDERATIONS

2.1 Overview Of High Speed Effects

This section is included as a brief overview of general issues that are applicable to both higher and lower frequencies of circuit design.

The CHMOS IV 386 DX CPU differs from previous HMOS microprocessors in that its power dissipation is primarily capacitive; there is almost no DC power dissipation. Power dissipation depends mostly on frequency. This fact is used in designs where power consumption is critical.

Power dissipation can be distinguished as either internal (logic) power or I/O (bus) power. Internal power varies with operating frequency and to some extent with wait states and software. Internal power increases with supply voltage also. Process variations in manufacturing affect internal power, although to a lesser extent than with NMOS processes.

I/O power, which accounts for roughly one-fifth of the total power dissipation, varies with frequency and voltage. It also depends on capacitive bus load. Capacitive bus loadings for all output pins are specified in the 386 DX CPU data sheet. The 386 DX CPU output valid delays will increase if these loadings are exceeded. The addressing pattern of the software can affect I/O power by changing the effective frequency at the address pins. The variation in frequency at the data pins tends to be smaller; thus varying data patterns should not cause a significant change in power dissipation.

POWER AND GROUND PLANES

Power and ground planes must be used in 386 DX CPU systems to minimize noise. Power and ground lines have inherent inductance and capacitance, therefore an impedance $z = (L/C)^{*1/2}$. The total characteristic impedance for the power supply can be reduced by adding more lines. This effect is illustrated in 2.1 which shows that two lines in parallel have half the impedance of one. To reduce the impedance even further, the user should add more lines. In the limit, an infinite number of parallel lines, or a plane, results in the lowest impedance. Planes also provide the best distribution of power and ground.

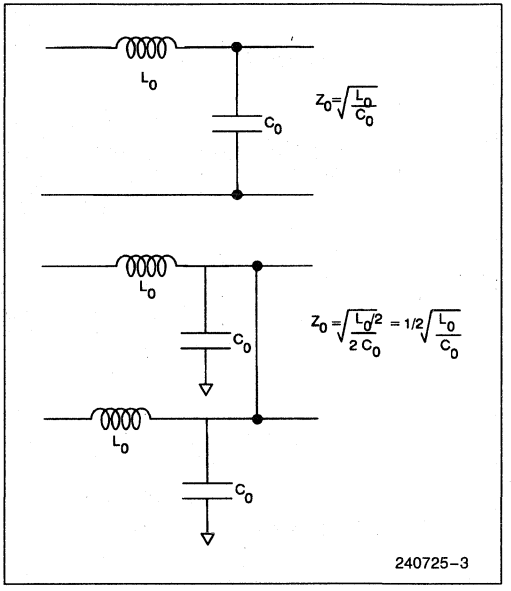


Figure 2-1. Reducing Characteristic Impedance

The 386 DX CPU has 20 V_{CC} pins and 21 V_{SS} (ground) pins. All power and ground pins must be connected to a plane. Ideally, the 386 DX CPU is located at the center of the board, to take full advantage of these planes. Although the 386 DX CPU generally demands less power than the 80286, the possibility of power surges is increased due to higher frequency and pin count. Peak-to-peak noise on V_{CC} relative to V_{SS} should be maintained at no more than 400 mV, and preferably to no more than 200 mV.

DECOUPLING CAPACITORS

The switching activity of one device can propagate to other devices through the power supply. For example, in the TTL NAND gate of Figure 2.2, both Q3 and Q4 transistors are on for a short time when the output is switching. This increased load causes a negative spike on V_{CC} and a positive spike on ground.

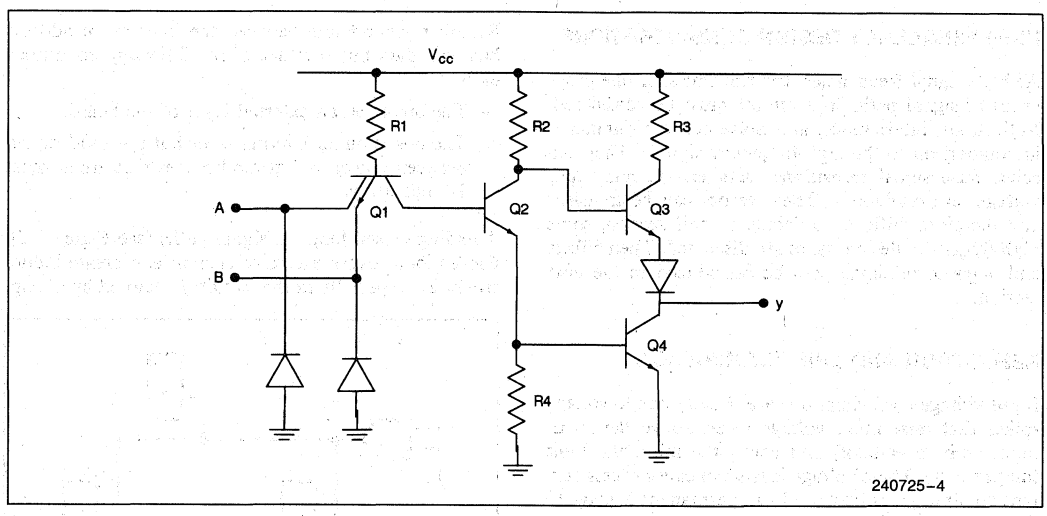


Figure 2-2. Circuit without Decoupling

In synchronous systems in which many gates switch simultaneously, the result is signifcant noise on the power and ground lines.

Decoupling capacitors placed across the device between Vcc and ground reduce Voltage spikes by supplying the extra current needed during switching. These capacitors should be placed close to their devices because the inductance or connection lines negates their effect.

When selecting decoupling capacitors, the user should provide 0.01 microfarads for each device and 0. 1 microfarads for every 20 gates. Radio-frequency capacitors must be used; they should be distributed evenly over the board to be most effective. In addition, the board should be decoupled from the external supply line with a 2.2 microfarad capacitor.

Chip capacitors (surface-mount) are preferable because they exhibit lower inductance and require less total board space. They should be connected as in Figure 2.3. Leaded capacitors can also be used if the leads are kept as short as possible. Six leaded capacitors are required to match the effectiveness of one chip capacitor, but because only a limited number can fit around the 386 DX, the configuration in Figure 2.4 results.

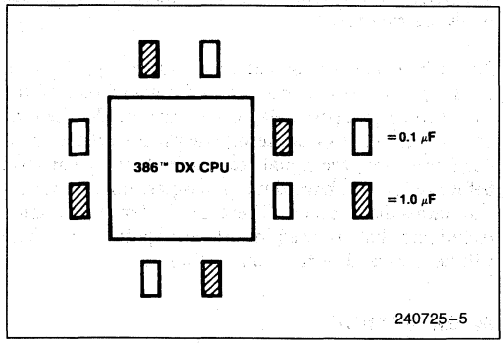


Figure 2-3. Decoupling Chip Capacitors

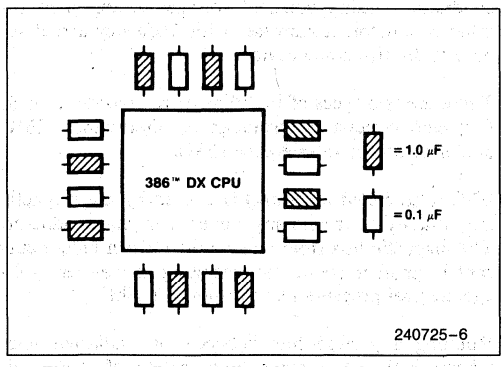


Figure 2-4. Decoupling Leaded Capacitors

5

HIGH FREQUENCY DESIGN CONSIDERATIONS

At high signal frequencies, the transmission line properties of signal paths in a circuit must be considered. Reflections, interference, and noise become significant in comparison to the high-frequency signals. They can cause false signal transitions, data errors, and input voltage level violations. These errors can be transient and therefore difficult to debug. In this section, some high-frequency design issues are discussed. Their effects and ways to minimize will be introduced in the next section.

REFLECTION AND LINE TERMINATION

Input voltage level violations are usually due to voltage spikes that raise input voltage levels above the maximum limit (overshoot) and below the minimum limit (undershoot). These voltage levels can cause excess current on input gates that results in permanent damage to the device. Even if no damage occurs, most devices are not guaranteed to function as specified if input voltage levels are exceeded.

Signal lines are terminated to minimize signal reflections and prevent overshoot and undershoot. If the round-trip signal path delay is greater than the rise time or fall time of the signal, terminate the line. If the line is not terminated, the signal reaches its high or low level before reflections have time to dissipate, and overshoot and undershoot occur. There are a few termination techniques that are used in different applications, these will be discussed in the next section.

INTERFERENCE

Interference is the result of electrical activity in one conductor causing transient voltages to appear in another conductor. It increases with frequency and closeness of the two conductors.

There are two types of interference to consider in high frequency circuits: electromagnetic interference (EMI) and electrostatic interference (ESI).

EMI (also called crosstalk) is caused by the magnetic field that exists around any current carrying conductor. The magnetic flux from one conductor can induce current in another conductor, resulting in transient voltage. Several precautions can minimize EMI.

Running a ground line between two adjacent lines wherever they traverse a long section of the circuit board. The ground line should be grounded at both ends.

Running ground line between the lines of an address bus or a data bus if either of the following conditions exist.

— The bus is on an external layer of the board.

— The bus is on an internal layer but not sandwiched between power and ground planes that are at most 10 mils away.

Avoiding closed loops in signal paths (see Figure 2.5). Closed loops cause excessive current and create inductive noise, especially in the circuitry enclosed by a loop.

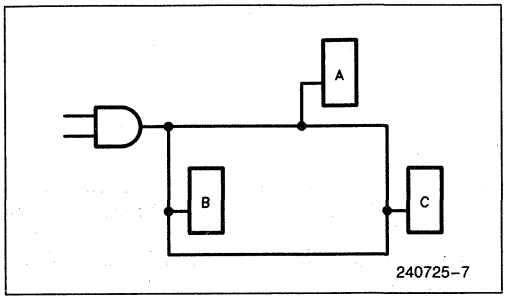


Figure 2-5. Avoid Closed-Loop Signal Paths

ESI is caused by the capacitive coupling of two adjacent conductors. The conductors act as the plates of a capacitor; a charge built up on one induces the opposite charge on the other.

The following steps reduce ESI:

Separating signal lines so that capacitive coupling becomes negligible.

Running a ground line between two lines to cancel the electrostatic fields.

LATCHUP

Latchup is a condition in a CMOS circuit in which V_{CC} becomes shorted to Vss. Intel's CHMOS IV process is immune to latchup under normal operating conditions. Latchup can be triggered when the voltage limits on I/O pins are exceeded, causing internal PN junctions to become forward biased. The following guidelines help prevent latchup:

Observing the maximum rating for input voltage on I/O pins.

Never applying power to an 386 DX CPU pin or a device connected to an 386 DX CPU pin before applying power to the 386 DX CPU itself.

Preventing overshoot and undershoot on I/O pins by adding line termination and by designing to reduce noise and reflection on signal lines.

THERMAL CHARACTERISTICS

The thermal specification for the 386 DX CPU defines the maximum case temperature. This section describes how to ensure that an 386 DX CPU system meets this specification.

Thermal specifications for the 386 DX CPU are designed to guarantee a tolerable temperature at the surface of the 386 DX CPU chip. This temperature (called the junction temperature) can be determined from external measurements using the known thermal characteristics of the package. Two equations for calculating junction temperature are as follows:

$T_j = T_a + (@j_a * PD)$ and

$T_j = T_c + (@j_c * PD)$

where:

T_j = Junction Temperature
$@j_a$ = Junction to ambient temperature coeff.
T_c = Case Temperature
T_a = Ambient Temperature
$@j_c$ = Junction to Case
PD = Power Dissapation temperature coeff.

Case temperature calculations offer several advantages over ambient temperature calculations.

Case temperature is easier to measure accurately than ambient temperature because the measurement is localized to a single point (top center of the package).

The worst-case junction temperature (T_j) is lower when calculated with case temperature for the following reasons:

— The junction-to-case thermal coefficient ($@j_c$) is lower than the junction-to-ambient thermal coefficient ($@j_a$); therefore, calculated junction temperature varies less with power dissipation (PD).
— $@j_c$ is not affected by airflow in the system; $@j_a$ varies with air flow.

With the case-temperature specification, the designer can either set the ambient temperature or use fans to control case temperature. Finned heat sinks or conductive cooling may also be used in environments where the use of fans is precluded. To approximate the case temperature for various environments, the two equations above should be combined by setting the junction temperature equal for both, resulting in this equation:

$$T_a = T_c - ((@j_a - @j_c) * PD)$$

The current data sheet should be consulted to determine the values of @ja (for the system's air flow) and ambient temperature that will yield the desired case temperature. Whatever the conditions are, the case temperature is easy to verify.

2.2 Transmission Line Effects

As a general rule, any interconnection is considered a transmission line when the time required for the signal to travel the length of the interconnection is greater than one-eighth of the signal rise time. (True K. M. , "Reflection: Computations and Waveforms, The Interface Handbook", Fairchild Corp, Mountain View, CA, 1975, Ch. 3). As frequencies increase, designers must account for the negative effects associated with transmission lines. The section that follows will attempt to describe these effects and provide some suggestions for minimizing their negative effect on the system.

Before describing each effect, it is important to know how to characterize a trace on different types of transmision lines. This includes knowing the characteristic impedance of a trace, Z_o, and the propagation delay for a given trace, t_{pd}. These parameters will be used in determining what effects must be accounted for and to select component values used in minimizing the effects.

5

TRANSMISSION LINES TYPES

Although many types of transmission lines (conductors) exist, those most commonly used on the printed circuit boards are microstrip lines, strip lines, printed circuit traces, side-by-side conductors and flat conductors.

MICRO STRIP LINES

The micro strip trace consists of a signal plane that is seperated from a ground plane by a dielectric as shown in Figure 2.6. G-10 fiber-glass epoxy, which is most common, has an $e_r = 5$ where e_r is the dielctric constant of the insulation. Let:

w = the width of the signal line (inches)
t = the thickness of copper
h = the height of dielectric for controlled impedance (inches)

The characteristic impedance Z_o, is a function of dielectric constant and the geometry of the board. This is given by:

$$Z_o = (87/(e_r + 1.41)^{1/2} \ln (5.98/0.8\ w + t)\ \Omega$$

where e_r is the relative dielectric constant of the board material.

The propagation delay (t_{pd}) associated with the trace is a function of the dielectric only.

$$t_{pd} = 1.017\ (0.475e_r + 0.67)^{1/2}\ \text{ns/ft}$$

STRIP LINES

A strip line is a strip conductor centered in a dielectric medium between two voltage planes. The characteristic impedance is given by:

$$Z_o = 60/(e_r)^{1/2} \ln (5.98b/(0.8W + t))\ \Omega$$

where b = distance between the planes for the controlled impedance as shown in Figure 2.10

The propagation delay is given by:

$$t_{pd} = 1.017\ (e_r)^{1/2}\ \text{ns/ft}$$

Typical values of the characteristic impedance and propagation delay of these types of lines are:

$$Z_O = 50\Omega$$

$$t_{pd} = 2\ \text{ns/ft (or 6 in/ns)}$$

2.3 Reflection

The first effect is reflection. As the name indicates it is the reflection of a signal as it propagates down the trace. The reflection results from a mismatch in impedance. The impedance of a transmission line is a function of the geometry of the line, its distance from the ground plane, and the loads long the line. Any discontinuity in the impedance will cause reflections.

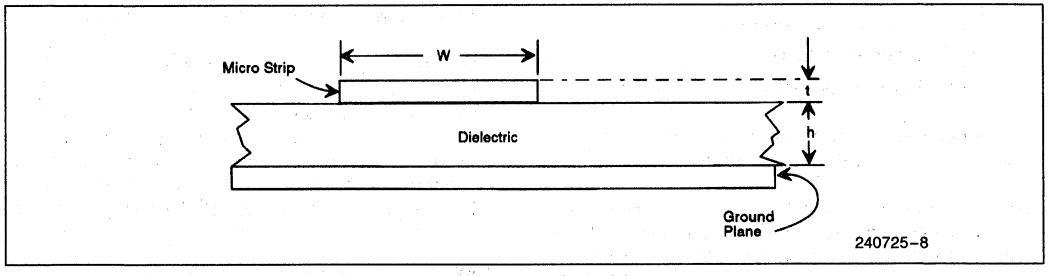


Figure 2-6. Micro Strip Lines

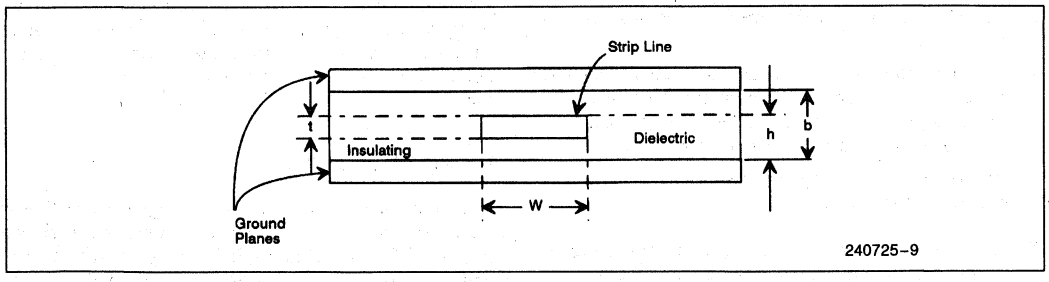


Figure 2-7. Strip Lines

Impedance mismatch occurs between the transmission line characteristic impedance and the input or output impedance of the devices that are connected to the line. The result is that the signals are reflected back and forth on the line. These reflections can attentuate or reinforce the signal depending upon the phase relationships. The results of these reflections include overshoot, undershoot, ringing and other undesirable effects.

At lower edge rates, the effects of these reflections are not severe. However at higher rates, the rise time of the signal is short with respect to the propagation delay. Thus it can cause problems as shown in Figure 2-8.

Overshoot occurs when the voltage level exceeds the maximum (upper) limit of the output voltage, while undershoot occurs when the level passes below the minimum (lower) limit. These conditions can cause excess current on the input gates which results in permanent damage to the device.

The amount of reflection voltage can be easily calculated. Figure 2-9 shows a system exhibiting reflections.

The magnitude of a reflection is usually represented in terms of a reflection coefficient. This is illustrated in the following equations:

$$T = v_r/v_i = \text{Reflected voltage/Incident voltage}$$

$$T_{load} = (Z_{load} - Z_O)/(Z_{load} + Z_O)$$

$$T_{source} = (Z_{source} - Z_O)/(Z_{source} + Z_O)$$

Reflections voltage V_r is given by V_i, the voltage incident at the point of the reflections, and the reflection coefficient.

The model transmission line can now be completed. In Figure 2-9, the voltage seen at point A is given by the following equation:

$$V_a = V_s * Z_O/(Z_O + Z_s)$$

This voltage V_a enters the transmission line at "A" and appears at "B" delayed by t_{pd}.

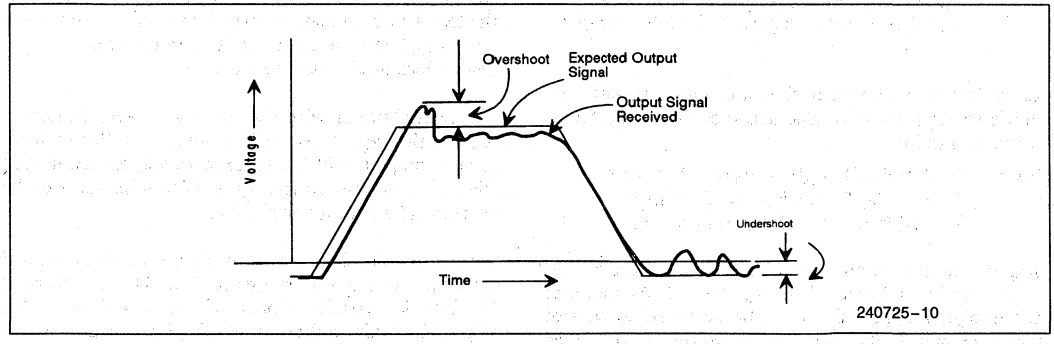


Figure 2-8. Overshoot and Undershoot Effects

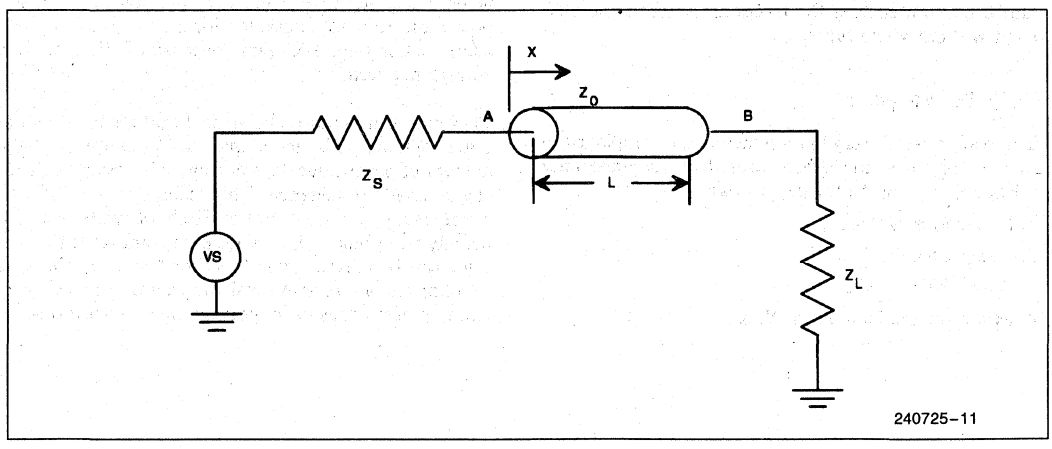


Figure 2-9. Loaded Transmission Line

$$V_b(t - x/t)\ H(t - x/v)$$

where x = distance along the transmission line from point "A" and H(t) is the unit step function. The waveform encounters the loads ZL, and this may cause reflection. The reflected wave enters the transmission line at "B" and appears at point "A" after time delay (t_{pd}):

$$V_{r1} = T_{load} * V_b$$

This phenomenon continues infinitely, but it is negligible after 3 or 4 reflections. Hence:

$$V_{r2} = T_{source} * V_{r1}$$

Each reflected waveform is treated as a seperate source that is independent of the reflection coefficient at that point and the incident waveform. Thus the waveform from any point and on the transmission line and at any given time is as follows:

$$\begin{aligned} V(x,t) = {} & (Z_O/(Z_O+Z_s))\ \{V_s(t-(x/v))\ H(t-(x/v)) + \\ & T_1\ [V_s(t-((2L-x)/v)\ H(t-(t-((2L-x)/v)))] + \\ & T_1T_s\ [V_s(t-((2L+x)/v)\ H(t-(t-((2L+x)/v)))] + \\ & T_{12}T_s\ [V_s(t-((4L-x)/v)\ H(t-(t-((4L-x)/v)))] + \\ & T_{12}T_s^2\ [V_s(t-((4L+x)/v)\ H(t-(t-((4L+x)/v)))] \\ & + \ldots\} \end{aligned}$$

Each reflection is added to the total voltage through the unit step function H(t). The above equation can be rewritten as follows:

$$\begin{aligned} V(x,t) = {} & (Z_O/(Z_O+Z_s))\ \{V_s(t-(t-t_{pd}x)\ H(t-t_{pd}x) + \\ & T_1\ [V_s(t-t_{pd}(2L-x))\ H(t-t_{pd}(2L-x))] + \\ & T_1T_s\ [V_s(t-t_{pd}(2L+x))\ H(t-t_{pd}(2L+x))] + \ldots\} \end{aligned}$$

Impedance discontinuity problems are managed by imposing limits and control during the routing phase of the design. Design rules must be observed to control trace geometry, including specification of the trace width and spacing for each layer. This is very important because it ensures the traces are smooth and constant without sharp turns.

HOW TO MINIMIZE

There are several techniques which can be employed to further minimize the effects caused by an impedance mismatch during the layout process:

1. Impedance Matching
2. Daisy Chaining
3. Avoid 90° Corners
4. Minimize the Number of Vias

IMPEDANCE MATCHING

Impedance matching is the process of matching the impedance of the the source or load to the impedance of the trace. This matching is accomplished using a technique called termination. Termination makes the effective source or load impedance, seen by the trace, to be approximately equal to the characteristic impedance of the trace. Before terminating a line one must determine if termination is required. This is done by a simple calculation. If the propagation delay down a trace from source to destination is greater than or equal to one-third the signals rise time, termination is needed. (i. e. $T_{pd} \geq \frac{1}{3}\ t_r$). The rise time is the 0%-100% rise time specified for the source. If this value is specified for 10%-90% or 20%-80%, it must be scaled by multiplying the specified value by 1.25 or 1.67, respectively. The propagation delay is caculated by multiplying the trace propagation delay, t_{pd}, descibed earlier by the trace length.

Once it is determined that termination is needed, use the equation described earlier to calculate the trace's characteristic impedance. The specification sheets for the load can be consulted to determine the load impedance, Z_L. These values are needed to select the component values used to terminate.

The next chore is selecting the type of termination to use. In this section we will examine 4 different techniques and point out the advantages and disadvantages. Figure 2.10 shows the four types of termination and the corresponding component values.

Parallel termination, shown in Figure 2-10(a), is a good technique to maintain the waveform. The waveform at the load is a perfect image of the waveform at the source. In addition there is no added propagation delay associated with this technique. The disadvantage of this technique is that it requires a fair amount of additional power and it is not suggested for characteristic impedances of less than 100 ohms because of the large d.c. current required.

Thevenin termination, shown in Figure 2-10(b), is another option. This technique also requires a large amount of power, but does not have the restrictions for characteristic impedance. This technique is very good at removing overshoot and undershoot while not adding any additional delay. Another advantage is that the trace can be biased toward Vcc or GND by simpling selecting the appropriate resistor values. This can help maintain fast edges on important signal transitions.

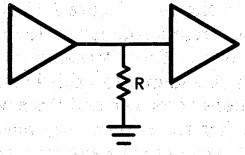
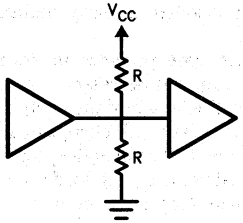
Name	Circuitry	Advantages	Disadvantages
Parallel	 $R = Z_O$	Waveform at receiver is almost perfect image of input Bipolar/Advanced CMOS No added T_{PD}	High power dissipation $Z_O \geq 100\Omega$, else D.C. current limit
Thevenin	 $R = 2Z_O$	Good overshoot and undershoot suppression Bipolar or Bipolar/CMOS systems No added T_{PD}	High power dissipation

Figure 2-10(a). Termination Techniques

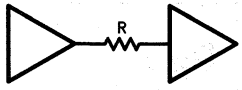
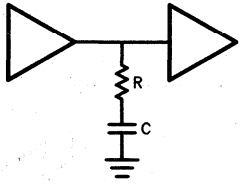
Name	Circuitry	Advantages	Disadvantages
Series	 $R = Z_O - Z_{OUT.}$	Low power consumption CMOS—CMOS Systems Easy to adjust signal amplitude to match switching threshold	Added T_{PD}
A.C.	 $R = Z_O$, C = 200 pF–500 pF	Low—medium power dissipation (capacitor blocks D.C. coupling of signal) No added delays High-speed CMOS families	Two added components

Figure 2-10(b). Termination Techniques

5

Series termination, shown in Figure 2-10(b), is a very easy technique of matching impedance. It only requires on resistor and very little additional power is required. In addition the resistor value can be selected to provide constructive or destructive reflections and thus alter the signal amplitude to match the switching threshold. The major disadvantage of this technique is the added delay it introduces.

The fourth technique is A.C. termination, shown in Figure 2-10(b). It requires a small amount of additional power, this is decreased over parallel termination by the introduction of the capacitor, and adds no extra delay to the path. The major disadvantage is that it requires two extra components.

After examing the systems needs and selecting a termination technique, the impedance values determined earlier, Z_O and Z_L, can be used to determine the component values to implement the termination. These values should be seen as a starting point and may be altered to remove a specific problem experienced on a signal or to bias signals in an appropriate fashion.

DAISY CHAINING

Another technique of minimizing reflections is to daisy-chain signals, shown in Figure 2-11. This means to run a single trace from a source and to distribute the loads along this trace. The alternative is to run multiple traces from the source to each load. Each trace will have reflections of its own and these will be transmitted down the other traces once they have returned to the source. To manage such a system separate termination would be required for each branch. To eliminate these multiple terminators from T-connections, high frequency designs are routed as daisy chains.

Because each gate provides its own impedance load along the chain, it is necessary to distribute these loads evenly along the length of the chain. Hence, the impedance along the chain will change in a series of steps and is easier to match. The overall speed of this line is faster and predictable. Also all loads should be placed at equal distances (regular intervals).

90 DEGREE ANGLES

Eliminating 90° angles also minimizes reflections. It is much more desirable to use 45° or 135° angles as shown in Figure 2-12.

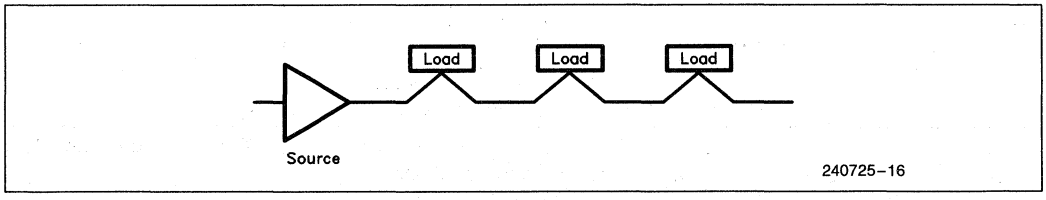


Figure 2-11. Daisy Chaining

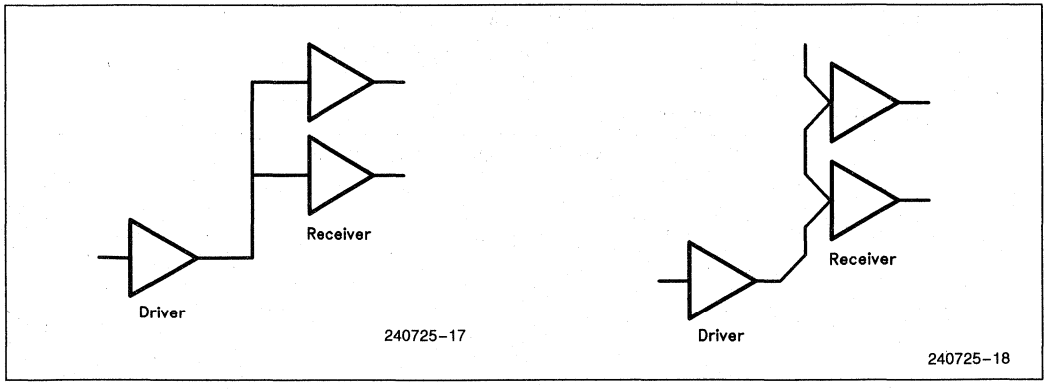


Figure 2-12. Avoiding 90 Degree Angles

VIAS (FEED THROUGH CONNECTIONS)

Another impedance source that degrades high frequency circuit performance is the via. Expert layout techniques can reduce vias to avoid reflection sites on PCBs.

Following these guidelines will not guarantee elimination of all reflections, but they will minimize the number and size.

2.4 Cross Talk

Cross talk is another negative effect of transmission lines. It is a problem at high frequencies because, as operating frequency increase, the signal wavelength become comparable to the length of the interconnections on the PC board. In general, interference such as cross talk, occurs when electrical activity in one conductor causes a transient voltage to appear in another conductor. Main factors that increase interference in any circuit are:

1. Variation of current and voltage in the lines causes frequency interference. This interence increases with increase in frequency.
2. Coupling occurs when conductors are in close proximity.

Cross talk is the phenomenom of a signal in one trace producing a similar signal in an adjacent trace. It may not be a carbon copy of the original signal. It may only be occasional noise that corrupts the integrity of the second signal. The easiest way to minimize crosstalk is to eliminate or at least minimize the number of parallel traces. Parallel traces can be on a single layer or on adjacent signal layers.

There are three ways that parallel traces can couple and thereby produce a signal or at least influence the signal on a second trace. These methods of coupling are inductive, radiative, and capacitive. Inductive coupling is where the two traces act as inductors. The field produced by a signal in one trace induces a current in the second trace. Radiative coupling occurs when the two parallel traces act as a dipole, an antenna. One radiates a signal and the other receives it, thus corupting the signal already present on the trace. The final method is capacitive coupling. Two parallel traces separated by a dielectric act as a capacitor. If both traces are in a high state and one transitions to a low. The capacitor will try to maintain the high and thus cause a slow transition time on the second trace. These effects can be minimized by reducing the number of parallel traces.

HOW TO MINIMIZE

When laying out a board for an high speed 386 DX based system, several guidelines should be followed to minimize crosstalk. Some of them are as follows:

1. To reduce crosstalk, it is necesary to minimize the common impedance paths.
2. Run a ground line between two adjacent lines. The lines should be grounded at both ends.
3. Seperate the address and data busses by a ground line. This technique may however be expensive due to large number of address and data lines.
4. Remove closed loop signal paths which create inductive noise.
5. Capacitive coupling can be reduced by reducing the number of parellel traces. Parallel traces can be minimized by insuring that signals on adjacent signal layers run orthogonal, perpendicular. Ground planes or traces can be inserted to provide shielding. A ground plane between signal layers eliminates any coupling that could occur. On a single trace, a ground trace can be run between traces to prevent coupling.

In some instances it is necessary to run traces parallel to each other. In these cases try to make the distance as short as possible and choose signals in which the transition time is not as critical so that the coupling effects do not produce problems. In addition the coupling can be minimized by increasing the spacing between parallel traces.

2.5 Skew

Skew is another effect of transmission lines. This is very important in a synchronous system. Long traces add propagation delay. A longer trace or a load placed further down a trace will experience more delay than a short trace or loads very close to the source. This must be taken into account when doing the worst case timing analysis. In a system where events must occur synchronous to a clock signal, it is important to make sure the signal is available to all input a sufficient amount of time prior to the corresponding clock edge. When performing the component placement this is one of the considerations that must be accounted for.

These guidelines have always been recommended for board design; however, they are much more important at higher frequencies. At the slower frequencies designers could ignore these practices occassionally and not experience difficulties. This is not the case at higher frequencies.

2.6 DC Loading

To maintain proper logic levels, all digital signal outputs have a maximum load, they are capable of driving. DC loading is the constant current required by an input in either the high or the low state. It limits the ability of a device driving the bus to maintain proper logic levels. For a 386 DX based system, a careful analysis must be performed to ensure that in a worst case situation no loading limits are exceeded. Even if a bus is loaded slightly beyond its worst case limit, it might cause problems if a batch of parts whose input loading is close to maximum is encountered. Proper logic level will then fail to be maintained and unreliable operation may result. Marginal loading problems are particularly insidious, since the effect is often erratic operation and non repetitive errors that are extremely difficult to track down. For both the high and low logic levels, the sum of the currents required by all the inputs and the leakage currents of all outputs (drivers) on the bus must be added together. This sum must be less than the output capability of the weakest driver. Since the 386 DX is a CHMOS device having negligible dc loading, the main contributors to dc loading will be the TTL devices.

2.7 AC Loading

The AC or capacitive loading is caused by the input capacitance of each device and limits the speed at which a device driving a bus signal can change the state from high to low or low to high. Designers of microprocessor systems have traditionally calculated load capacitance of their systems by determining the number of devices and their individual capacitance loading attached to a signal plus the amount of trace capacitance. Typically, the trace capacitance was a set "lumped" number of pf (i.e. 2 pf to 3 pf per inch) when it is thought of at all. This lumped method is a general rule-of-thumb which generates a good first pass approximation. For low frequency designs, the lumped method works since system and component margins are large enough to cover any minor differences due to the approximation.

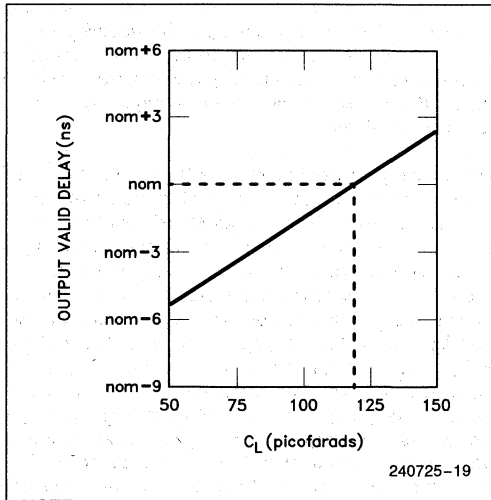
For high frequency designs, the component and system margins are no longer available to the designer. With less than 1 ns of margin, even the amount of trace capacitance can make a circuit path critical.

A more accurate calculation of capacitive loading can be derived by modeling the device loads and system traces as a series of Transmission Lines Theory. Transmission Line Theory provides a more accurate picture of system loading in high frequency systems. In addition, it allows new factors such as inductance and the effect of reflections upon the quality of the signal waveform to be factored into consideration.

2.8 Derating Curve and Its Effects:

A derating curve is a graph that plots the output buffer against the capacitive load. The curve is used to analyze a signal delay without necessitating a simulation every time the processor's loading changes. This graph assumes the lumped capacitance model to calculate the total capacitance. The delay in the graph should be added to the specified AC timing value for the device that is driving the load. The derating curve is different for different devices because each device has different output buffers.

A derating curve is generated by tying the chip's output buffers to a range of capacitors. The voltage and resistance values chosen for the output buffers are at the highest specified temperature and are rising (worst case) values. The value of the capacitors centres around the AC timing values for the chip. For 33 MHz and above, this is 50 pF. Since the AC timing specifications are measured for a signal reaching 1.5 V. A curve is then drawn from kthe range of time and capacitance values, with 50 pF representing the average and with nominal or zero derating. These curves are valid only for 50 pF–150 pF load range. Beyond this range the output buffers are not characterized. The the derating curve for the 386 DX are shown in 2-13. These curves use the lumped capacitance model for circuit capacitance measurements and must be modified slightly when doing worst-case calculations that involve transmission line effects.



NOTE:
This graph will not be linear outside of the C_L range shown.

Figure 2-13. Typical Output Valid Delay Versus Load Capacitance at Maximum Operating Temperature (C_L = 120 pF)

2.9 High Speed Clock Circuits

For performance at high frequencies, the clock signal (CLK2) for the 386 DX CPU must be free of noise and within the specifcations listed in the 386 DX CPU data sheet. Achieving the proper clock routing around a 33 MHz printed circuit board is delicate because a myriad of problems, some of them subtle, can arise design guidelines are not followed. For example, fast clock edges cause reflections from high impedance terminations. These reflections can cause significant signal degradation in systems operating at 33 MHz clock rates. This section covers some design guidelines which should be observed to properly lay out the clock lines for efficient 386 DX operation.

- Since the rise/fall time of the clock signal is typically in the range of 2-4 ns, the reflections at this speed could result in undesirable noise and unacceptable signal degradation. The degree of reflections depends on the impedance of the traces of the clock connections. These reflections can be optimized by terminating the CLK2 output with proper terminations and by keeping length of the traces as short as possible. The preferred method is to connect all of the loads via a single trace as shown in Figure 2-14, thus avoiding the extra stubs associated with each load. The loads should be as close to one another as possible. Multiple clock sources should be for distributed loads.
- A less desirable method is the star connection layout in which the clock traces branch to the load as closely as posssible (Figure 2-15). In this layout, the stubs should be kept as short as possible. The maximum allowable length of the traces depends upon the frequency and the total fanout, but the length of all the traces in the star connection should be equal. Lengths of less than one inch are recommended. In this method the CLK2 signal is terminated by a series resistor. The resistor value is calculated by measuring the total capacitive load on the CLK2 signal and referring to Figure 2-16. If the total capacitive load is less than 80 pF, the user should add capacitors to make up the diference. Because of the high frequency of CLK2, the terminating resistor must have low inductance; carbon resistors are recommended.
- Use an oscilloscope to compare the CLK2 waveform with those in Figure 2-17.

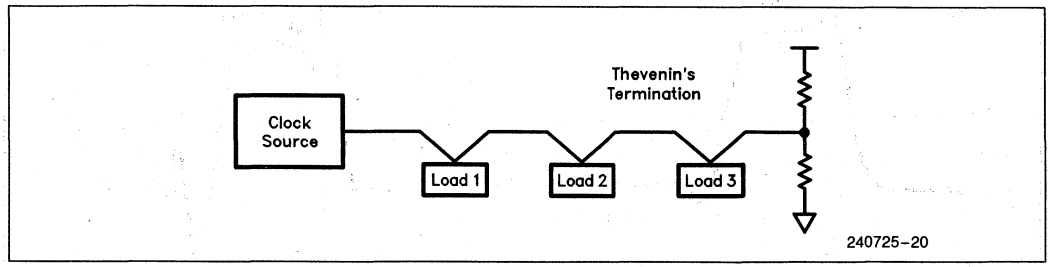


Figure 2-14. Clock Routing

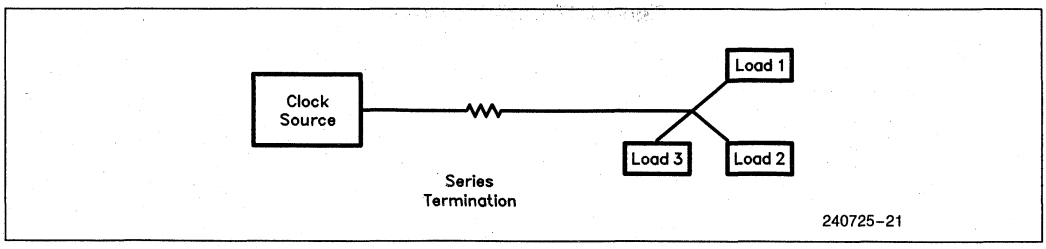
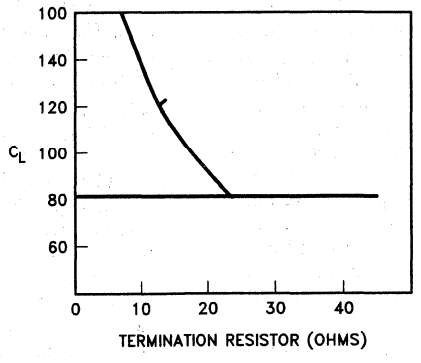


Figure 2-15. Star Connection



- $C_L = C_{IN}$ (386) + C_{IN} (387) + C_{IN} (PALs) + ... + C_{BOARD}.
  C_{BOARD} is calculated from layout and board parameters; thickness, dielectric constant, distance to ground/V_{CC} planes.
- Termination resistor must be low inductance type. Recommend carbon filled type.

Figure 2-16. CLK2 Series Termination

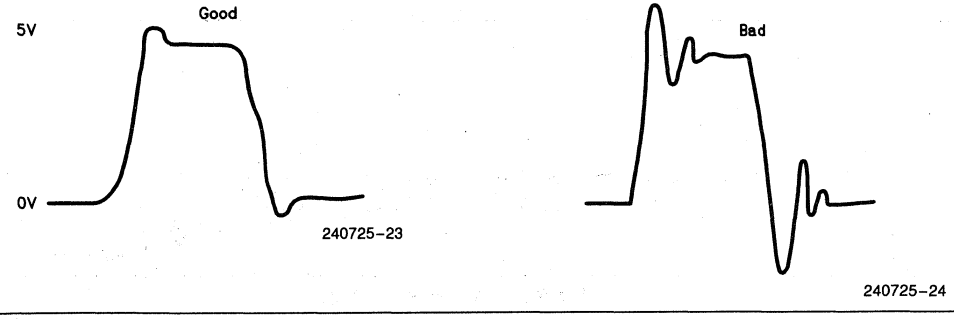


Figure 2-17. CLK2 Waveforms

SECTION III. DESIGN EXAMPLE

At higher processor speeds the window of time available to perform specific tasks become very small. This window can be equated to multiples of the CLK2 period. Within this time signals must be supplied from a source and reach a destination in time to meet any set-up requirements. At 16 MHz the CLK2 period is 31 ns. At 33 MHz it shrinks to half this value, 15 ns. The longer time allowed the use of slower logic families and the delays associated with longer traces. As the window decreases system designers have to practice more care in the selection of logic families and in the choices made for component placement and signal routing on PCBs. This section attempts to list the signal paths whose worst case timing analysis results in very small margins and therefore require closer attention from designers to guarantee that all a. c. timing specifications are met.

This section also includes a sample design based on 33 MHz version of the 386 DX. It should not be taken as a recommended design. The circuit is used only to highlight the design considerations for high speed systems.

3.1 System Architecture

Figure 3.1 shows the system block diagram. It has four major subsystems.

1) CPU subsystem
2) DRAM subsystem
3) Cache subsystem
4) ROM and I/O subsystem

The system has 1 megabytes of Page-Mode DRAMS (60 ns RAS access time), 128 kilobytes of EPROMS (200 ns access time), an 8259A-2, and an 82510. The cache subsystem is optional. Schematics and PAL codes are given in appendix A and B respectively.

3.2 CPU Subsystem

The CPU subsystem consists of the 386 DX microprocessor, a clock and reset circuitry, and bus control logic. Clean and proper clock is very important in the designs at high frequencies.

RESET STATE MACHINE

This state machine is used to generate three control signals, namely RESET, REFREQ and CLK. The CLK signal is half of the CPU clock, CLK2 and is used mainly in I/O and EPROM subsystem.

RESET is generated through the input from RESET triggering circuitry (as shown in the CPU schematic). The min RESET Setup and Hold time for operation at 33 MHz are 5 ns and 2 ns respectively.

A 61.44 KHz clock is used to produce a synchronous refresh request (REFREQ) signal for the DRAM controller, which employ a transparent, distributed, DRAM refresh technique that allows the processor and cache to run while the refresh cycle is in progress.

3.3 DRAM Subsystem

An non-interleaved DRAM system is used in the sample board, which simplifies the design. Since the board provide caching, the performance of DRAM subsystem is outweighed by the simplicity and economy of the design. It employs a transparent, distributed, DRAM refresh technique which allows the processor and cache to run while the refresh cycle is in progress. It uses the 3-state capability of the 16R8-7 and the 74ACT258 to multiplex the refresh address. A further consideration is the choice of DRAM devices. If one uses a memory device such as the AAA2801 (which supports a CAS# before RAS# refresh and provides an internal refresh counter) further simplifications can be made in both the circuitry and the control logic.

DRAM CONTROL STATE MACHINE

The state machine is implemented with three 16R8-type E-speed PALs (see page 4 of the schematics). E-speed PALs must be used since the CLK2 frequency, 66.67 MHz, is higher than the maximum clock frequency of the D-speed PALs.

In order to generate DRAM control signals with smallest delay from the CLK2 edges, all state machines are implemented as Moore machines. The state machines flip-flops generate most of the DRAM control signals directly. This is an expensive design approach in terms of hardware but allows signal timings and skews to be fine tuned.

DRAM CYCLES—NO CACHE CONFIGURATION

Pages C-1 through C-4 show examples of DRAM cycles. In order to hide the DRAM page hit-or-miss decision time, the DRAM controller always tries to put the 386 DX in pipelined mode. The first read cycle requires only two wait states since RAS# has been precharged (see page C-1). The second cycle takes only two clock cycles. The second cycle is a pipelined, page-hit read cycle, which is the best case. The third cycle is a pipelined, page-hit write cycle. This cycle requires one wait state. DRAMs capture data at the falling edge of CAS# during Early Write cycles. The 386 DX drives

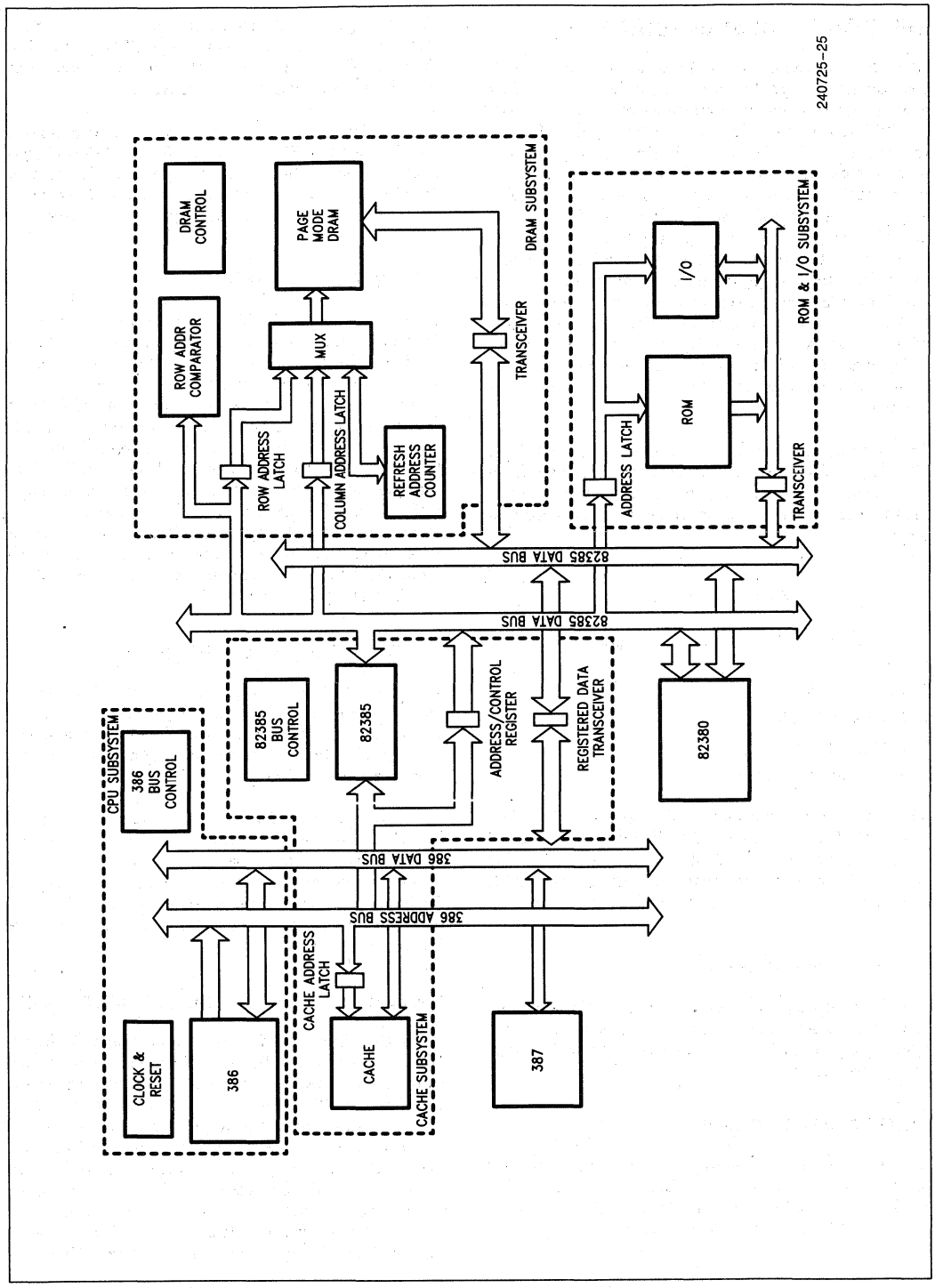


Figure 3-1. Block Diagram

valid write data at the rising edge in the middle of Tip (edge C) with a max prop delay of 24 ns (T12 max). This means that the CAS# is generated after the rising edge in the middle of the second T2p (edge A). CAS# is, therefore, generated at the end of RAS# hold time with respect to CAS# (if the next cycle is a page miss, RAS# will go inactive at the end of the current write cycle), and so on.

The fifth cycle is a page miss, which is actually detected at the end of the fourth cycle (page C-2). Since the DRAM controller must wait for minimum RAS# precharge time, the fifth cycle requires three wait states. The sixth cycle is also a page miss. This cycle, however, requires only two wait states because the miss was detected early enough in the previous cycle to have RAS# precharged by the end of the T1p. If the seventh cycle is another page miss, it will require three wait states.

The eighth cycle is ended with T2i. Consequently, the ninth cycle must wait for minimum RAS# precharge time and requires three wait states.

A DRAM refresh cycle is shown on page C-4. The DRAM address multiplexer output is disabled, and the refresh address counter output is enabled. The cycle does a RAS# only refresh cycle where only RAS# is asserted with a proper refresh address. After the refresh cycle is completed, a read cycle which has been suspended due to the refresh is resumed.

STATE DIAGRAMS

Pages B-1 through B-11 show state diagrams of the DRAM controller. The precharge state machine on page B-2 measures the required RAS# precharge time and CAS#-to-RAS# precharge time. The CAS#-READY# state machine on page B-2 implements a pin strap option of having or not having the 82385. For no cache configuration, the Cache variable must be forced low.

TIMING CALCULATIONS

Timing equations are described on pages D-1 and D-2. Their corresponding results are given on pages D-3 through D-7.

Capacitive load on the 386 DX address bus was assumed to be less than 85 pF. Capacitive load on the DRAM address bus was calculated to be less than 22 pF.

3.4 CACHE Subsystem

At 33 MHz DRAM speeds are not fast enough to design zero wait state memory systems. A cache can be used to take advantage of the higher performance available from the higher speed 386 DX microprocessors. The cache takes advantage of the faster SRAM while keeping system costs down by using the cheaper but slower DRAMs.

Details of the cache subsystem are shown on Figure 3.2 and 3.3. The 82385 address and data busses are interfaced to the 386 DX address and data busses via 74AS574s and 74AS646s. Static RAMs (20 ns access time) are used for the cache memory.

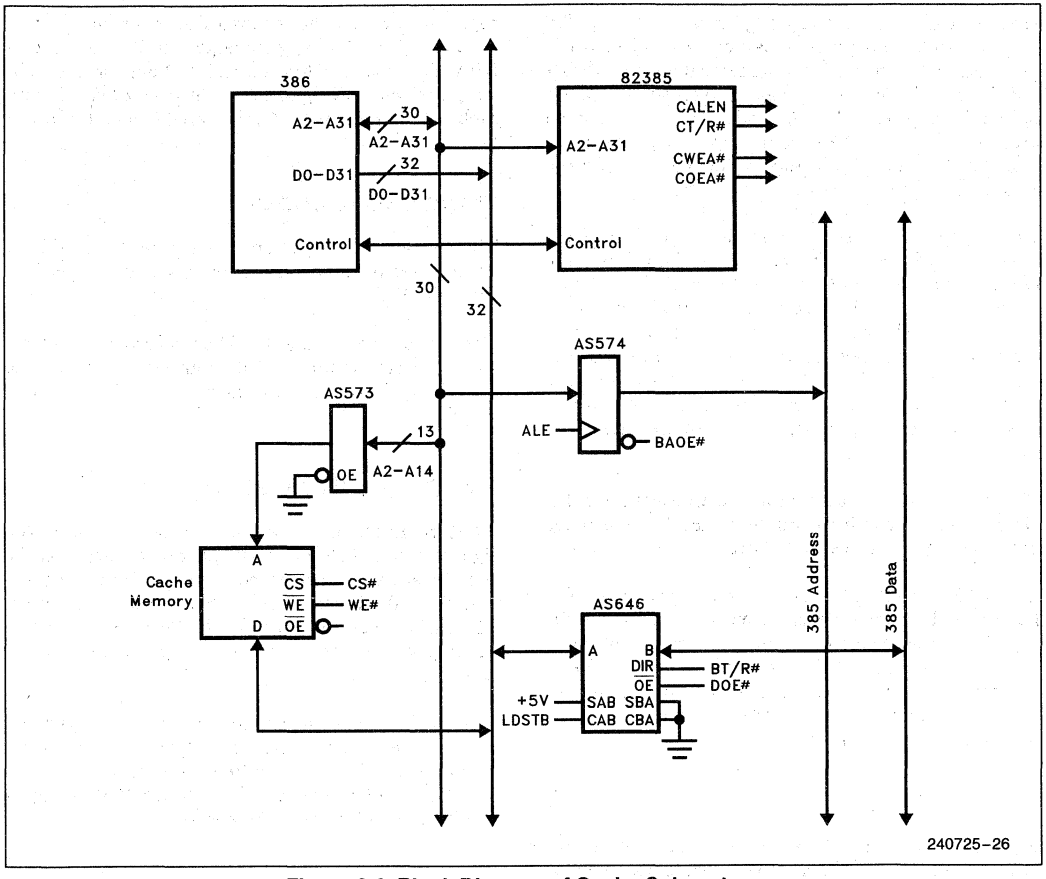


Figure 3-2. Block Diagram of Cache Subsystem

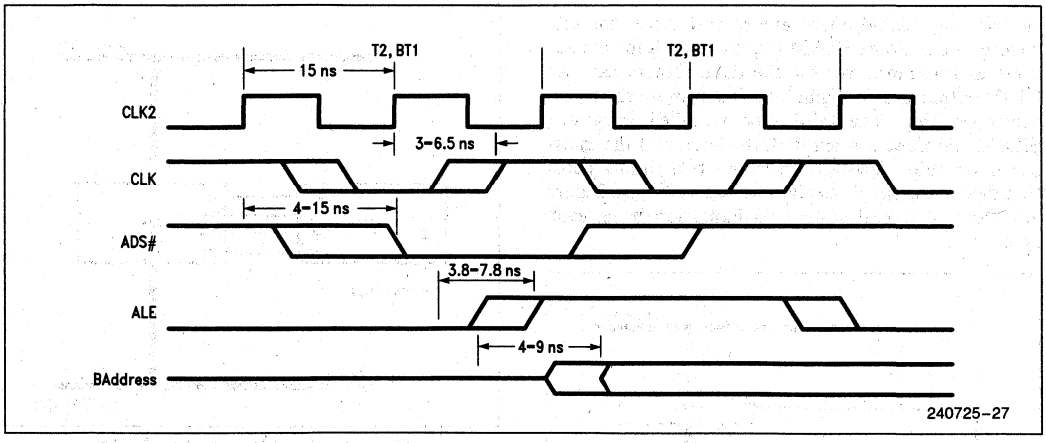


Figure 3-3. Address Valid Delay for Cache Subsystem

5

In selecting SRAM there are several types one can choose to use. Some SRAM require a latch for the address and a transceiver for the data. Others have an OE#, output enable, signal and incorporate the transceiver on chip. The third type is called integrated SRAM and these contain both the latch and the transceiver on chip. However, there are two timing paths that dictate the speed selection within each type. Figure 3.4 shows a typical system configuration using each type.

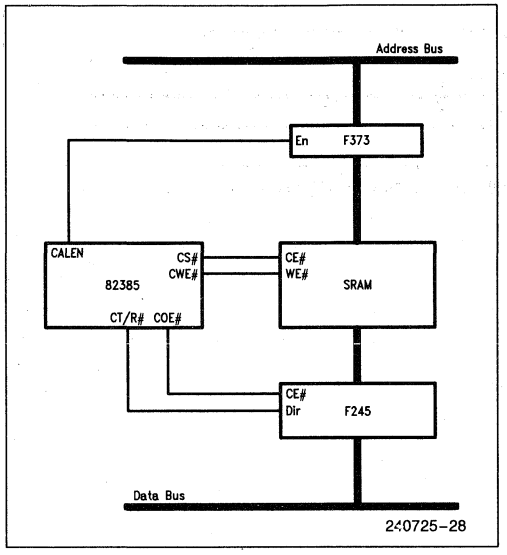


Figure 3.4(a) SRAM w/o OE#

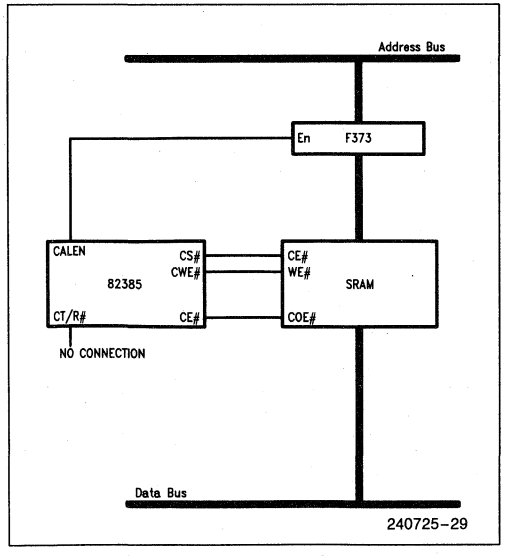


Figure 3.4(b) SRAM with OE# Control

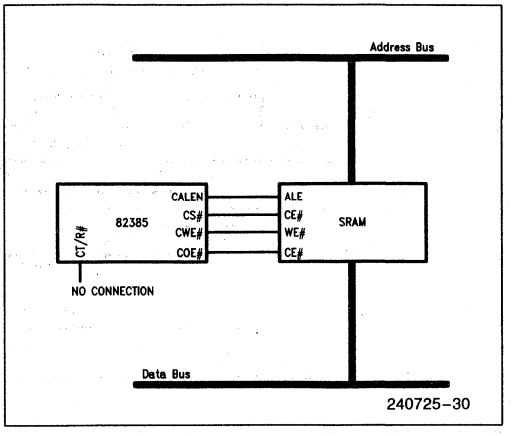


Figure 3-4. (c) Integrated SRAM

The critical times for the SRAM are the SRAM OE# to data delay and the SRAM address to data delay. The following analysis applies to SRAMs with an OE# signal as shown in Figure 3.4b. First examine the path of OE# to data. This path must be completed within 2 CLK periods. The COE# signal from the 385 Cache Controller must be valid and the SRAM must drive data onto the data bus so that the data setup time of the 386 DX CPU is met.

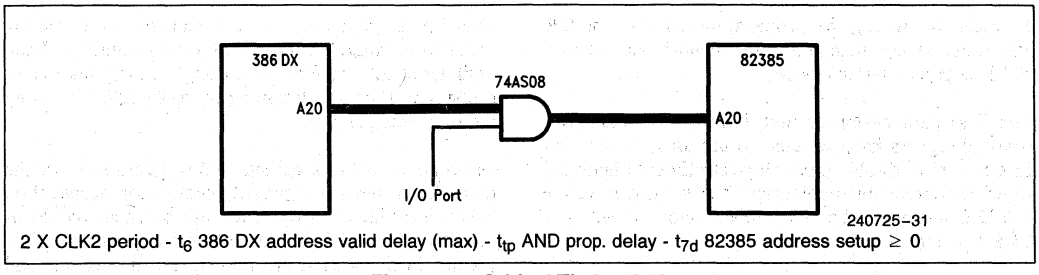
2 X CLK2 period - t_{25b} 82385 COE# valid delay (max) - SRAM access time (OE# to data) - t_{21} 386 DX data setup $\geq$ 0

Using the specified values from the data sheets reveals that the SRAM must have an OE# to data delay of 10ns or less. The other path is for the address to become available and data to reach the 386 DX CPU. This path has 4 CLK2 periods. The 385 Cache Controller must supply the CALEN signal to pass the address to the SRAM and then the SRAM must drive the data on the data bus so that the data setup time is met on the 386 DX CPU.

4 X CLK2 period - t_{21b} 82385 CALEN valid delay (max) - t_{pd} (x373 latch) - SRAM access time (address to data) - t_{21} 386 DX data setup $\geq$ 0

Once again using the data sheet the access time can be determined. Depending on the type of transparent latch the SRAM needs an address to data access time of 20ns or 25ns. If an F series 373 is used the faster 20ns SRAM must be used, but if an FCT373a or PCT373a is used the 25ns SRAM is sufficient.

The A_{20} path is another path with a small margin. The reason is the AND gate that many designers insert to provide 1MB wraparound of address in real mode. Figure 3.5 shows the circuit block diagram. A_{20} must leave the 386 DX and reach the 385 Cache Controller within 2 CLK2 periods.

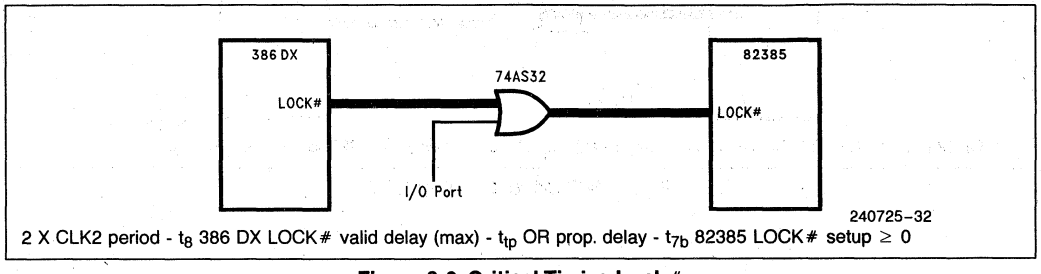


2 X CLK2 period - t_6 386 DX address valid delay (max) - t_{tp} AND prop. delay - t_{7d} 82385 address setup ≥ 0

Figure 3-5. Critical Timing A20

To meet this timing the propagation delay of the AND gate must be less than 6ns. This dictates the use of a 74AS08 gate or faster device.

Analysis of the LOCK# path also shows a small margin. The reason is the OR gate that many designers insert to disable the LOCK# signal to the 385 Cache Controller. This allows locked accesses to be cached. Figure 3.6 shows the circuit block diagram. LOCK# must leave the 386 DX and reach the 385 Cache Controller within 2 CLK2 periods.



2 X CLK2 period - t_8 386 DX LOCK# valid delay (max) - t_{tp} OR prop. delay - t_{7b} 82385 LOCK# setup ≥ 0

Figure 3-6. Critical Timing Lock#

5

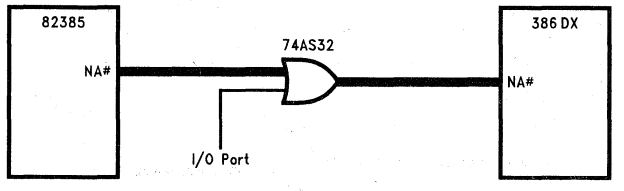
To meet this timing the propagation delay of the OR gate must be less than 6ns. This dictates the use of a 74AS32 gate or faster device.

The final path examined here is the NA# path. Recently designers have selected to use an I/O port and an OR gate to disable pipelining selectively. Figure 3.7 shows the circuit block diagram. NA# must leave the 386 DX and reach the 385 Cache Controller within 2 CLK2 periods.

Using the specified values in the appropriate data sheets results in the need for the propagation delay of the OR gate must be no greater than 5.8ns. This dictates the use of a 74AS32 gate or faster device.

This list is not meant to be exhaustive. It is merely meant to highlight a few of the critical timings. Each designer should perform a thorough timing analysis of the system they are designing to verify that all timing requirements are met.

In addition to the specified timing parameters in the data sheets, designers should account for propagation delays introduced by the trace and by capacitive loading. The propagation delay added by the trace is explained in the section on transmission line effects and supplies an equation to determine the amount of delay.



2 X CLK2 period - t_{17} 386 DX NA# valid delay (max) - t_{tp} OR prop. delay - t_{15} 82385 NA# setup $\geq$ 0

Figure 3-7. Critical Timing NA#

Another factor that becomes more important at higher frequencies is loading. DC loading and especially capacitive loading must be considered during the design stage. If the board is to be assembled and tested in stages, then the DC loads should be considered for all configurations of the board. Most termination techniques require additional current. If a board has a marginal loading situation, one is limited in one's choices of termination techniques. If a capacitive loading problem exists, the timing situations can become extremely difficult at higher frequencies. If timing is critical, do not overload the capacitance at which a device was tested. If a device is overloaded, derating must be taken into consideration.

Capacitive loading also introduces a delay on signals. Many components including the 386 DX include a capacitive derating curve in the data sheet. To use the curve in the 386 DX data sheet, the capacitive load must be calculated. This is done by summing the input capacitances of all devices driven by a given output from the 386 Microprocessor. Find this value on the X-axis of the derating curve in the data sheet and move up till the derating curve is intersected. Then move at a right angle to the left until intersecting the Y-axis. A value of nom+ or nom− something is found. This is the nominal value plus or minus some amount. The nominal value is the value found in the data sheet. Add the offset from the curve to this nominal value to get the resulting delay corresponding to the capacitive loading in the system. Note: The trace capacitance was not included in this calculation. It is accounted for in the trace propagation delay mentioned earlier.

DRAM CYCLES WITH 82385 ENABLED

When the 82385 is enabled (the CACHE variable of the state machine on page B-2 is forced High), the DRAM controller inserts one extra wait state in all read cycles. This extra time is needed to allow a cache update cycle to occur after each cache read miss cycle. During a cache update cycle, the read data from DRAMs must propagate through the 74AS646 and the 74F245 (optional) and must be ready for a SRAM write cycle with enough setup time.

Timing diagrams on pages C-5 through C-9 show cache and DRAM cycles.

TIMING CALCULATIONS

Timing equations are found on pages D-8 and D-9. Only tCAS, tRAC, tCAC, tAA, tPC, and tCAP are different in this configuration. Actual values for DRAM timings are found on page D-10.

3.5 I/O - EPROM Subsystem

A block diagram of the I/O-EPROM subsystem is shown on Figure 3.8. This subsystem has separate address and data busses. The address bus is 14 bits wide, and the data bus is 16 bits wide.

The bus controller is designed with B-speed PALs which are clocked by the CLK# signal (Figure 3.8). There are a few unique design issues in this scheme.

As shown on Figure 3.10, ADS# is now an asynchronous signal for the state machine. It is impossible for the state machine to capture valid ADS# without re-synchronization of the signal. To guarantee recognition of valid ADS#, two D flip-flop is clocked by CLK# and provides a synchronous ADS# (or Latched ADS#) which is in phase with the state machine.

The second issue is its asynchronous nature of the state machine output signal. With the state machine running almost asynchronously to CLK2 (B PALs also have a long clock-to-output propagation delay), signals generated by the state machine must be re-synchronized before they are returned to the 386 DX. Signals that go to I/O devices and EPROMs need no re-synchronization since these devices are asynchronous. Signals which require re-synchronization are BS16# and DEN#. Each rising edge of DEN# is synchronized to CLK2 by a J-K flip-flop as shown on Figure 3.9. This is important to avoid bus contention after an I/O or EPROM read-cycle. BS16# is synchronized to CLK2 by D flip-flops.

EPROM and I/O cycle timings are shown on pages C-10 through C-13. The worst case is a write cycle to the 82510 and may require as many as 14 wait states.

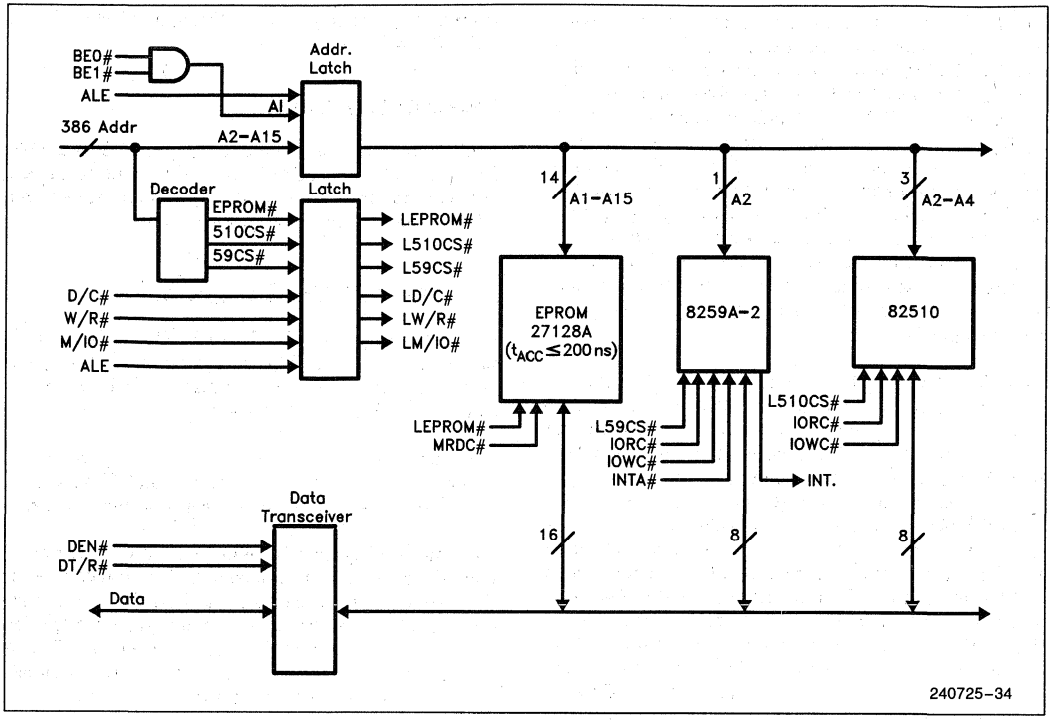
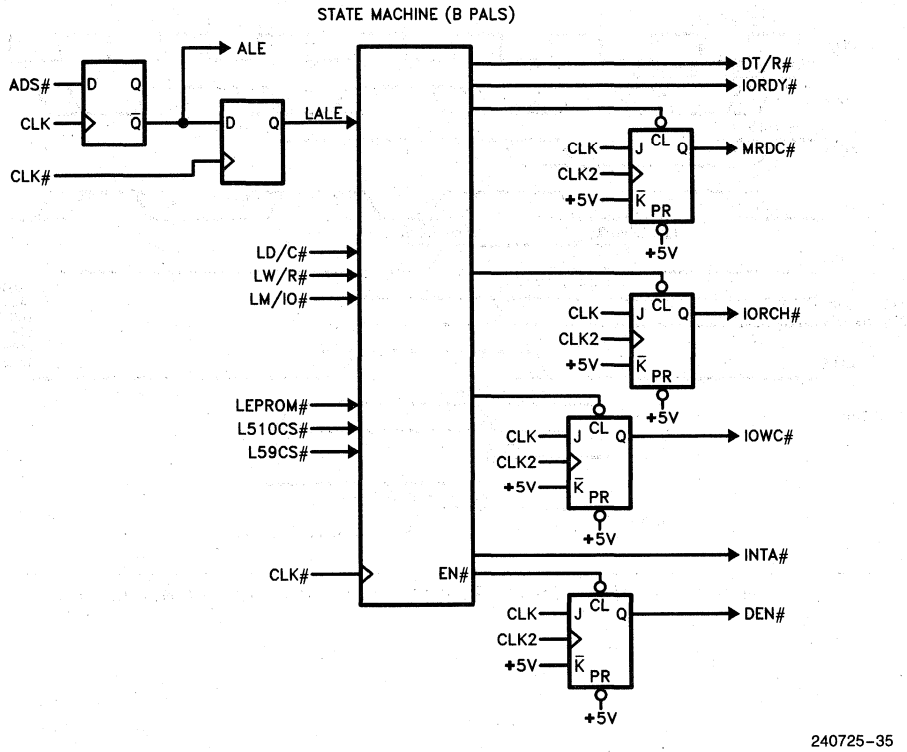


Figure 3-8. Block Diagram of I/O, EPROM Subsystem



NOTE:
Create BS16# for 386 Using DEN# and EPROM# (Synch to CLK)

Figure 3-9. Control Logic for I/O, EPROM Subsystem

5

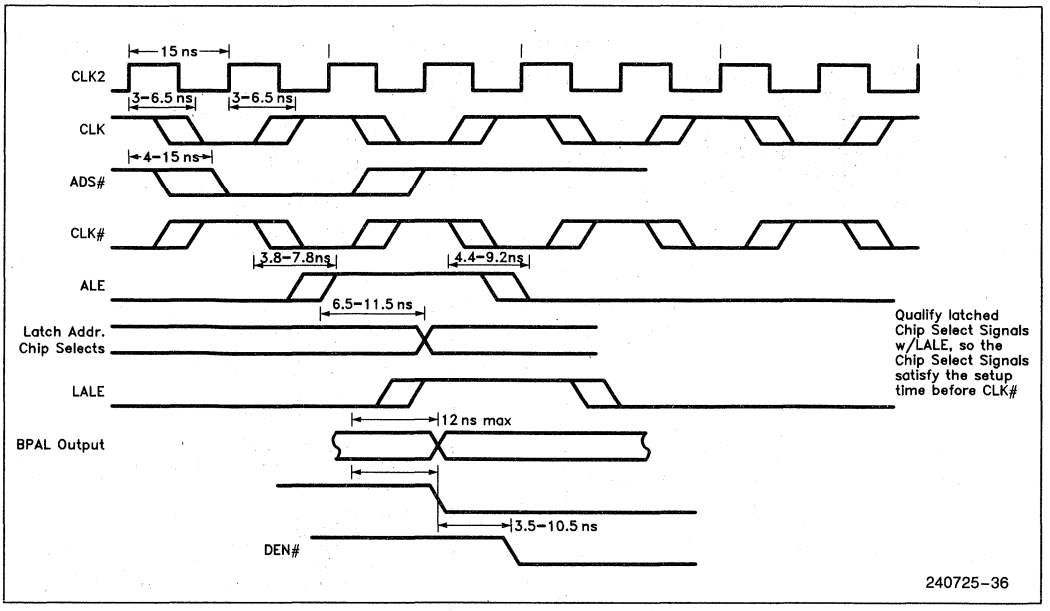
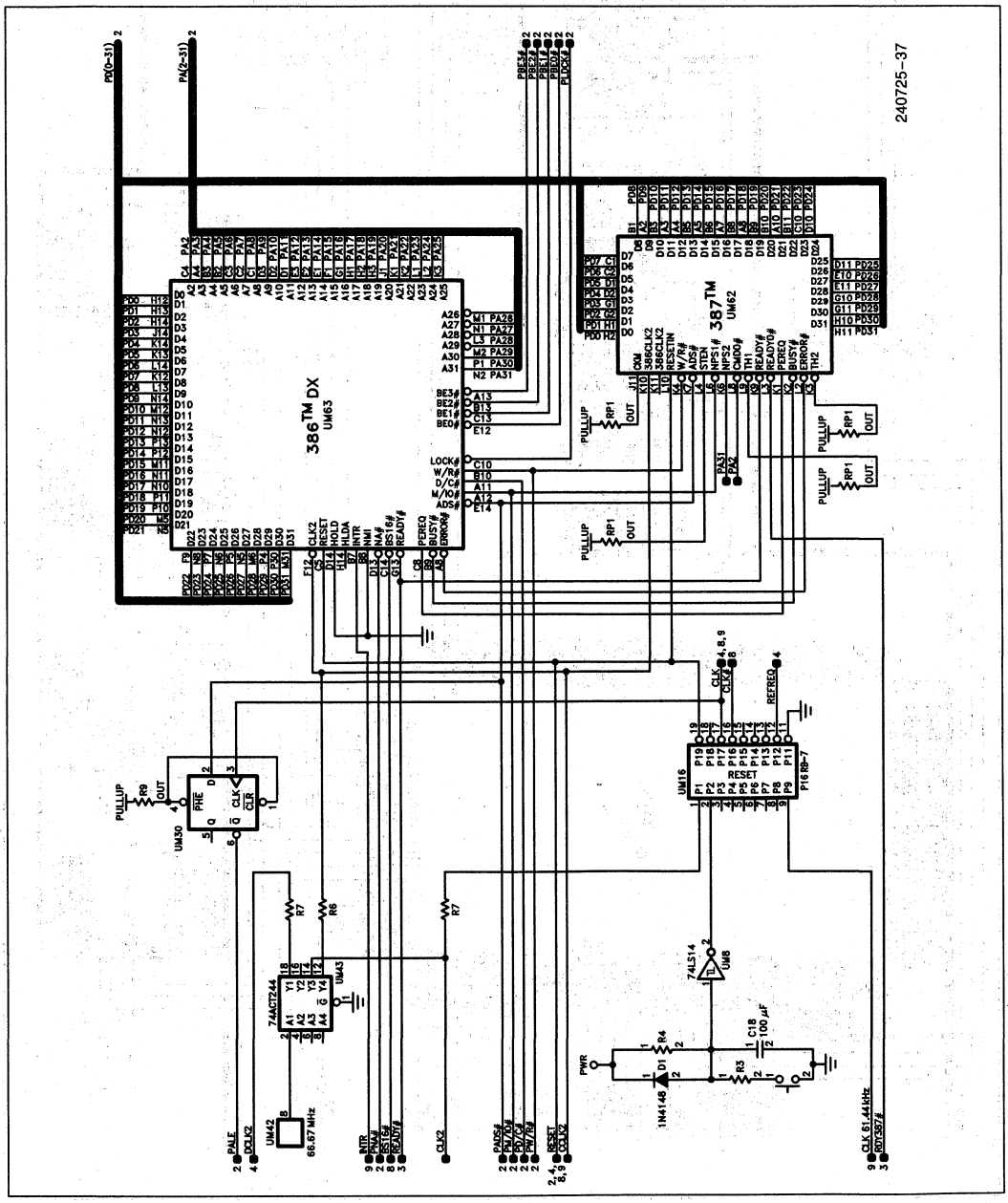
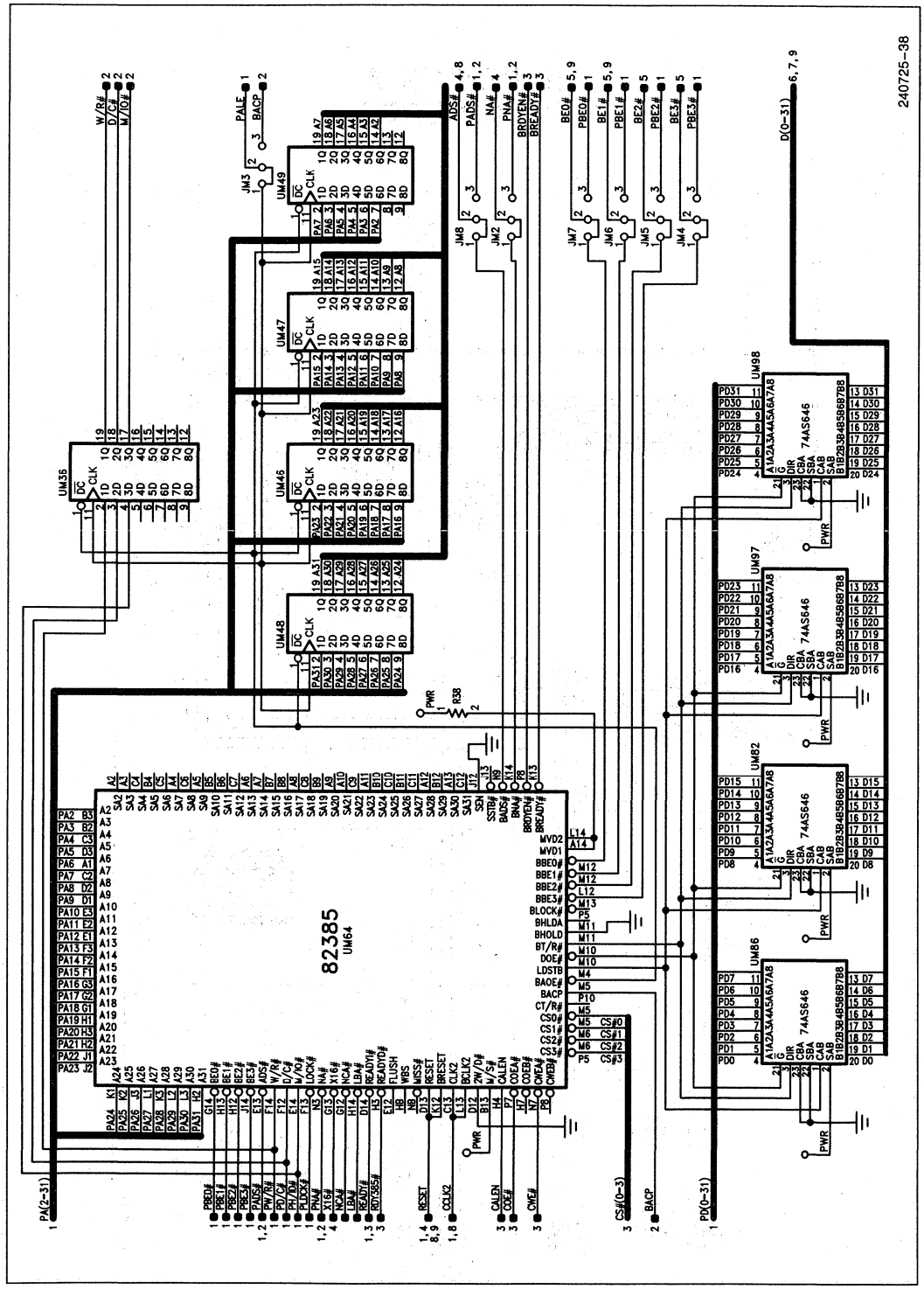
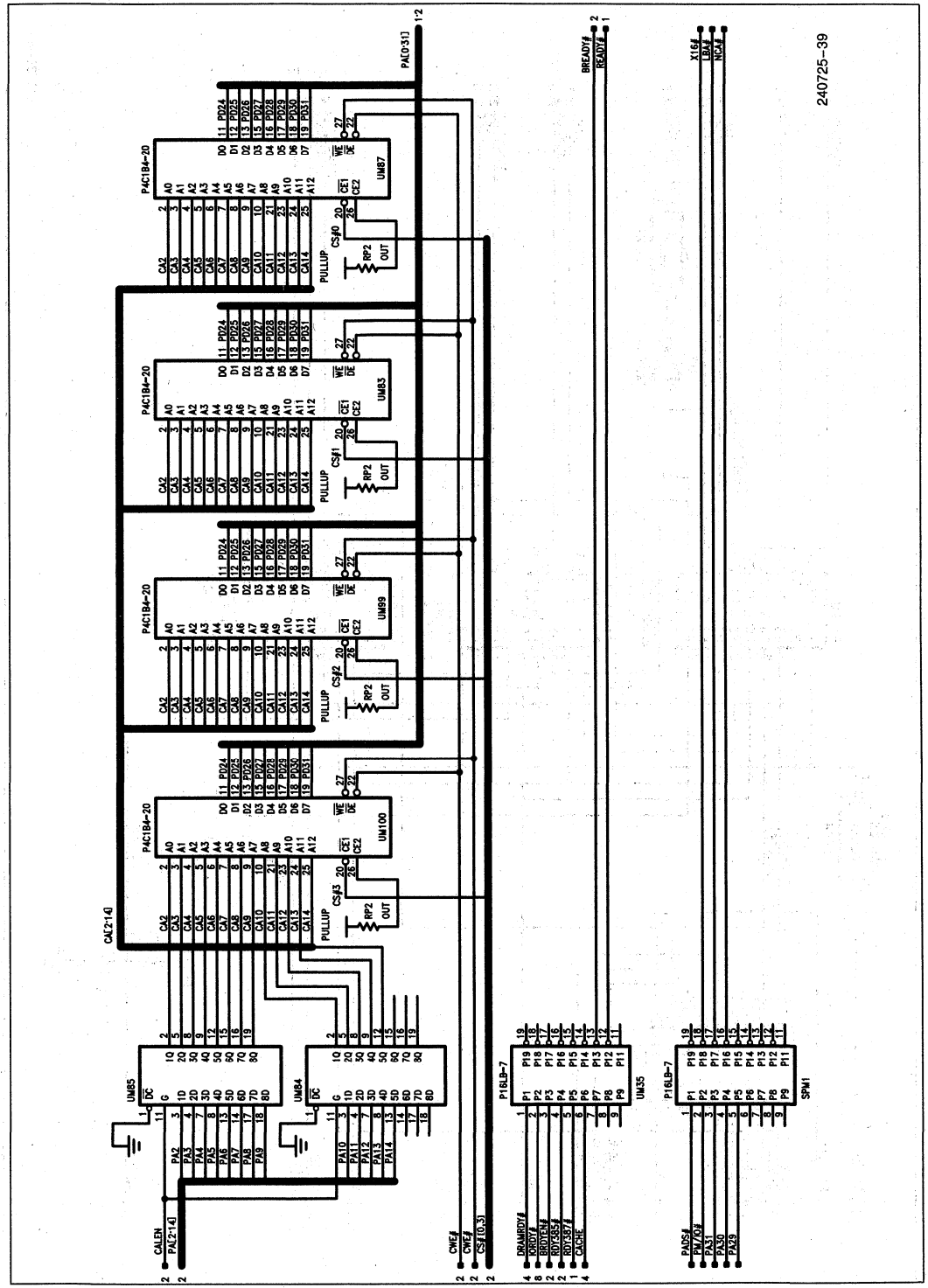


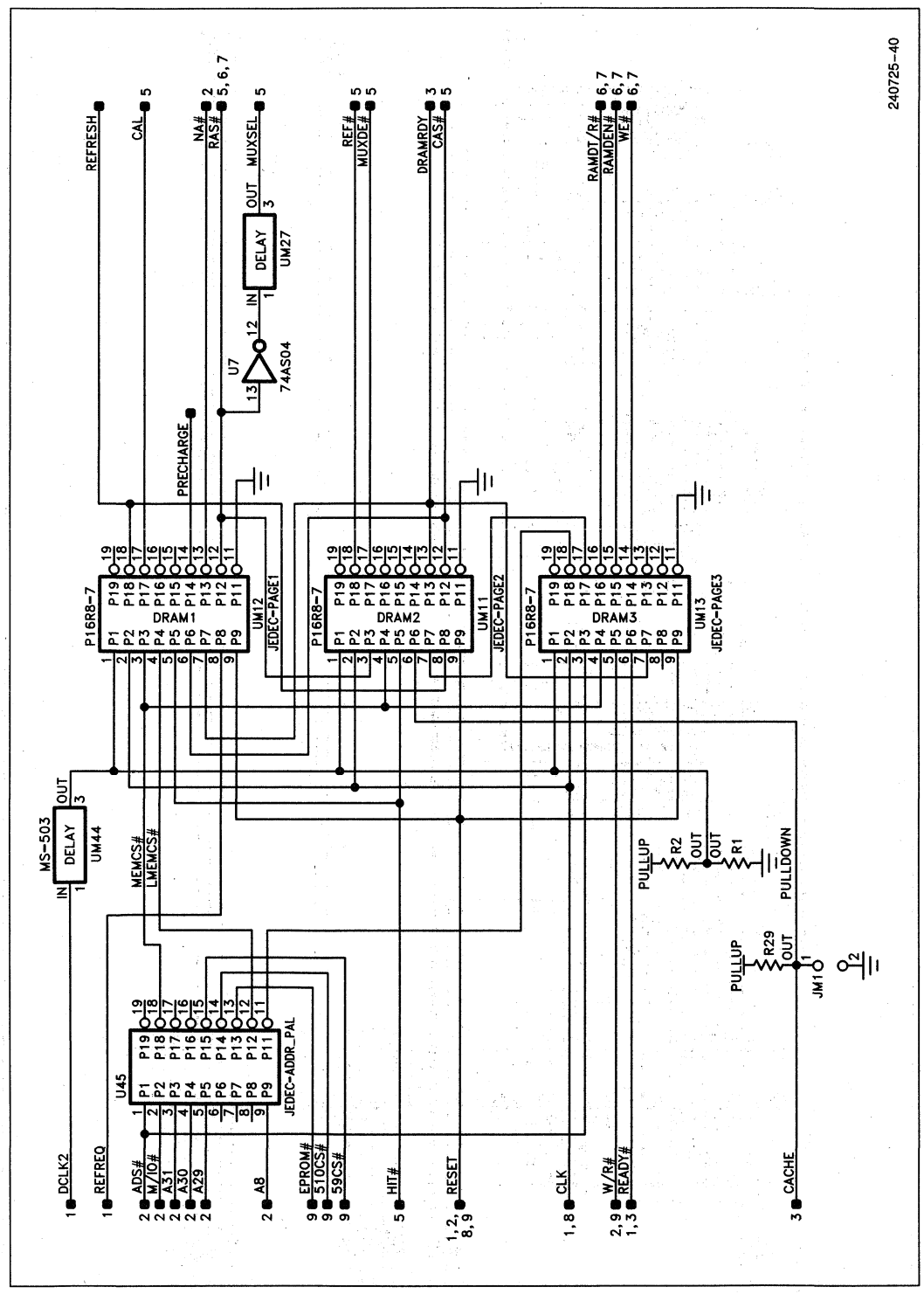
Figure 3-10. ADS# Should Be Synchronized to Guarantee Recognition

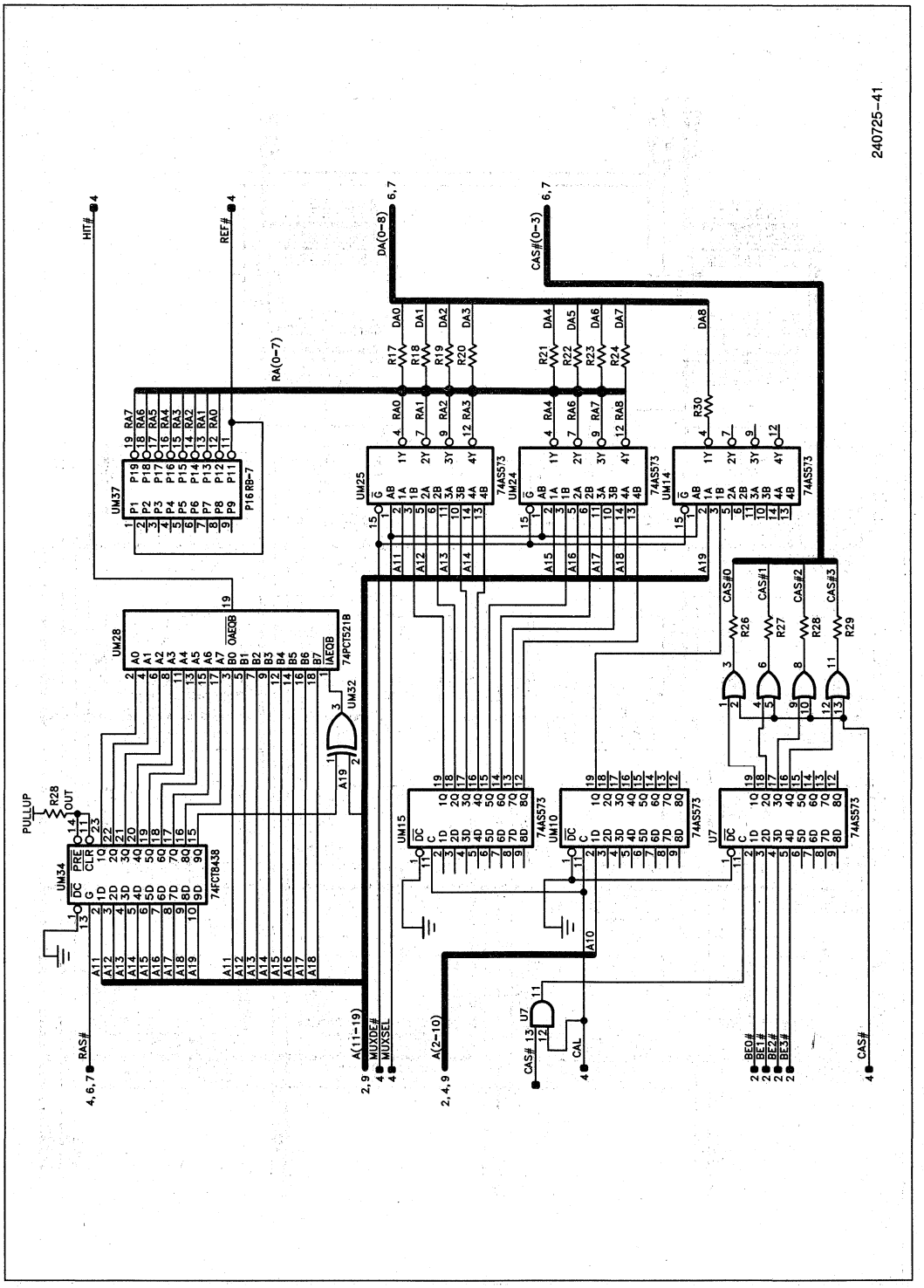
APPENDIX A
SCHEMATICS



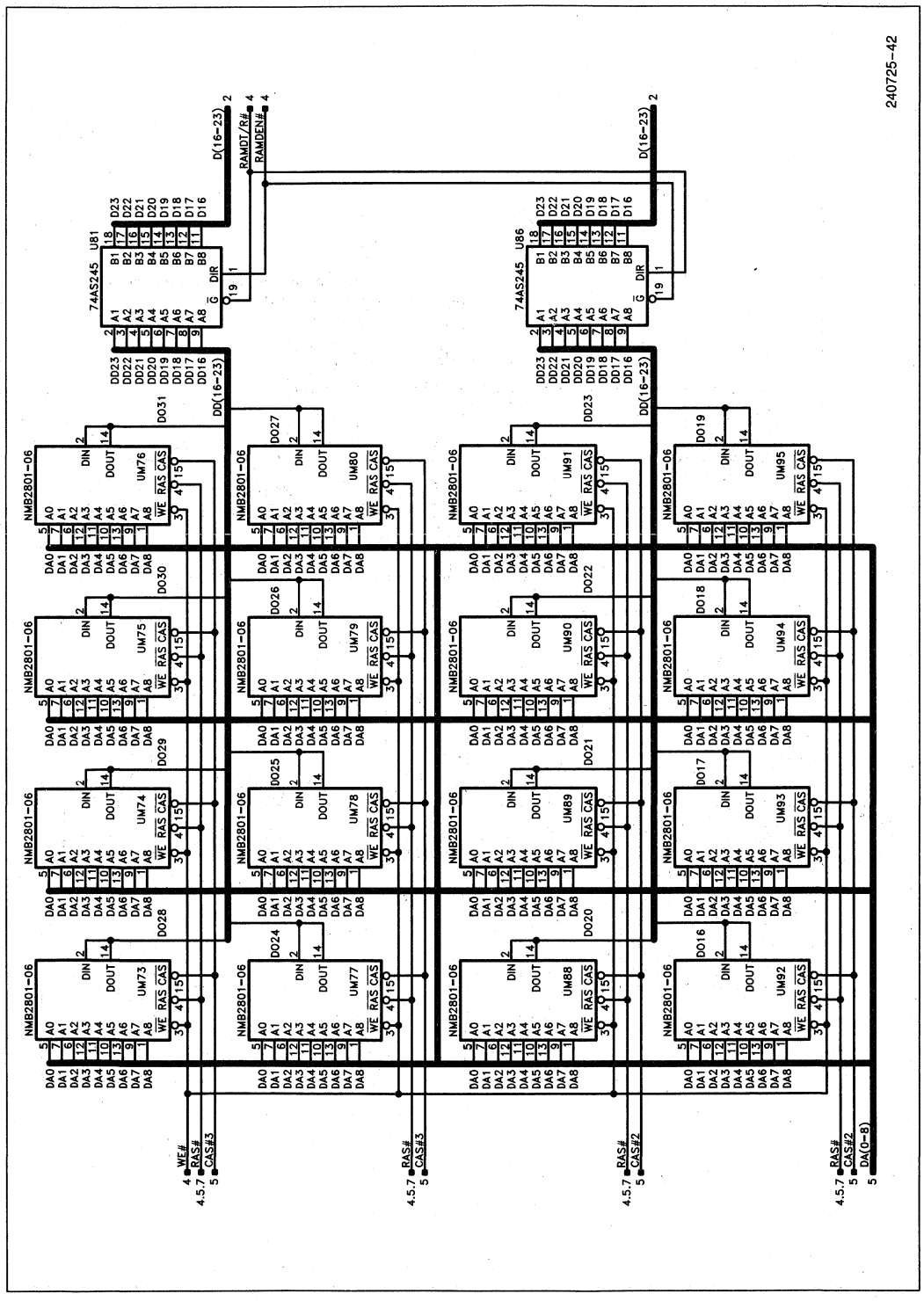

82385
UM64
UM35
UM46
UM47
UM48
UM49
UM97
UM98
UM82
UM86
74AS646
240725-38

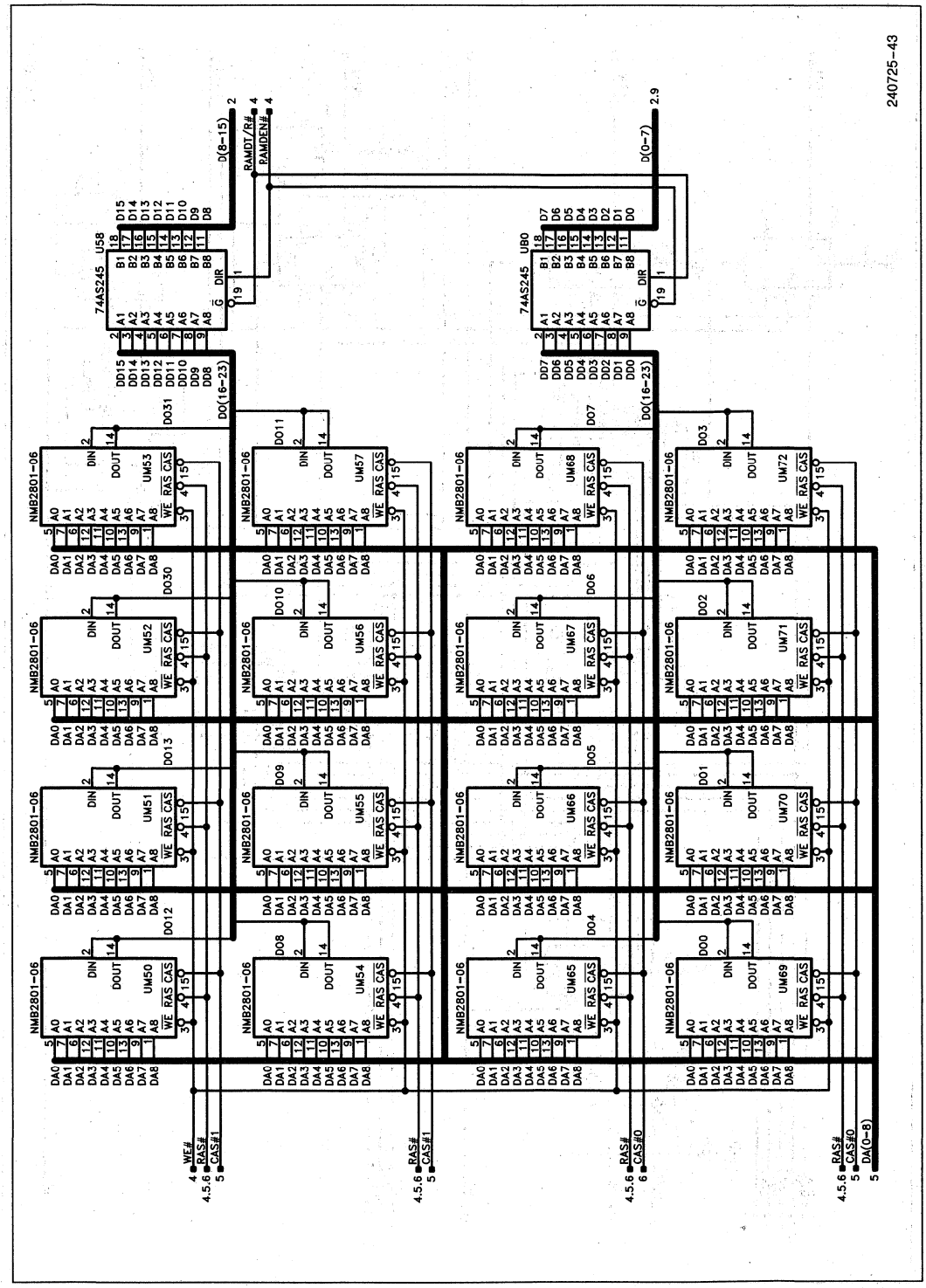

240725-39

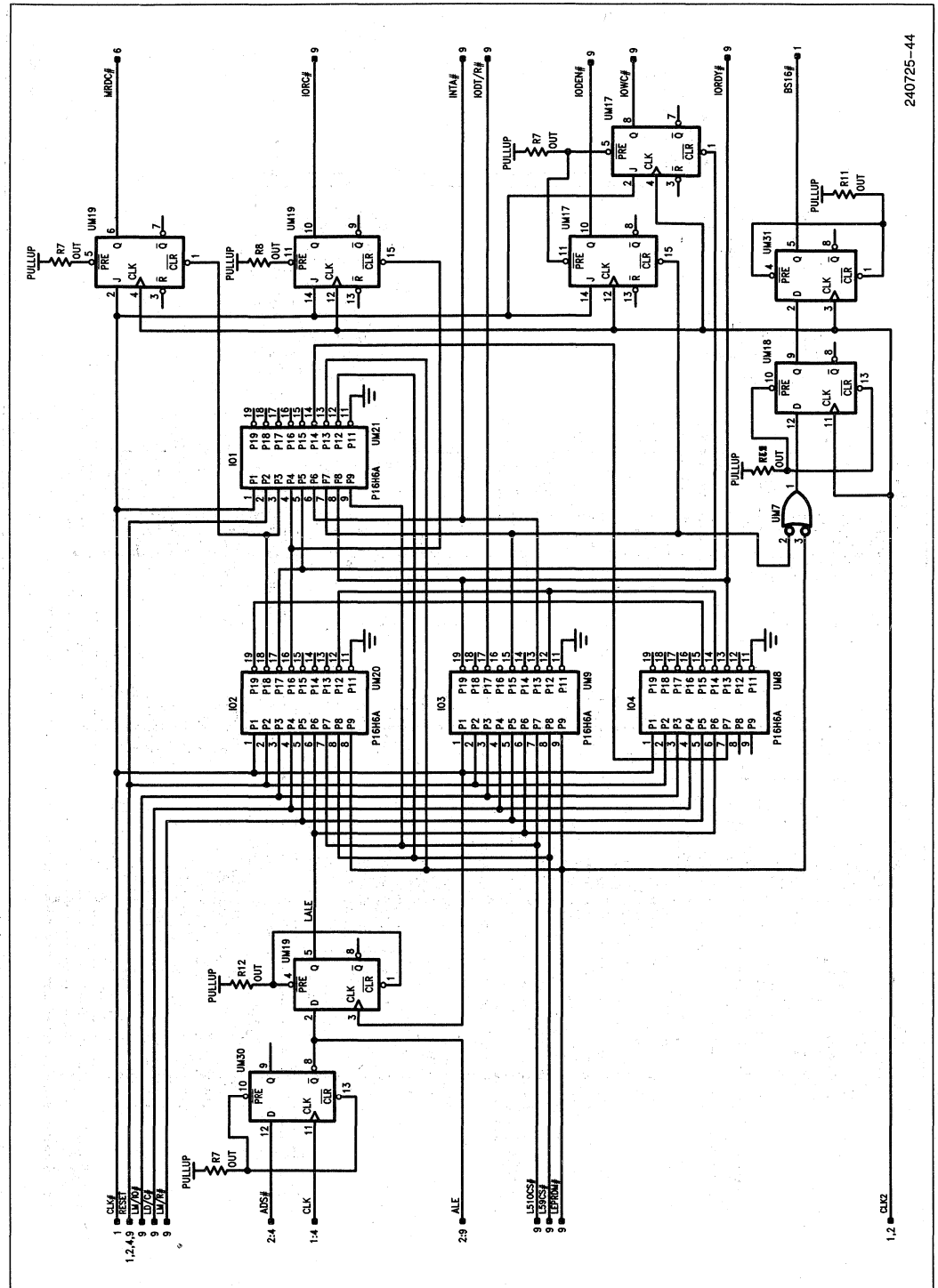

240725-40
REFRESH
CAL
NA#
RAS#
MUXSEL
REF#
MUXDE#
DRAMRDY
CAS#
RAMDT/R#
RAMDEN#
WE#
PRECHARGE
U7
74AS04
DELAY
UM27
P16R8-7
DRAM1
UM12
JEDEC-PAGE1
DRAM2
UM11
JEDEC-PAGE2
DRAM3
UM13
JEDEC-PAGE3
MS-503
UM44
MEMCS#
LMEMCS#
U45
JEDEC-ADDR_PAL
PULLUP
R2
R1
PULLDOWN
R29
JM1
DCLK2
REFREQ
ADS#
M/IO#
A31
A30
A29
A8
EPROM#
510CS#
59CS#
HIT#
RESET
CLK
W/R#
READY#
CACHE

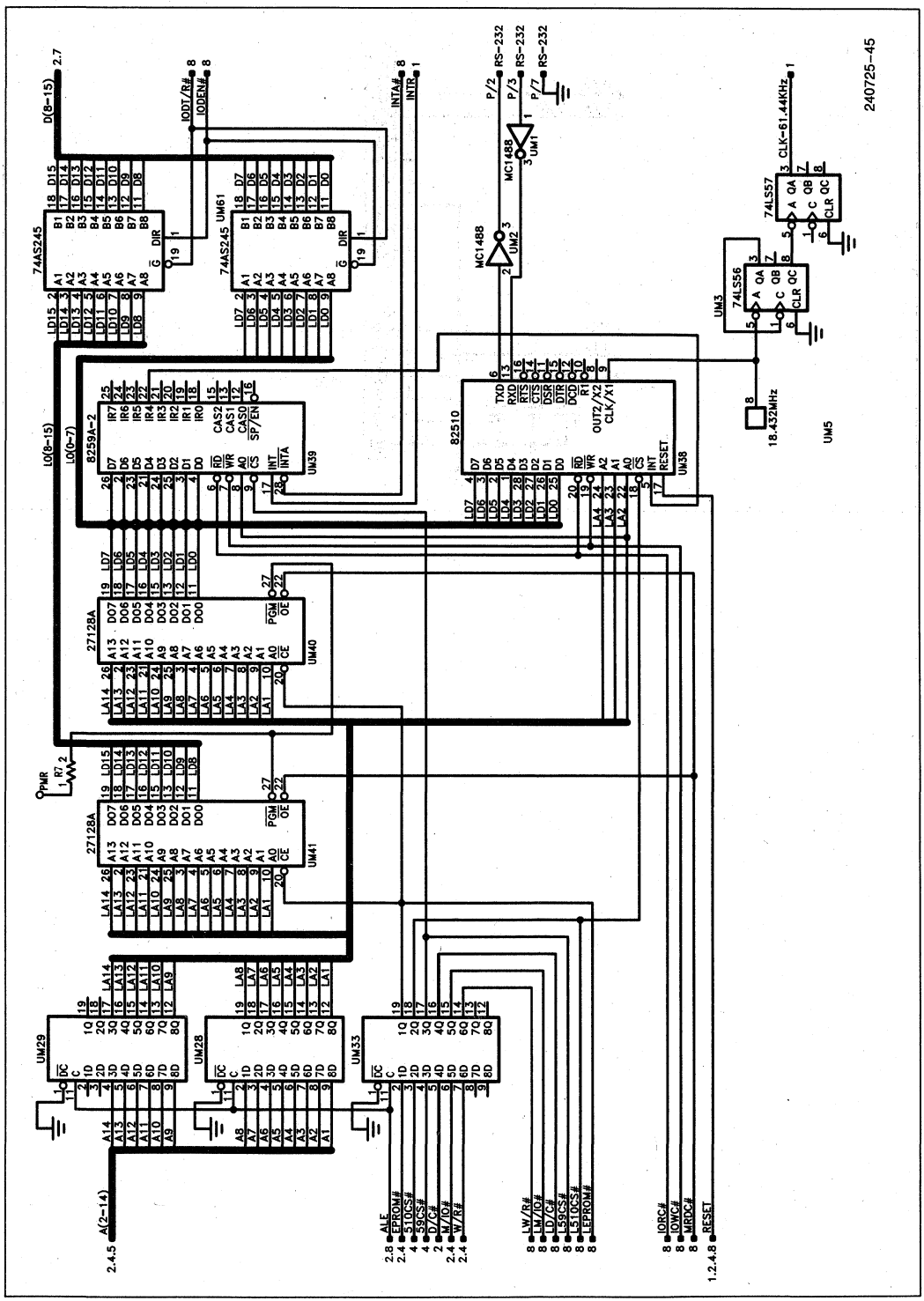

240725-41
HIT#
REF#
DA(0-8)
CAS#(0-3)
RA(0-7)
PULLUP
R28
OUT
UM34
74FCT8438
UM28
74PCT521B
UM32
UM37
P16 RB-7
UM25
UM24
UM14
74AS573
R17
R18
R19
R20
R21
R22
R23
R24
R30
DA0
DA1
DA2
DA3
DA4
DA5
DA6
DA7
DA8
CAS#0
CAS#1
CAS#2
CAS#3
R26
R27
R28
R29
UM15
UM10
U7
RAS#
A(11-19)
A(2-10)
MUXDE#
MUXSEL
CAS# 13
CAL
BE0#
BE1#
BE2#
BE3#
CAS#

5

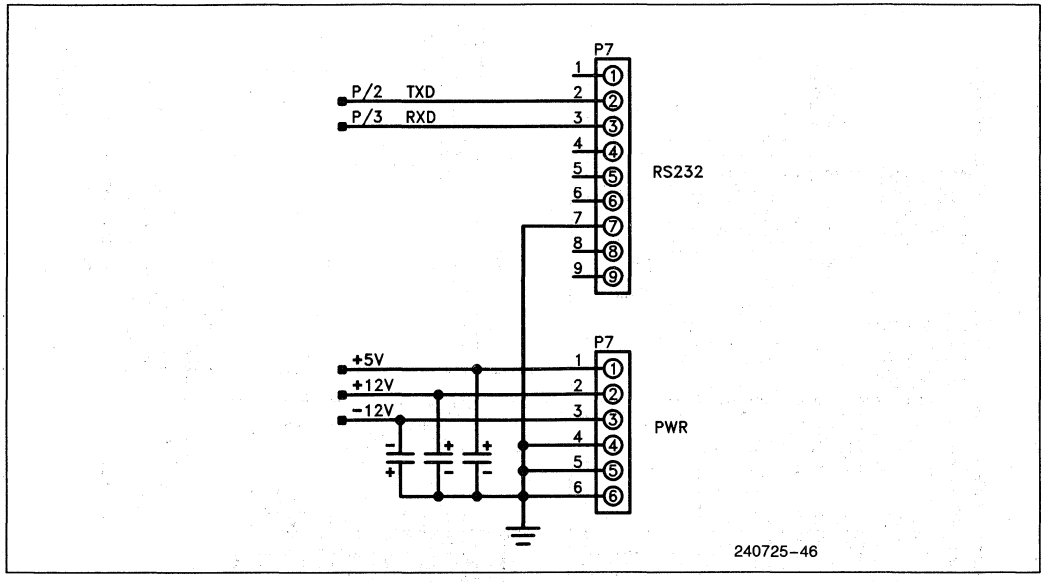

240725-42
74AS245 U81
74AS245 U86
D(16-23)
RAMDT/R#
RAMDEN#
DD(16-23)
DD23
DD22
DD21
DD20
DD19
DD18
DD17
DD16
DIR
NMB2801-06
WE#
RAS#
CAS#3
CAS#2
DA(0-8)

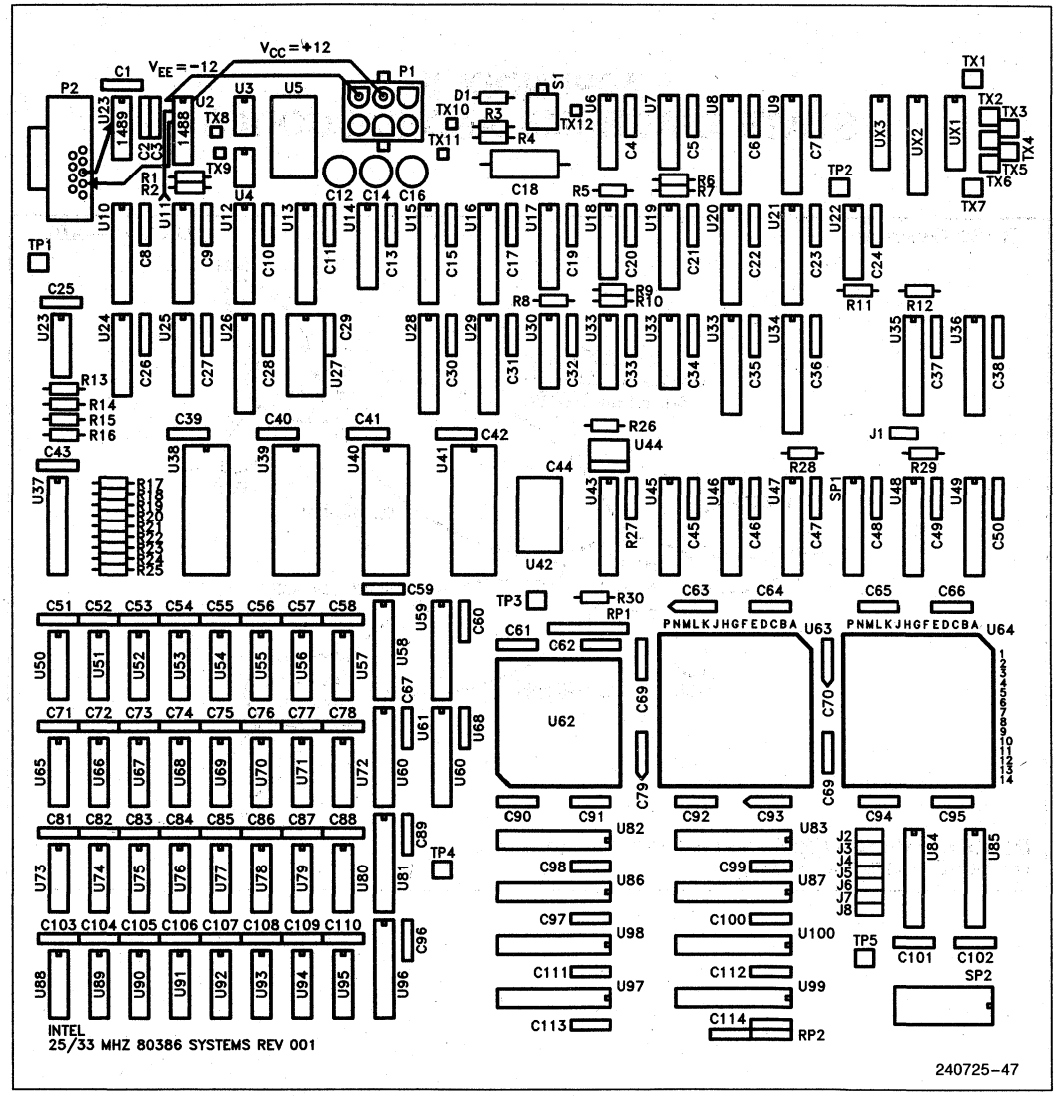

240725-43
74AS245 U58
74AS245 U80
D(8-15)
D(0-7)
RAMDT/R#
RAMDEN#
DO(16-23)
DO(0-7)
NMB2801-06
WE#
RAS#
CAS#1
CAS#0
DA(0-8)


240725-44
MRDC#
IORC#
INTA#
IODT/R#
IODEN#
IOWC#
IORDY#
BS16#
PULLUP
R7
R8
R11
R12
R88
OUT
UM19
UM17
UM31
UM18
UM7
UM30
UM21
UM20
UM9
UM8
P16H6A
IO1
IO2
IO3
IO4
LALE
CLK#
RESET
LS10CS#
LS9CS#
EPROM#
ADS#
CLK
ALE
CLK2


240725-45
D(8-15)
LO(8-15)
LO(0-7)
A(2-14)
74AS245
UM61
8259A-2
UM39
27128A
UM40
UM41
UM29
UM28
UM33
82510
UM38
MC1488
UM1
UM2
74LS557
74LS56
UM3
UM5
18.432MHz
CLK-61.44KHz
RS-232
PWR
R7
IODT/R#
IODEN#
INTA#
INTR
ALE
EPROM#
S10CS#
S39CS#
D/C#
W/O#
W/R#
L39CS#
L510CS#
LEPROM#
IORC#
IOWC#
MRDC#
RESET

5

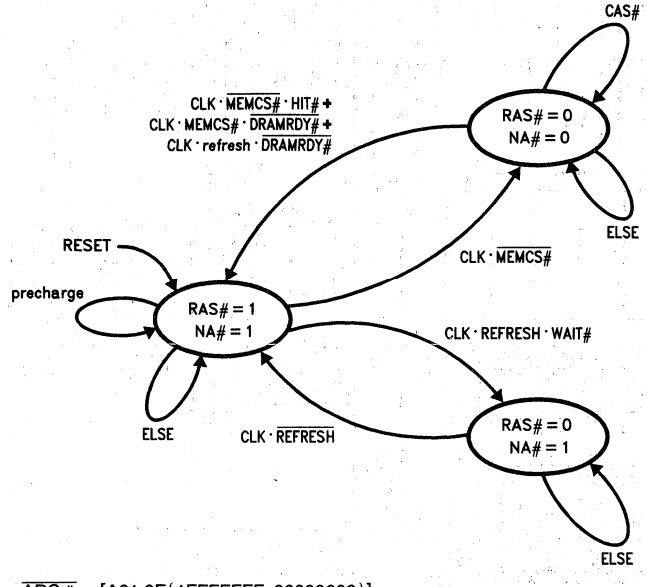

P7
P/2 TXD
P/3 RXD
RS232
P7
+5V
+12V
−12V
PWR
240725-46


V_{CC} = +12
V_{EE} = −12
P1
P2
C1
U1
U2
U3
U4
U5
TX8
TX9
TX10
TX11
TX12
D1
R3
R4
S1
C18
R5
R6
R7
TP1
TP2
TX1
TX2
TX3
TX4
TX5
TX6
TX7
UX1
UX2
UX3
C25
R8
R9
R10
R11
R12
R13
R14
R15
R16
C39
C40
C41
C42
C43
C44
R26
U44
J1
R28
R29
SP1
C59
TP3
R30
RP1
C63
C64
C65
C66
U62
U63
U64
C69
C70
C79
C90
C91
C92
C93
C94
C95
U82
U83
U84
U85
U86
U87
U97
U98
U99
U100
C97
C98
C99
C100
C101
C102
C111
C112
C113
C114
TP4
TP5
RP2
SP2
J2
J8
INTEL
25/33 MHZ 80386 SYSTEMS REV 001
240725-47

5

APPENDIX B
STATE DIAGRAMS AND PALCODES

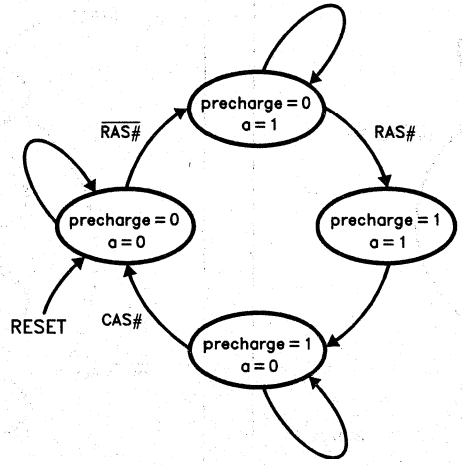
RAS# Generator — DRAM 1—ABL



240725-51

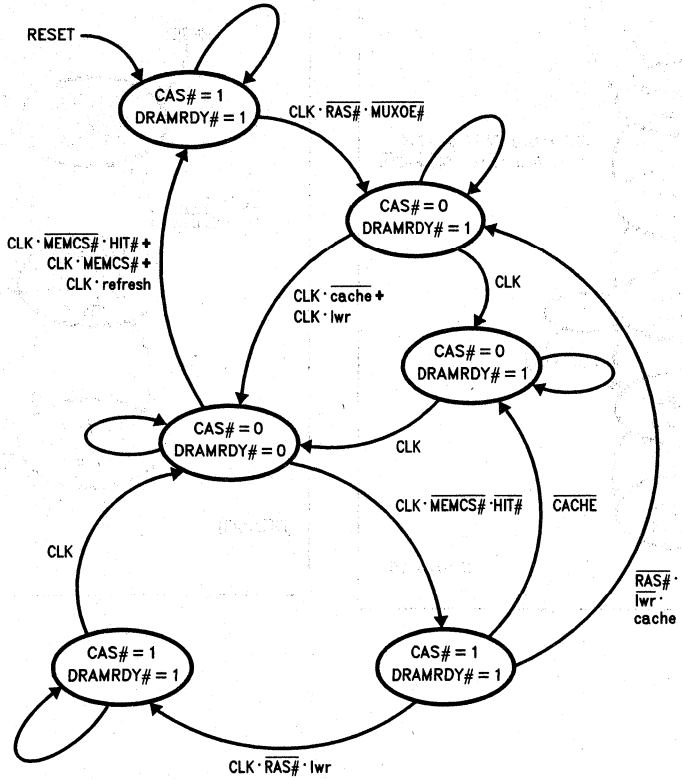
$\overline{MEMCS\#}$ = M/IO# • $\overline{ADS\#}$ • [A31-0E{1FFFFFFF..00000000}]
$\overline{LMEMCS\#}$ = MEMCS# + mreq

Precharge



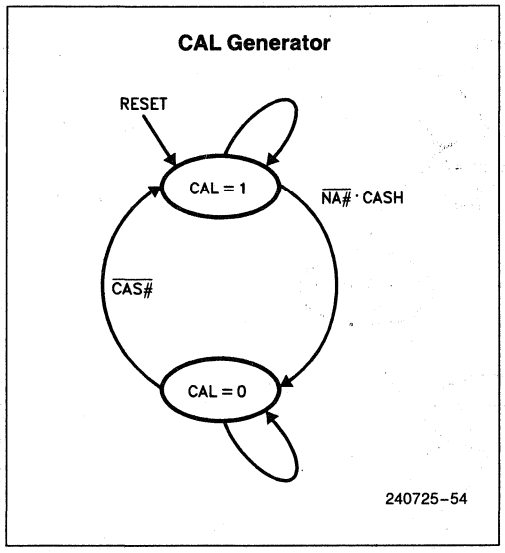
240725–52

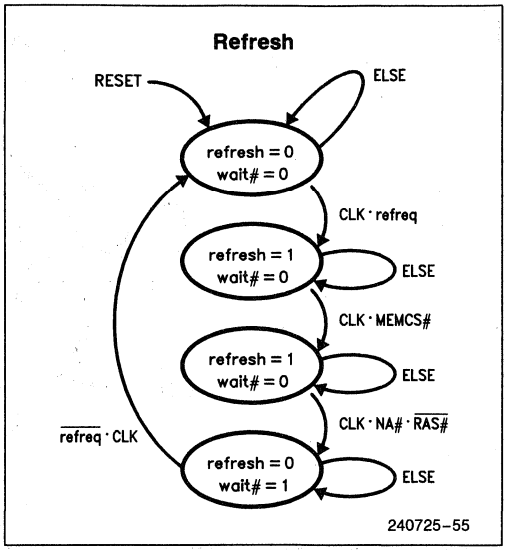
CAS#, READY#

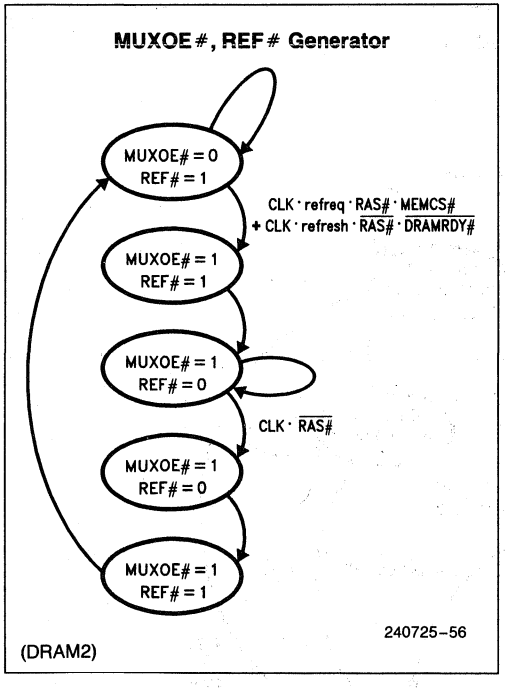


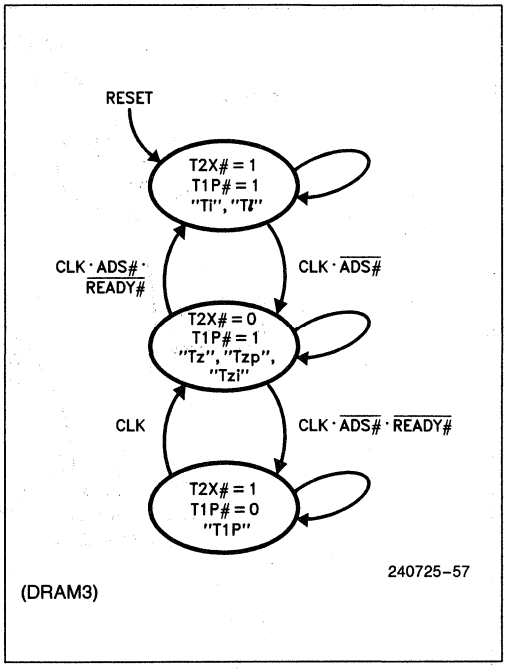
240725–53

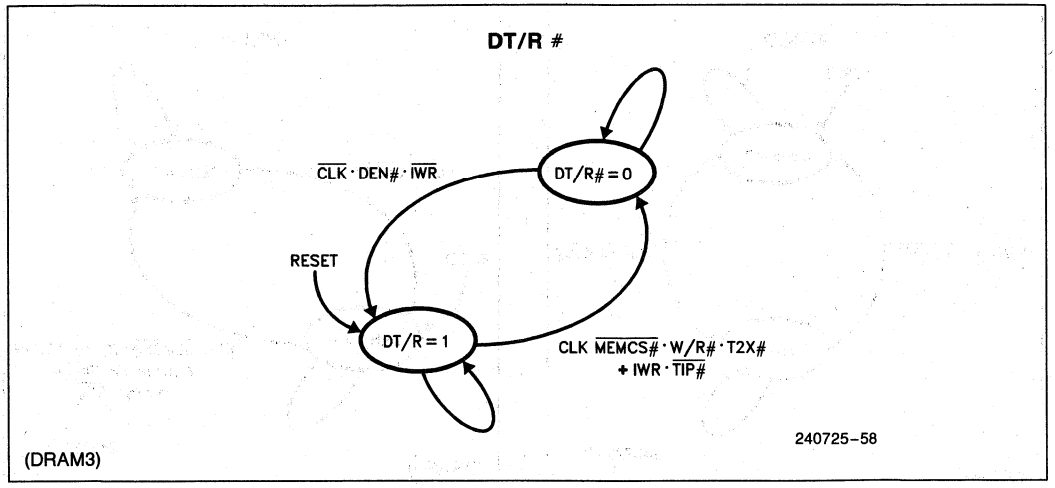
5

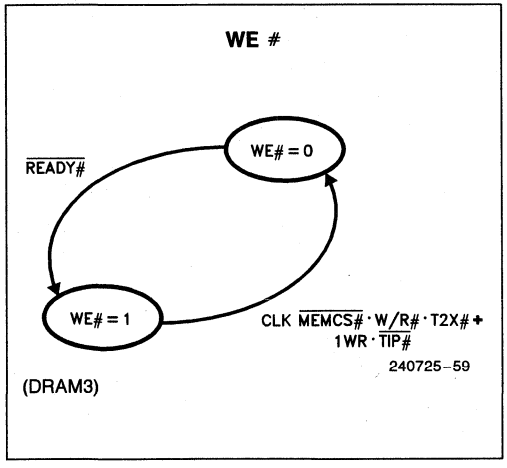

CAL Generator
RESET
CAL = 1
NA# · CASH
CAS#
CAL = 0
240725-54

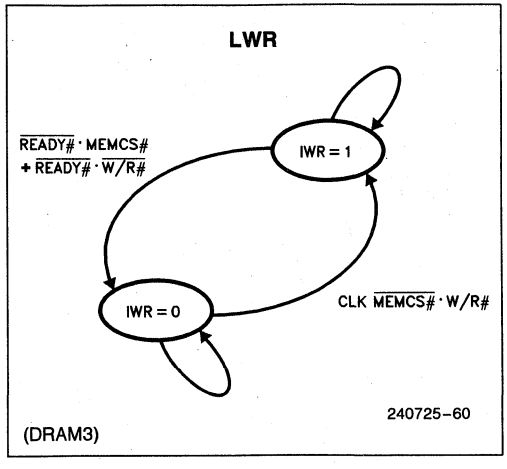

Refresh
RESET
ELSE
refresh = 0
wait# = 0
CLK · refreq
refresh = 1
wait# = 0
ELSE
CLK · MEMCS#
refresh = 1
wait# = 0
ELSE
CLK · NA# · RAS#
refreq · CLK
refresh = 0
wait# = 1
ELSE
240725-55


MUXOE#, REF# Generator
MUXOE# = 0
REF# = 1
CLK · refreq · RAS# · MEMCS#
+ CLK · refresh · RAS# · DRAMRDY#
MUXOE# = 1
REF# = 1
MUXOE# = 1
REF# = 0
CLK · RAS#
MUXOE# = 1
REF# = 0
MUXOE# = 1
REF# = 1
240725-56
(DRAM2)

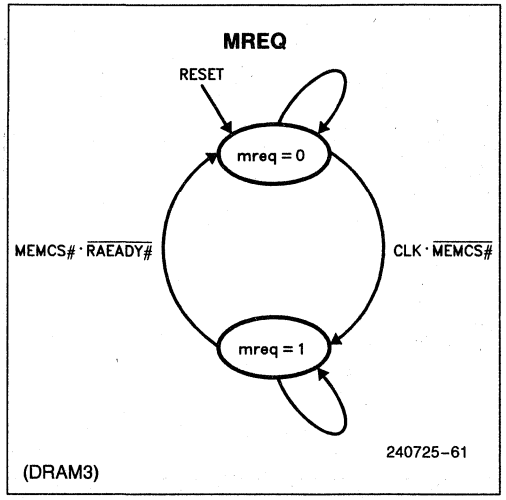

RESET
T2X# = 1
T1P# = 1
"TI", "Tt"
CLK · ADS# ·
READY#
CLK · ADS#
T2X# = 0
T1P# = 1
"Tz", "Tzp",
"Tzi"
CLK
CLK · ADS# · READY#
T2X# = 1
T1P# = 0
"T1P"
240725-57
(DRAM3)

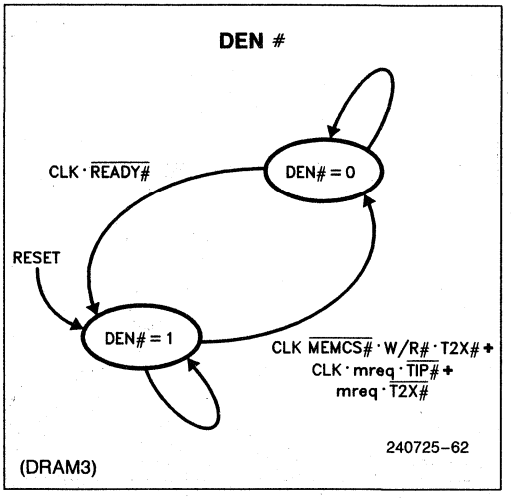

DT/R #
CLK · DEN# · IWR
DT/R# = 0
RESET
DT/R = 1
CLK MEMCS# · W/R# · T2X#
+ IWR · TIP#
240725-58
(DRAM3)


WE #
READY#
WE# = 0
WE# = 1
CLK MEMCS# · W/R# · T2X# +
1WR · TIP#
240725-59
(DRAM3)


LWR
READY# · MEMCS#
+ READY# · W/R#
IWR = 1
IWR = 0
CLK MEMCS# · W/R#
240725-60
(DRAM3)

5


MREQ
RESET
mreq = 0
MEMCS# · RAEADY#
CLK · MEMCS#
mreq = 1
240725-61
(DRAM3)


DEN #
CLK · READY#
DEN# = 0
RESET
DEN# = 1
CLK MEMCS# · W/R# · T2X# +
CLK · mreq · TIP# +
mreq · T2X#
240725-62
(DRAM3)

```
module          RESET_GEN flag '-r3'

title   'RESET_GENERATION_LOGIC - INTEL CORPORATION'

    RESET_PAL     device        'P16R8';

    x = .X.;              "ABEL don't care symbol
    c = .C.;              "ABEL clocking input sybol

" Inputs

    CLK2    pin   1;    "CLK2
    RESTRIG pin   2;    "signal from reset circuitry
    CLK_61  pin   9;    "61.44KHz clock

" Outputs

    REFREQ  pin  12;    "REFREQ, sync 61.44KHz clock
    RFQTMP  pin  13;    "temporary stage in sync of 61.44MHz clk
    CLK~    pin  16;    "CLK#
    CLK     pin  17;    "CLK = CLK2 / 2
    RESTMP  pin  18;    "temporary stage in generating RESET
    RESET   pin  19;    "RESET

equations

    CLK := (!CLK # (!RESTMP & RESET));
    CLK~ := CLK;
    RESTMP := RESTRIG;
    RESET := RESTMP;
    RFQTMP := CLK_61;
    REFREQ := RFQTMP;


 test_vectors

    ([CLK2, CLK_61, RESTRIG, CLK, CLK~, RESTMP, RESET, RFQTMP, REFREQ] ->
     [CLK, CLK~, RESTMP, RESET, RFQTMP, REFREQ])

"   C  C  R  C  C  R  R  R  R      C  C  R  R  R  R
"   L  L  E  L  L  E  E  F  E      L  L  E  E  F  E
"   K  K  S  K  K  S  S  Q  F      K  K  S  S  Q  F
"   2     T     ~  T  E  T  R         ~  T  E  T  R
"      6  R        M  T  M  E            M  T  M  E
"      1  I        P     P  Q            P     P  Q
"         G

    [c, x, 1, x, x, x, x, x] -> [x, x, 1, x, x, x];
    [c, x, 1, x, x, 1, x, x] -> [x, x, 1, 1, x, x];
    [c, x, 0, x, x, 1, x, x] -> [x, x, 0, 1, x, x];

    [c, x, x, x, x, 0, 1, x] -> [1, x, x, x, x, x];  " clk generation
    [c, x, x, 1, x, x, 0, x] -> [0, 1, x, x, x, x];
    [c, x, x, 0, x, x, x, x] -> [1, 0, x, x, x, x];
```

240725-48

PAL Codes: RESET

5

```
    [c, x, x, 1, x, 1, x, x, x] -> [0, 1, x, x, x, x];

    [c, x, 0, x, x, x, x, x, x] -> [x, x, 0, x, x, x];  " restmp gen
    [c, x, x, x, x, 0, x, x, x] -> [x, x, x, 0, x, x];  " reset gen
    [c, x, 1, x, x, x, x, x, x] -> [x, x, 1, x, x, x];

    [c, x, x, x, x, 1, x, x, x] -> [x, x, x, 1, x, x];

    [c, 0, x, x, x, x, x, x, x] -> [x, x, x, x, 0, x];  " 61.44KHz clk
    [c, x, x, x, x, x, x, 0, x] -> [x, x, x, x, x, 0];
    [c, 1, x, x, x, x, x, x, x] -> [x, x, x, x, 1, x];
    [c, x, x, x, x, x, x, 1, x] -> [x, x, x, x, x, 1];

end RESET_GEN;
```

240725-49

```
ABEL(tm) 3.10  -  Document Generator          14-Feb-90 09:53 AM
RESET_GENERATION_LOGIC - INTEL CORPORATION
Equations for Module RESET_GEN

Device RESET_PAL


- Reduced Equations:

    !CLK := (CLK & !RESET # CLK & RESTMP);

    !CLK- := (!CLK);

    !RESTMP := (!RESTRIG);

    !RESET := (!RESTMP);

    !RFQTMP := (!CLK_61);

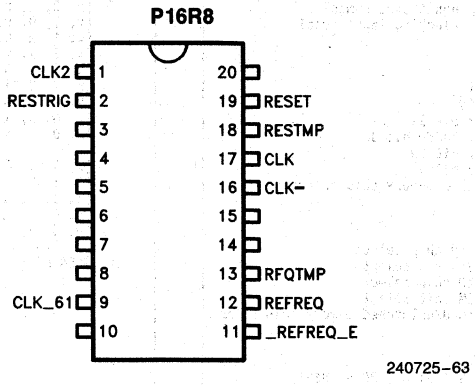
    !REFREQ := (!RFQTMP);
```

240725-D4

PAL Codes: RESET (Continued)

ABEL™ 3.10—Document Generator 14-Feb-90 09:53 AM
RESET_GENERATION_LOGIC—INTEL CORPORATION
Chip diagram for Module RESET_GEN

Device RESET_PAL



PAL Codes: RESET (Continued)

5

```
module          ADDR_DEC flag '-r3'

title   'ADDRESS_DECODE_LOGIC - INTEL CORPORATION'

    ADDR_PAL     device  'P16L8';

    x = .X.;  '          "ABEL don't care symbol
    c = .C.;              "ABEL clocking input sybol

" Inputs

    ADS~     pin   1;    "ADS#
    M_IO~    pin   2;    "M/IO#
    A31      pin   3;    "Addr bit 31
    A30              pin   4;    "Addr bit 30
    A29      pin   5;    "Addr bit 29
    A6       pin   9;    "Addr bit 6
    mreq     pin  11;    "Latched memory chip select

" Outputs

    MEMCS~   pin  18;    "Memory chip select
    _59CS~   pin  15;    "8259A chip select
    _510CS~  pin  14;    "82510 chip select
    EPRDM~   pin  13;    "EPROM chip select
    LMEMCS~  pin  12;    "Latched/unlatched memory chip select

equations

    !MEMCS~ = !ADS~ & M_IO~ & !A31 & !A30 & !A29;
    !LMEMCS~ = (!ADS~ & M_IO~ & !A31 & !A30 & !A29) # mreq;
    !_59CS~ = !M_IO~ & !A6;
    !_510CS~ = !M_IO~ & A6;
    !EPRDM~ =  M_IO~ & A31 & A30 & A29;


 test_vectors

    ([ADS~, M_IO~, A31, A30, A29, A6, mreq, MEMCS~] ->
     [MEMCS~, LMEMCS~, _59CS~, _510CS~, EPRDM~])

"   A  M  A  A  A  A  m  M       M  L  5  5  E
"   D     3  3  2  6  r  E       E  M  9  1  P
"   S  I  1  0  9     e  M       M  E  C  0  R
"   ~  O             q  C       C  M  S  C  D
"      ~                S       S  C  ~  S  M
"                       ~       ~  S     ~  ~
"                                  ~
"

    [1, x, x, x, x, x, 0, 1] -> [1, 1, x, x, x];

    [1, x, x, x, x, x, 1, 1] -> [1, 0, x, x, x];  "LMEMCS~
    [0, 1, 0, 0, 0, x, x, x] -> [0, x, 1, 1, 1];
```

240725-92

```
    [0, 1, 0, 0, 0, x, 0, 0] -> [0, 0, 1, 1, 1];
    [0, 1, 0, 0, 0, x, x, x] -> [0, x, 1, 1, 1];
    [1, x, x, x, x, x, 1, 0] -> [1, 0, x, x, x];

    [1, x, x, x, x, x, x, x] -> [1, x, x, x, x];  "---CS~
    [x, 1, x, x, x, x, x, x] -> [x, x, 1, 1, x];
    [x, 0, x, x, x, x, x, x] -> [1, x, x, x, 1];
    [x, 1, 0, x, x, x, x, x] -> [x, x, 1, 1, 1];
    [x, 1, x, 0, x, x, x, x] -> [x, x, 1, 1, 1];
    [x, 1, x, x, 0, x, x, x] -> [x, x, 1, 1, 1];
    [x, 1, 1, 0, 0, x, x, x] -> [1, x, 1, 1, 1];
    [x, 1, 0, 1, 0, x, x, x] -> [1, x, 1, 1, 1];
    [x, 1, 0, 0, 1, x, x, x] -> [1, x, 1, 1, 1];
    [x, 0, x, x, x, 0, x, x] -> [1, x, x, 1, 1];
    [x, 0, x, x, x, 1, x, x] -> [1, x, 1, x, 1];

    [x, 1, 1, 1, 1, x, x, x] -> [1, x, 1, 1, 0];
    [0, 1, 0, 0, 0, x, x, x] -> [0, x, 1, 1, 1];
    [1, 1, 0, 0, 0, x, x, x] -> [1, x, 1, 1, 1];
    [0, 0, x, x, x, 0, x, x] -> [1, x, 0, 1, 1];
    [0, 0, x, x, x, 1, x, x] -> [1, x, 1, 0, 1];

end ADDR_DEC;
```

240725-93

PAL Codes: Address Decoder

```
ABEL(tm) 3.10  -  Document Generator          14-Feb-90 09:50 AM
ADDRESS_DECODE_LOGIC - INTEL CORPORATION
Equations for Module ADDR_DEC

Device ADDR_PAL


- Reduced Equations:

    !MEMCS~ = (!A29 & !A30 & !A31 & !ADS~ & M_IO~);

    !LMEMCS~ = (mreq # !A29 & !A30 & !A31 & !ADS~ & M_IO~);

    !_59CS~ = (!A6 & !M_IO~);

    !_510CS~ = (A6 & !M_IO~);

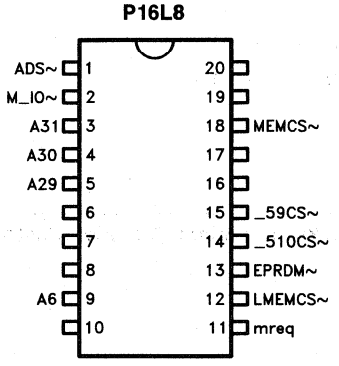
    !EPRDM~ = (A29 & A30 & A31 & M_IO~);
```

240725-D5

PAL Codes: Address Decoder (Continued)

ABEL™ 3.10—Document Generator 14-Feb-90 09:50 AM
ADDRESS_DECODE_LOGIC—INTEL CORPORATION
Chip diagram for Module ADDR_DEC

Device ADDR_PAL



PAL Codes: Address Decoder (Continued)

```
module          PAGE_MODE_DRAM_CTRL_1  flag '-r3'

title   'PAGE MODE DRAM CONTROLLER - PAL 1, INTEL CORPORATION'

    PAGE1    device      'P16R8';

    x       =     .X.;          " ABEL 'don't care' symbol
    c       =     .C.;          " ABEL 'clocking input' symbol

" Inputs

    CLK2    pin   1;   "80386 CLK2
    CLK     pin   2;   "Processor Clock
    MEMCS-  pin   3;   "Memory Chip Select
    LMEMCS- pin   4;   "Latched/Unlatched Memory Chip Select
    HIT-    pin   5;   "DRAM Page Hit Signal
    CAS-    pin   6;   "Column Address Strobe
    DRAMRDY-      pin   7;   "DRAM Ready Signal
    refreq  pin   8;   "Refresh Request Signal
    RESET   pin   9;   "System Reset

" Outputs

    RAS-    pin  12;    "Row Address Strobe
    NA-     pin  13;    "Next Address Signal
    precharge     pin  14;    "RAS Precharge Signal
    a       pin  15;
    wait-   pin  16;   "delays RAS- until refresh adress is valid
    CAL     pin  17;   "Column Address Latch
    refresh pin  18;   "Refresh Signal (active once refresh is acknowledged.)

    unused  pin  19;   "

state_diagram [RAS-, NA-]

    state [1, 1]:    if precharge then [1, 1] else
                   if (CLK & refresh & wait-) then [0, 1] else
                   if (CLK & !LMEMCS-& !refresh) then [0, 0] else [1, 1];
    state [0, 0]:    if RESET then [1, 1] else
                   if CAS- then [0, 0] else
                   if (CLK & !MEMCS- & HIT- #
                     CLK & MEMCS- & !DRAMRDY- #
                     CLK & refresh & !DRAMRDY-) then [1, 1] else [0, 0];
    state [0, 1]:    if RESET then [1, 1] else
                   if (CLK & !refresh) then [1, 1] else [0, 1];
    state [1, 0]:    goto [1, 1];

state_diagram [precharge, a]

    state [0, 0]:    if (!RAS-) then [0, 1] else [0, 0];
    state [0, 1]:    if (RESET) then [0, 0] else
                   if (RAS-) then [1, 1] else [0, 1];
    state [1, 1]:    goto [1, 0];
    state [1, 0]:    if (CAS-) then [0, 0] else [1, 0];
```

240725-94

PAL Codes: DRAM 1

5

```
state_diagram [CAL]

    state [1]:      if (!NA~ & CAS~) then [0] else [1];
    state [0]:      if (RESET) then [1] else
              if (!CAS~) then [1] else [0];

state_diagram [refresh, wait~]

    state[0, 0]:    if (CLK & refreq) then [1, 0] else [0, 0];
    state[1, 0]:    if (RESET) then [0,0] else
                        if (CLK & MEMCS~) then [1, 1] else [1, 0];
    state[1, 1]:    if (RESET) then [0,0] else
                        if (CLK & NA~ & !RAS~) then [0, 1] else [1, 1];
    state[0, 1]:    if (RESET) then [0,0] else
                        if (CLK & !refreq) then [0, 0] else [0, 1];

test_vectors

    ([CLK2,CLK,MEMCS~,LMEMCS~,HIT~,CAS~,DRAMRDY~,refreq,RESET] ->
     [RAS~,NA~,precharge,CAL,refresh])

"    C  C  M  L  H  C  D  r  R      R  N  p  C  r
"    L  L  E  M  I  A  R  e  E      A  A  r  A  e
"    K  K  M  E  T  S  A  f  S      S  ~  e  L  f
"    2     C  M  ~  ~  M  r  E      ~     c     r
"           S  C        R  e  T            h     e
"           ~  S        D  q               a     s
"              ~        Y                  r     h
"                       ~                  g
"                                          e

    [c, x, x, x, x, x, 1, x, 1] -> [1, 1, x, 1, 0];
    [c, x, x, x, x, x, 1, x, 1] -> [1, 1, x, 1, 0];
    [c, 1, 1, 1, x, 1, 1, 0, 0] -> [1, 1, x, 1, 0]; "Ti, phase 1
    [c, 0, 1, 1, x, 1, 1, 0, 0] -> [1, 1, x, 1, 0]; "    phase 2
    [c, 1, 1, 1, x, 1, 1, 0, 0] -> [1, 1, x, 1, 0]; "T1, Read, Non-Pipelined
    [c, 0, 0, 0, x, 1, 1, 0, 0] -> [1, 1, 0, 1, 0];
    [c, 1, 0, 0, x, 1, 1, 0, 0] -> [0, 0, 0, 1, 0]; "T2
    [c, 0, 1, 0, x, 1, 1, 0, 0] -> [0, 0, 0, 0, 0];
    [c, 1, 1, 0, x, 1, 1, 0, 0] -> [0, 0, 0, 0, 0]; "T2P
    [c, 0, 0, 0, x, 0, 1, 0, 0] -> [0, 0, 0, 1, 0]; "     Page Hit
    [c, 1, 0, 0, 0, 0, 1, 0, 0] -> [0, 0, 0, 1, 0]; "T2P
    [c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0];
    [c, 1, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0]; "T1P, Read, Pipelined
    [c, 0, 1, 0, 0, 1, 1, 0, 0] -> [0, 0, 0, 0, 0];
    [c, 1, 1, 0, 0, 0, 1, 0, 0] -> [0, 0, 0, 1, 0]; "T2P
    [c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0];
    [c, 1, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0]; "T1P, Write
    [c, 0, 1, 0, 0, 1, 1, 0, 0] -> [0, 0, 0, 0, 0];
    [c, 1, 1, 0, 0, 1, 1, 0, 0] -> [0, 0, 0, 0, 0]; "T2P
    [c, 0, 0, 0, 0, 1, 1, 0, 0] -> [0, 0, 0, 0, 0];
    [c, 1, 0, 0, 0, 1, 1, 0, 0] -> [0, 0, 0, 0, 0]; "T2P
    [c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0];
```

240725-95

PAL Codes: DRAM 1 (Continued)

```
[c, 1, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0]; "T1P
[c, 0, 1, 0, 0, 1, 1, 0, 0] -> [0, 0, 0, 0, 0];
[c, 1, 1, 0, 0, 0, 1, 0, 0] -> [0, 0, 0, 1, 0]; "T2P
[c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0]; "    Page Miss
[c, 1, 0, 0, 1, 0, 0, 0, 0] -> [1, 1, 0, 1, 0]; "T1P
[c, 0, 1, 0, 1, 1, 1, 0, 0] -> [1, 1, 1, 1, 0];
[c, 1, 1, 0, 1, 1, 1, 0, 0] -> [1, 1, 1, 1, 0]; "T2
[c, 0, 1, 0, 1, 1, 1, 0, 0] -> [1, 1, 0, 1, 0];
[c, 1, 1, 0, 1, 1, 1, 0, 0] -> [0, 0, 0, 1, 0]; "T2
[c, 0, 1, 0, 1, 1, 1, 0, 0] -> [0, 0, 0, 0, 0];
[c, 1, 1, 0, 1, 1, 1, 0, 0] -> [0, 0, 0, 0, 0]; "T2P
[c, 0, 0, 0, x, 0, 1, 0, 0] -> [0, 0, 0, 1, 0];
[c, 1, 0, 0, 1, 0, 1, 0, 0] -> [1, 1, 0, 1, 0]; "T2P
[c, 0, 0, 0, 1, 0, 0, 0, 0] -> [1, 1, 1, 1, 0];
[c, 1, 0, 0, 1, 0, 0, 0, 0] -> [1, 1, 1, 1, 0]; "T1P
[c, 0, 1, 0, 1, 1, 1, 0, 0] -> [1, 1, 0, 1, 0];
[c, 1, 1, 0, 1, 1, 1, 0, 0] -> [0, 0, 0, 1, 0]; "T2
[c, 0, 1, 0, 1, 1, 1, 0, 0] -> [0, 0, 0, 0, 0];
[c, 1, 1, 0, 1, 1, 1, 0, 0] -> [0, 0, 0, 0, 0]; "T2P
[c, 0, 0, 0, 0, 1, 0, 1, 0] -> [0, 0, 0, 1, 0];
[c, 1, 0, 0, 0, 0, 1, 0, 0] -> [0, 0, 0, 1, 0]; "T2P
[c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0];
[c, 1, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0]; "T1P
[c, 0, 1, 0, 0, 1, 1, 0, 0] -> [0, 0, 0, 0, 0];
[c, 1, 1, 0, 0, 0, 1, 0, 0] -> [0, 0, 0, 1, 0]; "T2P
[c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0];
[c, 1, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0]; "T1P
[c, 0, 1, 0, 0, 1, 1, 0, 0] -> [0, 0, 0, 0, 0];
[c, 1, 1, 0, 0, 0, 1, 0, 0] -> [0, 0, 0, 1, 0]; "T2i
[c, 0, 1, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0];
[c, 1, 1, 1, 0, 0, 0, 0, 0] -> [1, 1, 0, 1, 0]; "T1
[c, 0, 0, 0, x, 1, 1, 0, 0] -> [1, 1, 1, 1, 0];
[c, 1, 0, 0, x, 1, 1, 0, 0] -> [1, 1, 1, 1, 0]; "T2
[c, 0, 1, 0, x, 1, 1, 0, 0] -> [1, 1, 0, 1, 0];
[c, 1, 1, 0, x, 1, 1, 0, 0] -> [0, 0, 0, 1, 0]; "T2
[c, 0, 1, 0, x, 1, 1, 0, 0] -> [0, 0, 0, 0, 0];
[c, 1, 1, 0, x, 1, 1, 0, 0] -> [0, 0, 0, 0, 0]; "T2P
[c, 0, 0, 0, x, 0, 1, 0, 0] -> [0, 0, 0, 1, 0];
[c, 1, 0, 0, 0, 0, 1, 0, 0] -> [0, 0, 0, 1, 0]; "T2P
[c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0];
[c, 1, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0]; "T1P
[c, 0, 1, 0, 0, 1, 1, 0, 0] -> [0, 0, 0, 0, 0];
[c, 1, 1, 0, 0, 0, 1, 0, 0] -> [0, 0, 0, 1, 0]; "T2P
[c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0];
[c, 1, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0]; "T1P
[c, 0, 1, 0, 0, 1, 1, 1, 0] -> [0, 0, 0, 0, 0];
[c, 1, 1, 0, 0, 0, 1, 1, 0] -> [0, 0, 0, 1, 1]; "T2P
[c, 0, 0, 0, 0, 0, 0, 1, 0] -> [0, 0, 0, 1, 1];
[c, 1, 0, 0, 0, 0, 0, 1, 0] -> [1, 1, 0, 1, 1]; "T1P, Refresh
[c, 0, 1, 0, 0, 1, 1, 1, 0] -> [1, 1, 1, 1, 1];
[c, 1, 1, 0, 0, 1, 1, 1, 0] -> [1, 1, 1, 1, 1]; "T2
[c, 0, 1, 0, 0, 1, 1, 1, 0] -> [1, 1, 0, 1, 1];
[c, 1, 1, 0, 0, 1, 1, 1, 0] -> [0, 1, 0, 1, 1]; "T2
[c, 0, 1, 0, 0, 1, 1, 0, 0] -> [0, 1, 0, 1, 1];
```

240725-96

```
    [c, 1, 1, 0, 0, 1, 1, 0, 0] -> [0, 1, 0, 1, 0]; "T2
    [c, 0, 1, 0, 0, 1, 1, 0, 0] -> [0, 1, 0, 1, 0];
    [c, 1, 1, 0, 0, 1, 1, 0, 0] -> [1, 1, 0, 1, 0]; "T2, Pending Read
    [c, 0, 1, 0, 0, 1, 1, 0, 0] -> [1, 1, 1, 1, 0];
    [c, 1, 1, 0, 0, 1, 1, 0, 0] -> [1, 1, 1, 1, 0]; "T2
    [c, 0, 1, 0, 0, 1, 1, 0, 0] -> [1, 1, 0, 1, 0];
    [c, 1, 1, 0, 0, 1, 1, 0, 0] -> [0, 0, 0, 1, 0]; "T2
    [c, 0, 1, 0, 0, 1, 1, 0, 0] -> [0, 0, 0, 0, 0];
    [c, 1, 1, 0, 0, 1, 1, 0, 0] -> [0, 0, 0, 0, 0]; "T2P
    [c, 0, 0, 0, 0, 0, 1, 0, 0] -> [0, 0, 0, 1, 0];
    [c, 1, 0, 0, 0, 0, 1, 0, 0] -> [0, 0, 0, 1, 0]; "T2P
    [c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0];
    [c, 1, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1, 0]; "T1P
end PAGE_MODE_DRAM_CTRL_1;
```

240725-97

PAL Codes: DRAM 1 (Continued)

```
ABEL(tm) 3.10  -  Document Generator            15-Feb-90 05:47 PM
PAGE MODE DRAM CONTROLLER - PAL 1, INTEL CORPORATION
Equations for Module PAGE_MODE_DRAM_CTRL_1

Device PAGE1


- Reduced Equations:

    !RAS- := (NA- & !RAS- & !RESET & refresh
              # DRAMRDY- & !HIT- & !NA- & !RAS- & !RESET
              # DRAMRDY- & MEMCS- & !NA- & !RAS- & !RESET
              # !HIT- & !MEMCS- & !NA- & !RAS- & !RESET & !refresh
              # !CLK & !RAS- & !RESET
              # CAS- & !NA- & !RAS- & !RESET
              # CLK & !LMEMCS- & NA- & RAS- & !precharge & !refresh
              # CLK & NA- & RAS- & !precharge & refresh & wait-);


    !NA- := (DRAMRDY- & !HIT- & !NA- & !RAS- & !RESET
             # DRAMRDY- & MEMCS- & !NA- & !RAS- & !RESET
             # !HIT- & !MEMCS- & !NA- & !RAS- & !RESET & !refresh
             # !CLK & !NA- & !RAS- & !RESET
             # CAS- & !NA- & !RAS- & !RESET
             # CLK & !LMEMCS- & NA- & RAS- & !precharge & !refresh);


    !precharge := (CAS- & !a
                   # !RAS- & !precharge
                   # RESET & !precharge
                   # !a & !precharge);


    !a := (precharge # RESET & a # RAS- & !a);

    !CAL := (!CAL & CAS- & !RESET # CAL & CAS- & !NA-);

    !refresh := (!refresh & wait-
                 # CLK & NA- & !RAS- & wait-
                 # RESET & refresh
                 # !refreq & !refresh
                 # !CLK & !refresh);

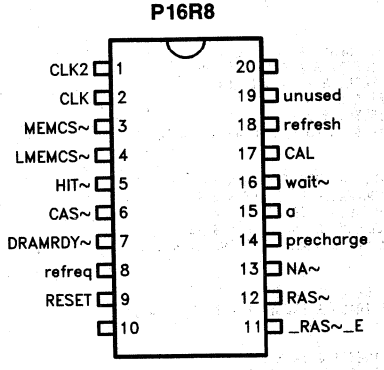

    !wait- := (CLK & !refreq & !refresh
               # !MEMCS- & !wait-
               # !CLK & !wait-
               # RESET
               # !refresh & !wait-);
```

240725-50

PAL Codes: DRAM 1 (Continued)

ABEL™ 3.10—Document Generator 15-Feb-90 05:47 PM
PAGE MODE DRAM CONTROLLER—PAL 1, INTEL CORPORATION
Chip diagram for Module PAGE_MODE_CTRL_1

Device PAGE1



240725-65

PAL Codes: DRAM 1 (Continued)

5

```
ABEL(tm) 3.10  -  Document Generator            15-Feb-90 06:16 PM
PAGE MODE DRAM CONTROLLER - PAL 2, INTEL CORPORATION
Equations for Module PAGE_MODE_DRAM_CTRL_2

Device PAGE2


- Reduced Equations:

    !CAS~ := (CAS~ & CLK & DRAMRDY~ & !RESET & !a & !b
              # !CACHE & DRAMRDY~ & !RESET & a & !b & !lwr
              # DRAMRDY~ & !RAS~ & !RESET & a & !b & !lwr
              # !CAS~ & !CLK & !RESET & a & b
              # !CAS~ & DRAMRDY~ & !RESET & a
              # CAS~ & CLK & DRAMRDY~ & !MUXOE~ & !RAS~ & a & b);


    !DRAMRDY~ := (CAS~ & CLK & DRAMRDY~ & !RESET & !a & !b
                  # !CAS~ & !CLK & !DRAMRDY~ & !RESET & a & b
                  # !CAS~ & CLK & DRAMRDY~ & !RESET & a & !b
                  # !CAS~ & CLK & DRAMRDY~ & !RESET & a & lwr
                  # !CACHE & !CAS~ & CLK & DRAMRDY~ & !RESET & a);


    !a := (CAS~ & !CLK & DRAMRDY~ & !RESET & !a & !b
           # CAS~ & CLK & DRAMRDY~ & !RAS~ & !RESET & a & !b & lwr);


    !b := (CAS~ & !CLK & DRAMRDY~ & !RESET & !a & !b
           # CAS~ & DRAMRDY~ & RAS~ & !RESET & a & !b
           # !CACHE & CAS~ & DRAMRDY~ & !RESET & a & !b
           # CAS~ & DRAMRDY~ & !RESET & a & !b & lwr
           # !CAS~ & CLK & !DRAMRDY~ & !MEMCS~ & !RAS~ & !RESET & a & b &
         !refresh
           # !CAS~ & !CLK & DRAMRDY~ & !RESET & a & !b
           # CACHE & !CAS~ & CLK & DRAMRDY~ & !RESET & a & b & !lwr);


    !MUXOE~ := (!MUXOE~ & !REF~
                # REF~ & !r
                # MUXOE~ & RESET
                # DRAMRDY~ & !MUXOE~ & !RAS~
                # !MEMCS~ & !MUXOE~ & RAS~
                # !MUXOE~ & !refresh
                # !CLK & !MUXOE~);


    !REF~ := (MUXOE~ & !RESET & r);

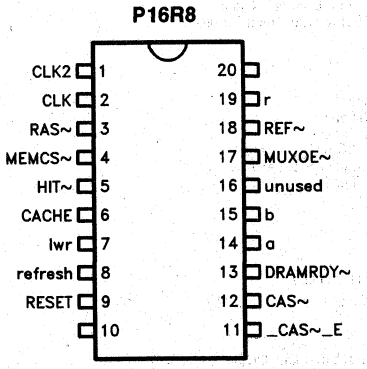
    !r := (MUXOE~ & !REF~ & !RESET & !r
           # CLK & MUXOE~ & !RAS~ & !REF~ & !RESET);
```

240725-98

PAL Codes: DRAM 2

ABEL™ 3.10—Document Generator 15-Feb-90 06:16 PM
PAGE MODE DRAM CONTROLLER—PAL 2, INTEL CORPORATION
Chip diagram for Module PAGE_MODE_DRAM_CTRL_2

Device PAGE2



240725-66

PAL Codes: DRAM 2 (Continued)

5

```
module          PAGE_MODE_DRAM_CTRL_2  flag '-r3'

title   'PAGE MODE DRAM CONTROLLER - PAL 2, INTEL CORPORATION'

    PAGE2    device      'P16R8';

    x       =     .X.;        " ABEL 'don't care' symbol
    c       =     .C.;        " ABEL 'clocking input' symbol

" Inputs

    CLK2    pin   1;   "80386 CLK2
    CLK     pin   2;   "Processor Clock
    RAS~    pin   3;   "Row Address Strobe
    MEMCS~  pin   4;   "Memory Chip Select
    HIT~    pin   5;   "DRAM Page Hit Signal (unused)
    CACHE   pin   6;   "Hi when 385 is used; otherwise, Low
    lwr     pin   7;   "Latched Write/Read
    refresh pin   8;   "Refresh Signal
    RESET   pin   9;   "System Reset

" Outputs

    CAS~    pin  12;    "Column Address Strobe
    DRAMRDY~     pin  13;    "DRAM Ready
    a       pin  14;    "
    b       pin  15;    "
    unused  pin  16;   "
    MUXOE~  pin  17;   "DRAM Address Multiplexer Output Enable
    REF~    pin  18;   "Enables refresh counter instead of MUX
    r       pin  19;

    cstate  =  [CAS~, DRAMRDY~,a, b];
    idle    =  [  1 ,   1     ,1, 1];  "Idle
    start   =  [  0 ,   1     ,1, 1];  "CAS~ Active
    wait    =  [  0,    1     ,1, 0];  "CAS~ Active, Wait State
    active  =  [  0 ,   0     ,1, 1];  "CAS~ and DRAMRDY~ Active
    inactive_1    =  [  1 ,   1     ,1,  0];   "Page  Hit,  CAS~  and  DRAMRDY-
Inactive
    inactive_2    =  [  1 ,   1     ,0,  0];   "Page  Hit,  CAS~  and  DRAMRDY-
Inactive
    illegal_a     =  [0,0,0,0];
    illegal_b     =  [0,0,0,1];
    illegal_c     =  [0,0,1,0];
    illegal_d     =  [0,1,0,0];
    illegal_e     =  [0,1,0,1];
    illegal_f     =  [1,0,0,1];
    illegal_g     =  [1,0,1,0];
    illegal_h     =  [1,0,1,1];
    illegal_i     =  [1,1,0,1];
    illegal_j     =  [1,0,0,0];

    muxstate      =  [MUXOE~, REF~, r];
    enabled =  [  0   ,  1    , 1];  "Multiplexer Outputs Enabled
```

240725-99

PAL Codes: DRAM 2 (Continued)

```
    disabled_1    = [  1   ,  1    , 1];  "Multiplexer Outputs Disabled
    disabled_2    = [  1   ,  0    , 1];  "Refresh Address Enabled
    disabled_3    = [  1   ,  0    , 0];  "Wait for RAS#
    disabled_4    = [  1   ,  1    , 0];  "Refresh Address Disabled
    illegal_z     = [0,0,0];
    illegal_y     = [0,0,1];
    illegal_x     = [0,1,0];

state_diagram cstate

    state idle:       if (CLK & !RAS- & !MUXOE-) then start else idle;
    state start:      if RESET then idle else
                    if (CLK & !CACHE # CLK & lwr) then active else
                    if CLK then wait else start;
    state wait:       if RESET then idle else
                    if CLK then active else wait;
    state active:     if RESET then idle else
                    if (CLK & !MEMCS- & RAS- #
                      CLK & MEMCS- #
                      CLK & refresh) then idle else
                    if (CLK & !MEMCS- & !RAS-) then inactive_1
                    else active;
    state inactive_1: if RESET then idle else
                    if (CLK & !RAS- & lwr) then inactive_2 else
                    if (!RAS- & !lwr & CACHE) then start else
                    if (!lwr & !CACHE) then wait else
                    inactive_1;
    state inactive_2: if RESET then idle else
                    if CLK then active else inactive_2;
    state illegal_a:  goto idle;
    state illegal_b:  goto idle;
    state illegal_c:  goto idle;
    state illegal_d:  goto idle;
    state illegal_e:  goto idle;
    state illegal_f:  goto idle;
    state illegal_g:  goto idle;
    state illegal_h:  goto idle;
    state illegal_i:  goto idle;
    state illegal_j:  goto idle;

state_diagram muxstate

    state enabled:    if (CLK & refresh & RAS- & MEMCS- #
                    CLK & refresh & !RAS- & !DRAMRDY-) then
                    disabled_1 else enabled;
    state disabled_1: if (RESET) then enabled else disabled_2;
    state disabled_2: if (RESET) then enabled else
                    if (CLK & !RAS-) then disabled_3 else disabled_2;
    state disabled_3: if (RESET) then enabled else disabled_4;
    state disabled_4: goto enabled;
    state illegal_z:  goto enabled;
    state illegal_y:  goto enabled;
    state illegal_x:  goto enabled;
```

240725-A0

PAL Codes: DRAM 2 (Continued)

5

```
test_vectors

    ([CLK2,CLK,MEMCS~,!wr,HIT~,RAS~,refresh,RESET,CACHE] ->
     [CAS~,DRAMRDY~,MUXOE~,REF~])


"    C  C  M  !  H  R  r  R  C      C  D  M  R
"    L  L  E  w  I  A  e  E  A      A  R  U  E
"    K  K  M  r  T  S  f  S  C      S  A  X  F
"    2     C     ~  ~  r  E  H      ~  M  O  ~
"          S           e  T  E         R  E
"          ~           s               D  ~
"                      h               Y
"                           ~


    [c, x, x, 0, x, x, x, 1, 0] -> [1, 1, 0, 1]; "Cache disabled
    [c, x, x, 0, x, x, x, 1, 0] -> [1, 1, 0, 1];
    [c, 0, 1, 0, x, 1, 0, 0, 0] -> [1, 1, 0, 1]; "Ti
    [c, 1, 1, 0, x, 1, 0, 0, 0] -> [1, 1, 0, 1];
    [c, 0, 0, 0, x, 1, 0, 0, 0] -> [1, 1, 0, 1]; "T1
    [c, 1, 0, 0, x, 1, 0, 0, 0] -> [1, 1, 0, 1];
    [c, 0, 1, 0, x, 0, 0, 0, 0] -> [1, 1, 0, 1]; "T2
    [c, 1, 1, 0, x, 0, 0, 0, 0] -> [0, 1, 0, 1];
    [c, 0, 0, 0, x, 0, 0, 0, 0] -> [0, 1, 0, 1]; "T2P
    [c, 1, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1];
    [c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1]; "T2P
    [c, 1, 0, 0, 0, 0, 0, 0, 0] -> [1, 1, 0, 1];
    [c, 0, 1, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 1]; "T1P
    [c, 1, 1, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1];
    [c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1]; "T2P
    [c, 1, 0, 0, 0, 0, 0, 0, 0] -> [1, 1, 0, 1];
    [c, 0, 1, 1, 0, 0, 0, 0, 0] -> [1, 1, 0, 1]; "T1P
    [c, 1, 1, 1, 0, 0, 0, 0, 0] -> [1, 1, 0, 1];
    [c, 0, 0, 1, 0, 0, 0, 0, 0] -> [1, 1, 0, 1]; "T2P
    [c, 1, 0, 1, 0, 0, 0, 0, 0] -> [0, 0, 0, 1];
    [c, 0, 0, 1, 0, 0, 0, 0, 0] -> [0, 0, 0, 1]; "T2p
    [c, 1, 0, 0, 0, 0, 0, 0, 0] -> [1, 1, 0, 1];
    [c, 0, 1, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 1]; "T1P
    [c, 1, 1, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1];
    [c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1]; "T2P
    [c, 1, 0, 0, 1, 1, 0, 0, 0] -> [1, 1, 0, 1];
    [c, 0, 1, 0, 1, 1, 0, 0, 0] -> [1, 1, 0, 1]; "T1P
    [c, 1, 1, 0, 1, 1, 0, 0, 0] -> [1, 1, 0, 1];
    [c, 0, 1, 0, 1, 1, 0, 0, 0] -> [1, 1, 0, 1]; "T2
    [c, 1, 1, 0, 1, 1, 0, 0, 0] -> [1, 1, 0, 1];
    [c, 0, 1, 0, 1, 0, 0, 0, 0] -> [1, 1, 0, 1]; "T2
    [c, 1, 1, 0, 1, 0, 0, 0, 0] -> [0, 1, 0, 1];
    [c, 0, 0, 0, x, 0, 0, 0, 0] -> [0, 1, 0, 1]; "T2P
    [c, 1, 0, 0, 1, 0, 0, 0, 0] -> [0, 0, 0, 1];
    [c, 0, 0, 1, 1, 1, 0, 0, 0] -> [0, 0, 0, 1]; "T2P
    [c, 1, 0, 0, 1, 1, 0, 0, 0] -> [1, 1, 0, 1];
    [c, 0, 1, 1, 1, 1, 0, 0, 0] -> [1, 1, 0, 1]; "T1P
    [c, 1, 1, 1, 1, 1, 0, 0, 0] -> [1, 1, 0, 1];
```

240725-A1

PAL Codes: DRAM 2 (Continued)

```
[c, 0, 1, 1, 1, 0, 0, 0, 0] -> [1, 1, 0, 1]; "T2
[c, 1, 1, 1, 1, 0, 0, 0, 0] -> [0, 1, 0, 1];
[c, 0, 0, 1, 1, 0, 0, 0, 0] -> [0, 1, 0, 1]; "T2P
[c, 1, 0, 1, 0, 0, 0, 0, 0] -> [0, 0, 0, 1];
[c, 0, 0, 1, 0, 0, 0, 0, 0] -> [0, 0, 0, 1]; "T2P
[c, 1, 0, 0, 0, 0, 0, 0, 0] -> [1, 1, 0, 1];
[c, 0, 1, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 1]; "T1P
[c, 1, 1, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1];
[c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1]; "T2P
[c, 1, 0, 0, 0, 0, 0, 0, 0] -> [1, 1, 0, 1];
[c, 0, 1, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 1]; "T1P
[c, 1, 1, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1];
[c, 0, 1, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1]; "T2i
[c, 1, 1, 0, 0, 0, 0, 0, 0] -> [1, 1, 0, 1];
[c, 0, 0, 0, x, 1, 0, 0, 0] -> [1, 1, 0, 1]; "T1
[c, 1, 0, 0, x, 1, 0, 0, 0] -> [1, 1, 0, 1];
[c, 0, 1, 1, x, 1, 0, 0, 0] -> [1, 1, 0, 1]; "T2
[c, 1, 1, 1, x, 1, 0, 0, 0] -> [1, 1, 0, 1];
[c, 0, 1, 1, x, 0, 0, 0, 0] -> [1, 1, 0, 1]; "T2
[c, 1, 1, 1, x, 0, 0, 0, 0] -> [0, 1, 0, 1];
[c, 0, 0, 1, x, 0, 0, 0, 0] -> [0, 1, 0, 1]; "T2P
[c, 1, 0, 1, 0, 0, 0, 0, 0] -> [0, 0, 0, 1];
[c, 0, 0, 1, 0, 0, 0, 0, 0] -> [0, 0, 0, 1]; "T2P
[c, 1, 0, 0, 0, 0, 0, 0, 0] -> [1, 1, 0, 1];
[c, 0, 1, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 1]; "T1P
[c, 1, 1, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1];
[c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1]; "T2P
[c, 1, 0, 0, 0, 0, 0, 0, 0] -> [1, 1, 0, 1];
[c, 0, 1, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 1]; "T1P
[c, 1, 1, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1];
[c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1]; "T2P
[c, 1, 0, 0, 0, 0, 1, 0, 0] -> [1, 1, 1, 1];
[c, 0, 1, 0, 0, 1, 1, 0, 0] -> [1, 1, 1, 0]; "T1P
[c, 1, 1, 0, 0, 1, 1, 0, 0] -> [1, 1, 1, 0];
[c, 0, 1, 0, 0, 1, 1, 0, 0] -> [1, 1, 1, 0]; "T2
[c, 1, 1, 0, 0, 1, 1, 0, 0] -> [1, 1, 1, 0];
[c, 0, 1, 0, 0, 0, 1, 0, 0] -> [1, 1, 1, 0]; "T2
[c, 1, 1, 0, 0, 0, 1, 0, 0] -> [1, 1, 1, 0];
[c, 0, 1, 0, 0, 0, 1, 0, 0] -> [1, 1, 1, 1]; "T2
[c, 1, 1, 0, 0, 0, 0, 0, 0] -> [1, 1, 0, 1];
[c, 0, 1, 0, 0, 1, 0, 0, 0] -> [1, 1, 0, 1]; "T2
[c, 1, 1, 0, 0, 1, 0, 0, 0] -> [1, 1, 0, 1];
[c, 0, 1, 0, 0, 1, 0, 0, 0] -> [1, 1, 0, 1]; "T2
[c, 1, 1, 0, 0, 1, 0, 0, 0] -> [1, 1, 0, 1];
[c, 0, 1, 0, 0, 0, 0, 0, 0] -> [1, 1, 0, 1]; "T2
[c, 1, 1, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 1];
[c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 1]; "T2P
[c, 1, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1];
[c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1]; "T2P
[c, 1, 0, 0, 0, 0, 0, 0, 0] -> [1, 1, 0, 1];
[c, 0, 1, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 1]; "T1P
[c, 1, 1, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1];
[c, 0, 0, 0, 0, 0, 0, 0, 0] -> [0, 0, 0, 1]; "T2P
[c, x, x, 0, x, x, x, 1, 1] -> [1, 1, 0, 1]; "Cache eanbled
```

240725-A2

PAL Codes: DRAM 2 (Continued)

5

```
    [c, x, x, 0, x, x, x, 1, 1] -> [1, 1, 0, 1];
    [c, 0, 1, 0, x, 1, 0, 0, 1] -> [1, 1, 0, 1]; "Ti
    [c, 1, 1, 0, x, 1, 0, 0, 1] -> [1, 1, 0, 1];
    [c, 0, 0, 0, x, 1, 0, 0, 1] -> [1, 1, 0, 1]; "T1, Read
    [c, 1, 0, 0, x, 1, 0, 0, 1] -> [1, 1, 0, 1];
    [c, 0, 1, 0, x, 0, 0, 0, 1] -> [1, 1, 0, 1]; "T2
    [c, 1, 1, 0, x, 0, 0, 0, 1] -> [0, 1, 0, 1];
    [c, 0, 0, 0, x, 0, 0, 0, 1] -> [0, 1, 0, 1]; "T2P
    [c, 1, 0, 0, 0, 0, 0, 0, 1] -> [0, 1, 0, 1];
    [c, 0, 0, 0, 0, 0, 0, 0, 1] -> [0, 1, 0, 1]; "T2P
    [c, 1, 0, 0, 0, 0, 0, 0, 1] -> [0, 0, 0, 1];
    [c, 0, 0, 0, 0, 0, 0, 0, 1] -> [0, 0, 0, 1]; "T2P
    [c, 1, 0, 0, 0, 0, 0, 0, 1] -> [1, 1, 0, 1];
    [c, 0, 1, 0, 0, 0, 0, 0, 1] -> [0, 1, 0, 1]; "T1P, Read
    [c, 1, 1, 0, 0, 0, 0, 0, 1] -> [0, 1, 0, 1];
    [c, 0, 0, 0, 0, 0, 0, 0, 1] -> [0, 1, 0, 1]; "T2P
    [c, 1, 0, 0, 0, 0, 0, 0, 1] -> [0, 0, 0, 1];
    [c, 0, 0, 0, 0, 0, 0, 0, 1] -> [0, 0, 0, 1]; "T2P
    [c, 1, 0, 0, 0, 0, 0, 0, 1] -> [1, 1, 0, 1];
    [c, 0, 1, 1, 0, 0, 0, 0, 1] -> [1, 1, 0, 1]; "T1P, Write
    [c, 1, 1, 1, 0, 0, 0, 0, 1] -> [1, 1, 0, 1];
    [c, 0, 0, 1, 0, 0, 0, 0, 1] -> [1, 1, 0, 1]; "T2P
    [c, 1, 0, 1, 0, 0, 0, 0, 1] -> [0, 0, 0, 1];
    [c, 0, 0, 1, 0, 0, 0, 0, 1] -> [0, 0, 0, 1]; "T2p
    [c, 1, 0, 0, 0, 0, 0, 0, 1] -> [1, 1, 0, 1];

end PAGE_MODE_DRAM_CTRL_2;
^Z
```

240725-A3

PAL Codes: DRAM 2 (Continued)

```
module          PAGE_MODE_DRAM_CTRL_3  flag '-r3'

title   'PAGE MODE DRAM CONTROLLER - PAL 3, INTEL CORPORATION'

    PAGE3    device      'P16R8';

    x       =     .X.;          " ABEL 'don't care' symbol
    c       =     .C.;          " ABEL 'clocking input' symbol

" Inputs

    CLK2    pin   1;   "80386 CLK2
    CLK     pin   2;   "Processor Clock
    ADS~    pin   3;   "Address Strobe
    MEMCS~  pin   4;   "Memory Chip Select
    WR      pin   5;   "Write/Read
    READY~  pin   6;   "System Ready
    DRMRDY~ pin   7;   "DRAM Ready
    unused1 pin   8;
    RESET   pin   9;   "System Reset

" Outputs

    T2X~    pin  12;    "active during T2, T2p, and T2i
    T1P~    pin  13;    "active during T1p
    WE~     pin  14;    "DRAM Write Enable
    DEN~    pin  15;    "DRAM Data Bus Transceiver Enable
    DTR     pin  16;    "DRAM Data Bus Transceiver R/W# Direction signal
    lwr     pin  17;    "Latched Write/Read
    mreq    pin  18;    "Latched Memory Chip Select
    unused2 pin  19;    "

state_diagram [T2X~, T1P~]

    state [1, 1]:    if (CLK & !ADS~) then [0, 1] else [1, 1];
    state [0, 1]:    if RESET then [1, 1] else
                  if (CLK & !ADS~ & !READY~) then [1, 0] else
                  if (CLK & ADS~ & !READY~) then [1, 1] else [0, 1];
    state [1, 0]:    if RESET then [1, 1] else
                  if (CLK) then [0, 1] else [1, 0];
    state [0, 0]:    goto [1, 1];

state_diagram [WE~]

    state [1]:      if (CLK & !MEMCS~ & WR & T2X~ #
                  lwr & !T1P~) then [0] else [1];
    state [0]:      if (RESET) then [1] else
                if (CLK & !READY~) then [1] else [0];

state_diagram [DEN~]

    state [1]:      if (CLK & !MEMCS~ & !WR & T2X~ #
                  mreq & !T2X~ #
                  CLK & mreq & !T1P~) then [0] else [1];
```

240725-A4

PAL Codes: DRAM 3

5

```
    state [0]:      if RESET then [1] else
                if (CLK & !READY~) then [1] else [0];

state_diagram [DTR]

    state [1]:      if (CLK & !MEMCS~ & WR & T2X~ #
                    1wr & !T1P~) then [0] else [1];
    state [0]:      if (RESET) then [1] else
                if (!CLK & DEN~ & !1wr) then [1] else [0];

state_diagram [1wr]

    state [0]:      if (CLK & !MEMCS~ & WR) then [1] else [0];
    state [1]:      if (RESET) then [0] else
                if (!READY~ & MEMCS~ #
                    !READY~ & !WR) then [0] else [1];

state_diagram [mreq]

    state [0]:      if (CLK & !MEMCS~) then [1] else [0];
    state [1]:      if RESET then [0] else
                if (!READY~ & MEMCS~) then [0] else [1];

test_vectors

    ([CLK2,CLK,ADS~,WR,MEMCS~,READY~,RESET] ->
     [T2X~,T1P~,DEN~,1wr,WE~,DTR, mreq])

"    C  C  A  W  M  R  R      T  T  D  1  W  D  m
"    L  L  D  R  E  E  E      2  1  E  w  E  T  r
"    K  K  S     M  A  S      X  P  N  r  ~  R  e
"    2     ~     C  D  E      ~  ~  ~           q
"                S  Y  T
"                ~  ~

    [c, x, x, x, x, x, 1] -> [1, 1, 1, 0, 1, 1, x];
    [c, x, x, x, x, x, 1] -> [1, 1, 1, 0, 1, 1, 0];
    [c, 1, 1, x, 1, 1, 0] -> [1, 1, 1, 0, 1, 1, 0];
    [c, 0, 1, x, 1, 1, 0] -> [1, 1, 1, 0, 1, 1, 0]; "Ti
    [c, 1, 1, x, 1, 1, 0] -> [1, 1, 1, 0, 1, 1, 0];
    [c, 0, 0, 0, 0, 1, 0] -> [1, 1, 1, 0, 1, 1, 0]; "T1
    [c, 1, 0, 0, 0, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
    [c, 0, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2
    [c, 1, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
    [c, 0, 0, 0, 0, 1, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2
    [c, 1, 0, 0, 0, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
    [c, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2P
    [c, 1, 0, 0, 0, 0, 0] -> [1, 0, 1, 0, 1, 1, 1];
    [c, 0, 1, 0, 1, 1, 0] -> [1, 0, 1, 0, 1, 1, 1]; "T1P
    [c, 1, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
    [c, 0, 0, 1, 0, 0, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2P
    [c, 1, 0, 1, 0, 0, 0] -> [1, 0, 1, 1, 1, 1, 1];
    [c, 0, 1, 1, 1, 1, 0] -> [1, 0, 1, 1, 0, 0, 1]; "T1P
    [c, 1, 1, 1, 1, 1, 0] -> [0, 1, 0, 1, 0, 0, 1];
```

240725-A5

PAL Codes: DRAM 3 (Continued)

```
[c, 0, 0, 0, 0, 1, 0] -> [0, 1, 0, 1, 0, 0, 1]; "T2P
[c, 1, 0, 0, 0, 1, 0] -> [0, 1, 0, 1, 0, 0, 1];
[c, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 0, 0, 0, 1]; "T2P
[c, 1, 0, 0, 0, 0, 0] -> [1, 0, 1, 0, 1, 0, 1];
[c, 0, 1, 0, 1, 1, 0] -> [1, 0, 1, 0, 1, 1, 1]; "T1P
[c, 1, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
[c, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2P
[c, 1, 0, 0, 0, 0, 0] -> [1, 0, 1, 0, 1, 1, 1];
[c, 0, 1, 0, 1, 1, 0] -> [1, 0, 1, 0, 1, 1, 1]; "T1p
[c, 1, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
[c, 0, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2
[c, 1, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
[c, 0, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2
[c, 1, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
[c, 0, 0, 1, 0, 1, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2P
[c, 1, 0, 1, 0, 1, 0] -> [0, 1, 0, 1, 1, 1, 1];
[c, 0, 0, 1, 0, 0, 0] -> [0, 1, 0, 1, 1, 1, 1]; "T2p
[c, 1, 0, 1, 0, 0, 0] -> [1, 0, 1, 1, 1, 1, 1];
[c, 0, 1, 1, 1, 1, 0] -> [1, 0, 1, 1, 0, 0, 1]; "T1P
[c, 1, 1, 1, 1, 1, 0] -> [0, 1, 0, 1, 0, 0, 1];
[c, 0, 1, 1, 1, 1, 0] -> [0, 1, 0, 1, 0, 0, 1]; "T2
[c, 1, 1, 1, 1, 1, 0] -> [0, 1, 0, 1, 0, 0, 1];
[c, 0, 0, 0, 0, 1, 0] -> [0, 1, 0, 1, 0, 0, 1]; "T2P
[c, 1, 0, 0, 0, 1, 0] -> [0, 1, 0, 1, 0, 0, 1];
[c, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 0, 0, 0, 1]; "T2P
[c, 1, 0, 0, 0, 0, 0] -> [1, 0, 1, 0, 1, 0, 1];
[c, 0, 1, 0, 1, 1, 0] -> [1, 0, 1, 0, 1, 1, 1]; "T1P
[c, 1, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
[c, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2P
[c, 1, 0, 0, 0, 0, 0] -> [1, 0, 1, 0, 1, 1, 1];
[c, 0, 1, 0, 1, 1, 0] -> [1, 0, 1, 0, 1, 1, 1]; "T1P
[c, 1, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
[c, 0, 1, 0, 1, 0, 0] -> [0, 1, 0, 0, 1, 1, 0]; "T2i
[c, 1, 1, 0, 1, 0, 0] -> [1, 1, 1, 0, 1, 1, 0];
[c, 0, 0, 1, 0, 1, 0] -> [1, 1, 1, 0, 1, 1, 0]; "T1
[c, 1, 0, 1, 0, 1, 0] -> [0, 1, 1, 1, 0, 0, 1];
[c, 0, 1, 1, 1, 1, 0] -> [0, 1, 0, 1, 0, 0, 1]; "T2
[c, 1, 1, 1, 1, 1, 0] -> [0, 1, 0, 1, 0, 0, 1];
[c, 0, 1, 1, 1, 1, 0] -> [0, 1, 0, 1, 0, 0, 1]; "T2
[c, 1, 1, 1, 1, 1, 0] -> [0, 1, 0, 1, 0, 0, 1];
[c, 0, 0, 0, 0, 1, 0] -> [0, 1, 0, 1, 0, 0, 1]; "T2P
[c, 1, 0, 0, 0, 1, 0] -> [0, 1, 0, 1, 0, 0, 1];
[c, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 0, 0, 0, 1]; "T2P
[c, 1, 0, 0, 0, 0, 0] -> [1, 0, 1, 0, 1, 0, 1];
[c, 0, 1, 0, 1, 1, 0] -> [1, 0, 1, 0, 1, 1, 1]; "T1P
[c, 1, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
[c, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2P
[c, 1, 0, 0, 0, 0, 0] -> [1, 0, 1, 0, 1, 1, 1];
[c, 0, 1, 0, 1, 1, 0] -> [1, 0, 1, 0, 1, 1, 1]; "T1P
[c, 1, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
[c, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2P
[c, 1, 0, 0, 0, 0, 0] -> [1, 0, 1, 0, 1, 1, 1];
[c, 0, 1, 0, 1, 1, 0] -> [1, 0, 1, 0, 1, 1, 1]; "T1P
[c, 1, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
```

240725-A6

```
    [c, 0, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2
    [c, 1, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
    [c, 0, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2
    [c, 1, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
    [c, 0, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2
    [c, 1, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
    [c, 0, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2
    [c, 1, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
    [c, 0, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2
    [c, 1, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
    [c, 0, 0, 0, 0, 1, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2P
    [c, 1, 0, 0, 0, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
    [c, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2P
    [c, 1, 0, 0, 0, 0, 0] -> [1, 0, 1, 0, 1, 1, 1];
    [c, 0, 1, 0, 1, 1, 0] -> [1, 0, 1, 0, 1, 1, 1]; "T1P
    [c, 1, 1, 0, 1, 1, 0] -> [0, 1, 0, 0, 1, 1, 1];
    [c, 0, 0, 0, 0, 0, 0] -> [0, 1, 0, 0, 1, 1, 1]; "T2P

end PAGE_MODE_DRAM_CTRL_3;
```

240725-A7

PAL Codes: DRAM 3 (Continued)

```
ABEL(tm) 3.10  -  Document Generator            14-Feb-90 09:54 AM
PAGE MODE DRAM CONTROLLER - PAL 3, INTEL CORPORATION
Equations for Module PAGE_MODE_DRAM_CTRL_3

Device PAGE3


- Reduced Equations:

    !T2X- := (CLK & !RESET & !T1P- & T2X-
              # READY- & !RESET & T1P- & !T2X-
              # !CLK & !RESET & T1P- & !T2X-
              # !ADS- & CLK & T1P- & T2X-);


    !T1P- := (!CLK & !RESET & !T1P- & T2X-
              # !ADS- & CLK & !READY- & !RESET & T1P- & !T2X-);


    !WE- := (READY- & !RESET & !WE-
             # !CLK & !RESET & !WE-
             # !T1P- & WE- & lwr
             # CLK & !MEMCS- & T2X- & WE- & WR);


    !DEN- := (!DEN- & READY- & !RESET
              # !CLK & !DEN- & !RESET
              # CLK & DEN- & !T1P- & mreq
              # DEN- & !T2X- & mreq
              # CLK & DEN- & !MEMCS- & T2X- & !WR);


    !DTR := (!DTR & !RESET & lwr
             # !DEN- & !DTR & !RESET
             # CLK & !DTR & !RESET
             # DTR & !T1P- & lwr
             # CLK & DTR & !MEMCS- & T2X- & WR);


    !lwr := (!READY- & !WR
             # MEMCS- & !READY-
             # RESET & lwr
             # !WR & !lwr
             # MEMCS- & !lwr
             # !CLK & !lwr);

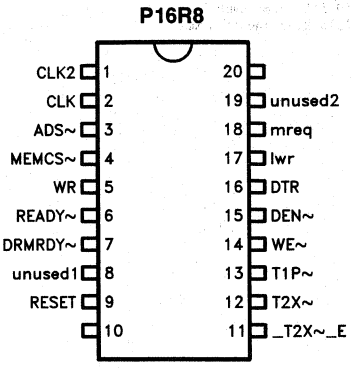

    !mreq := (MEMCS- & !READY-
              # RESET & mreq
              # MEMCS- & !mreq
              # !CLK & !mreq);
```

240725-A8

PAL Codes: DRAM 3 (Continued)

ABEL™ 3.10—Document Generator 14-Feb-90 09:54 AM
PAGE MODE DRAM CONTROLLER—PAL 3, INTEL CORPORATION
Chip diagram for Module PAGE_MODE_DRAM_CTRL_3

Device PAGE3



240725-67

PAL Codes: DRAM 3 (Continued)

5

```
module          PAGE_MODE_DRAM_CTRL_4  flag '-r3'

title   'PAGE MODE DRAM CONTROLLER - PAL 4, INTEL CORPORATION'

    PAGE4    device      'P16R8';

    x       =     .X.;        " ABEL 'don't care' symbol
    c       =     .C.;        " ABEL 'clocking input' symbol

" Inputs

    CLOCK   pin   1;
    D0      pin   2;
    D1      pin   3;
    D2      pin   4;
    D3      pin   5;
    D4      pin   6;
    D5      pin   7;
    D6      pin   8;
    D7      pin   9;
    OE      pin   11;

" Outputs

    A0      pin  12;
    A1      pin  13;
    A2      pin  14;
    A3      pin  15;
    A4      pin  16;
    A5      pin  17;
    A6      pin  18;
    A7      pin  19;

addr = [A7..A0];

equations

    addr := addr + 1;

end PAGE_MODE_DRAM_CTRL_4;
```

240725-A9

PAL Codes: DRAM 4

```
ABEL(tm) 3.10  -  Document Generator                    14-Feb-90 09:54 AM
PAGE MODE DRAM CONTROLLER - PAL 4, INTEL CORPORATION
Equations for Module PAGE_MODE_DRAM_CTRL_4

Device PAGE4

- Reduced Equations:

    !A7 := (A0 & A1 & A2 & A3 & A4 & A5 & A6 & A7
          # !A0 & !A7
          # !A1 & !A7
          # !A2 & !A7
          # !A3 & !A7
          # !A4 & !A7
          # !A5 & !A7
          # !A6 & !A7);


    !A6 := (A0 & A1 & A2 & A3 & A4 & A5 & A6
          # !A0 & !A6
          # !A1 & !A6
          # !A2 & !A6
          # !A3 & !A6
          # !A4 & !A6
          # !A5 & !A6);


    !A5 := (A0 & A1 & A2 & A3 & A4 & A5
          # !A0 & !A5
          # !A1 & !A5
          # !A2 & !A5
          # !A3 & !A5
          # !A4 & !A5);


    !A4 := (A0 & A1 & A2 & A3 & A4
          # !A0 & !A4
          # !A1 & !A4
          # !A2 & !A4
          # !A3 & !A4);

    !A3 := (A0 & A1 & A2 & A3 # !A0 & !A3 # !A1 & !A3 # !A2 & !A3);

    !A2 := (A0 & A1 & A2 # !A0 & !A2 # !A1 & !A2);

    !A1 := (A0 & A1 # !A0 & !A1);

    !A0 := (A0);
```

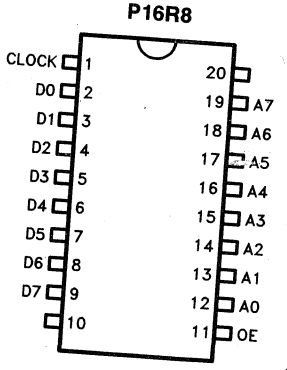
240725-B0

PAL Codes: DRAM 4 (Continued)

5

ABEL™ 3.10—Document Generator
PAGE MODE DRAM CONTROLLER—PAL 4, INTEL CORPORATION 14-Feb-90 09:54 AM
Chip diagram for Module PAGE_MODE_DRAM_CTRL_4

Device PAGE4



240725-68

end of module PAGE_MODE_DRAM_CTRL_4

PAL Codes: DRAM 4 (Continued)

```
module     IO_CTRL_1  flag '-r3'

title      'IO BUS CONTROLLER - PAL 1, INTEL CORPORATION'

    IO1    device      'P16R4';

    x          =     .X.;              " ABEL 'don't care' symbol
    c          =     .C.;              " ABEL 'clocking input' symbol

" Inputs

    CLK        pin   1;   "Processor Clock
    RESET      pin   2;   "System Reset
    MRDC~      pin   3;   "Memory (EPROM) Read Command
    IORC~      pin   4;   "I/O Read Command
    IOWC~      pin   5;   "I/O Write Command
    INTA~      pin   6;   "Interrupt Acknowledge
    DEN~       pin   7;   "I/O Bus Data Transceiver Enable
    IORDY~     pin   8;   "I/O-EPROM Ready
    L510CS~    pin   9;   "82510 Chip Select
    OEN~       pin  11;   "PAL output Enable
    L59CS~     pin  12;   "8259A-2 Chip Select
    LEPROM~    pin  13;   "EPROM Chip Select
    unused_0   pin  18;   "
    unused_1   pin  19;   "

" Outputs

    delay      pin  14;   "
    s2         pin  15;   "
    s1         pin  16;   "
    s0         pin  17;   "

    dstate     =  [delay, s2, s1, s0];
    idle       =  [  1  , 1 , 1 , 1 ];
    start      =  [  1  , 1 , 1 , 0 ];
    wait_14    =  [  1  , 0 , 1 , 0 ];
    wait_13    =  [  1  , 0 , 1 , 1 ];
    wait_12    =  [  1  , 0 , 0 , 0 ];
    wait_11    =  [  1  , 1 , 0 , 0 ];
    wait_10    =  [  1  , 1 , 0 , 1 ];
    active     =  [  0  , 1 , 1 , 1 ];


state_diagram dstate

    state idle:     if (!DEN~ & !MRDC~ # !DEN~ & !IORC~ #
                        !DEN~ & !IOWC~ # !DEN~ & !INTA~) then start
                    else idle;
    state start:    if (!L510CS~ & !IOWC~) then wait_14 else
                    if (!L510CS~ & !IORC~) then wait_13 else
                    if (!L59CS~ & !IOWC~) then wait_11 else
                    if (!LEPROM~ # !L59CS~ & !IORC~ # !INTA~) then wait_10;
    state wait_14: goto wait_13;
```

240725-B1

```
    state wait_13: goto wait_12;
    state wait_12: goto wait_11;
    state wait_11: goto wait_10;
    state wait_10: goto active;
    state active:  if !IORDY~ then idle else active;

end IO_CTRL_1;
^Z
```

240725-B2

PAL Codes: IO-1

5

```
ABEL(tm) 3.10  -  Document Generator            15-Feb-90 06:40 PM
IO BUS CONTROLLER - PAL 1, INTEL CORPORATION
Equations for Module IO_CTRL_1

Device IO1


- Reduced Equations:

    !delay := (IORDY~ & !delay & s0 & s1 & s2 # delay & s0 & !s1 & s2);

    !s2 := (delay & s1 & !s2
            # !IORC~ & !L51OCS~ & delay & !s0 & s1
            # !IOWC~ & !L51OCS~ & delay & !s0 & s1);


    !s1 := (delay & !s0 & !s1
            # delay & s0 & s1 & !s2
            # !INTA~ & IORC~ & IOWC~ & delay & !s0 & s2
            # IORC~ & IOWC~ & !LEPROM~ & delay & !s0 & s2
            # !IORC~ & L51OCS~ & !L59CS~ & delay & !s0 & s2
            # !INTA~ & L51OCS~ & delay & !s0 & s2
            # L51OCS~ & !LEPROM~ & delay & !s0 & s2
            # !IOWC~ & L51OCS~ & !L59CS~ & delay & !s0 & s2);

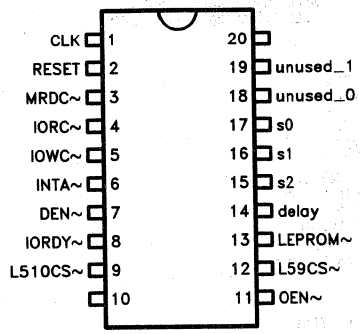

    !s0 := (delay & !s0 & !s1 & !s2
            # delay & s0 & s1 & !s2
            # !IOWC~ & !L59CS~ & delay & !s0 & s1 & s2
            # !IOWC~ & !L51OCS~ & delay & !s0 & s1 & s2
            # !DEN~ & !INTA~ & delay & s0 & s1
            # !DEN~ & !IOWC~ & delay & s0 & s1
            # !DEN~ & !IORC~ & delay & s0 & s1
            # !DEN~ & !MRDC~ & delay & s0 & s1);
```

240725-B3

PAL Codes: IO-1 (Continued)

ABEL™ 3.10—Document Generator 15-Feb-90 06:40 PM
IO BUS CONTROLLER—PAL 1, INTEL CORPORATION
Chip diagram for Module IO__CTRL__1

Device IO1



240725-69

end of module IO__CTRL__1

PAL Codes: IO-1 (Continued)

5

```
module     IO_CTRL_2  flag '-r3'

title     'IO BUS CONTROLLER - PAL 2, INTEL CORPORATION'

    IO2    device      'P16R6';

    x           =     .X.;              " ABEL 'don't care' symbol
    c           =     .C.;              " ABEL 'clocking input' symbol

" Inputs

    CLK         pin   1;   "Processor Clock
    RESET       pin   2;   "System Reset
    LMIO        pin   3;   "Latched M/IO#
    LDC         pin   4;   "Latched D/C#
    LWR         pin   5;   "Latched W/R#
    LALE        pin   6;   "Latched ALE
    L510CS~     pin   7;   "82510 Chip Select
    L59CS~      pin   8;   "8259A-2 Chip Select
    LEPROM~     pin   9;   "EPROM Chip Select
    OEN~        pin  11;   "PAL Output Enable
    rdy~        pin  12;   "I/O-EPROM Ready (n-1)
    rdy510~     pin  19;   "I/O-EPROM Ready (n-2)

" Outputs

    recovery    pin  13;   "I/O Recovery Time
    s1          pin  14;   "
    s0          pin  15;   "
    IORC~       pin  16;   "I/O Read Command
    IOWC~       pin  17;   "I/O Write Command
    MRDC~       pin  18;   "Memory (EPROM) Read Command

    rstate      =  [recovery, s1, s0];
    idle        =  [  0     , 1 , 0 ];
    active      =  [  0     , 1 , 1 ];
    inactive_0  =  [  1     , 1 , 1 ];
    inactive_1  =  [  1     , 0 , 1 ];
    inactive_2  =  [  1     , 0 , 0 ];
    inactive_3  =  [  1     , 1 , 0 ];
    illegal_a   =  [  0     , 0 , 0 ];
    illegal_b   =  [  0     , 0 , 1 ];


state_diagram rstate

    state idle:         if (!IORC~ # !IOWC~) then active else idle;
    state active:       if (IORC~ # IOWC~) then inactive_0 else active;
    state inactive_0:   goto inactive_1;
    state inactive_1:   goto inactive_2;
    state inactive_2:   goto inactive_3;
    state inactive_3:   goto idle;
    state illegal_a:    goto idle;
    state illegal_b:    goto idle;
```

240725-B4

```
state_diagram [IOWC~]

    state [1]: if (!recovery & !LMIO & LDC & LWR & (!L510CS~ # !L59CS~))
               then [0] else [1];
    state [0]: if RESET then [1] else
               if (!L510CS~ & !rdy510~ # !rdy~) then [1] else [0];

state_diagram [IORC~]

    state [1]: if (!recovery & !LMIO & LDC & !LWR & (!L510CS~ # !L59CS~))
               then [0] else [1];
    state [0]: if RESET then [1] else
               if !rdy~ then [1] else [0];

state_diagram [MRDC~]

    state [1]: if (LALE & LMIO & !LWR & !LEPROM~) then [0] else [1];
    state [0]: if RESET then [1] else
               if !rdy~ then [1] else [0];

end IO_CTRL_2;

^Z
```

240725-B5

PAL Codes: IO-2

```
ABEL(tm) 3.10  -  Document Generator            14-Feb-90 09:34 AM
IO BUS CONTROLLER - PAL 2, INTEL CORPORATION
Equations for Module IO_CTRL_2

Device IO2


- Reduced Equations:

    !recovery := (!recovery & !s1 # !IORC~ & !IOWC~ & !recovery # !s0 & s1);

    !s1 := (recovery & s0);

    !s0 := (recovery & !s0 # !s1 # IORC~ & IOWC~ & !s0);

    !IOWC~ := (!IOWC~ & !RESET & rdy510~ & rdy~
              # !IOWC~ & L510CS~ & !RESET & rdy~
              # IOWC~ & !L59CS~ & LDC & !LMIO & LWR & !recovery
              # IOWC~ & !L510CS~ & LDC & !LMIO & LWR & !recovery);


    !IORC~ := (!IORC~ & !RESET & rdy~
              # IORC~ & !L59CS~ & LDC & !LMIO & !LWR & !recovery
              # IORC~ & !L510CS~ & LDC & !LMIO & !LWR & !recovery);


    !MRDC~ := (!MRDC~ & !RESET & rdy~
              # LALE & !LEPROM~ & LMIO & !LWR & MRDC~);
```

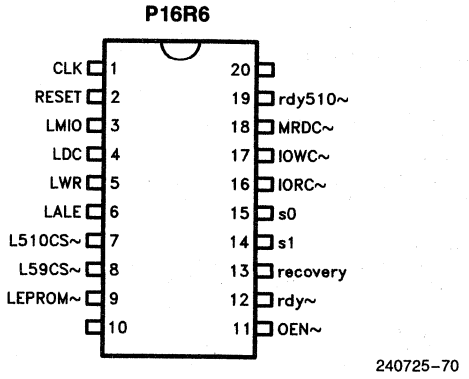
240725-B6

PAL Codes: IO-2 (Continued)

5

ABEL™ 3.10—Document Generator 14-Feb-90 09:34 AM
IO BUS CONTROLLER—PAL 2, INTEL CORPORATION
Chip diagram for Module IO__CTRL__2

Device IO2



end of module IO__CTRL__2

PAL Codes: IO-2 (Continued)

```
module     IO_CTRL_3  flag '-r3'

title     'IO BUS CONTROLLER - PAL 2, INTEL CORPORATION'

    IO3    device      'P16R6';

    x           =     .X.;              " ABEL 'don't care' symbol
    c           =     .C.;              " ABEL 'clocking input' symbol

" Inputs

    CLK         pin   1;   "Processor Clock
    RESET       pin   2;   "System Reset
    LMIO        pin   3;   "Latched M/IO#
    LDC         pin   4;   "Latched D/C#
    LWR         pin   5;   "Latched W/R#
    LALE        pin   6;   "Latched ALE
    L510CS~     pin   7;   "82510 Chip Select
    L59CS~      pin   8;   "8259A-2 Chip Select
    LEPROM~     pin   9;   "EPROM Chip Select
    OEN~        pin  11;   "PAL Output Enable
    rdy~        pin  12;   "I/O-EPROM Ready (n-1)
    IORDY~      pin  19;   "I/O-EPROM Ready

" Outputs

    INTA~       pin  13;   "Interrupt Acknowledge
    st0         pin  14;   "
    DEN~        pin  15;   "I/O Bus Transceiver Enable
    st1         pin  16;   "
    DTR         pin  17;   "I/O Bus Transceiver Direction
    st2         pin  18;   "

state_diagram [INTA~, st0]

    state [1, 1]:  if (!LMIO & !LDC & !LWR & LALE) then [1, 0] else [1, 1];
    state [1, 0]:  if RESET then [1, 1] else
                   if !LALE then [0, 0] else [1, 0];
    state [0, 0]:  if RESET then [1, 1] else
                   if !rdy~ then [1, 1] else [0, 0];
    state [0, 1]:  goto [1, 1];

state_diagram [DEN~, st1]

    state [1, 1]:  if LALE & (!LEPROM~ # !L510CS~ # !L59CS~) then [1, 0] else

                   if !INTA~ then [0, 0] else [1, 1];
    state [1, 0]:  if RESET then [1, 1] else
                   if !LALE then [0, 0] else [1, 0];
    state [0, 0]:  if RESET then [1, 1] else
                   if !rdy~ then [1, 1] else [0, 0];
    state [0, 1]:  goto [1, 1];

state_diagram [DTR, st2]
```

240725-B7

```
    state [1, 1]:  if LALE & (!LEPROM~ # !L510CS~ # !L59CS~) & LWR then [0, 1]

                   else [1, 1];
    state [0, 1]:  if RESET then [1, 1] else
                   if !IORDY~ then [0, 0] else [0, 1];
    state [0, 0]:  goto [1, 1];
    state [1, 0]:  goto [1, 1];

end IO_CTRL_3;
^Z
```

240725-B8

PAL Codes: IO-3

```
ABEL(tm) 3.10  -  Document Generator            15-Feb-90 06:45 PM
IO BUS CONTROLLER - PAL 2, INTEL CORPORATION
Equations for Module IO_CTRL_3

Device IO3


- Reduced Equations:

    !INTA~ := (!INTA~ & !RESET & rdy~ & !st0
              # INTA~ & !LALE & !RESET & !st0);


    !st0 := (!RESET & rdy~ & !st0
             # INTA~ & !RESET & !st0
             # INTA~ & LALE & !LDC & !LM1O & !LWR & st0);


    !DEN~ := (!DEN~ & !RESET & rdy~ & !st1
              # DEN~ & !LALE & !RESET & !st1
              # DEN~ & !INTA~ & L51OCS~ & L59CS~ & LEPROM~ & st1
              # DEN~ & !INTA~ & !LALE & st1);


    !st1 := (!RESET & rdy~ & !st1
             # DEN~ & !RESET & !st1
             # DEN~ & !INTA~ & st1
             # DEN~ & !L59CS~ & LALE & st1
             # DEN~ & !L51OCS~ & LALE & st1
             # DEN~ & LALE & !LEPROM~ & st1);


    !DTR := (!DTR & !RESET & st2
             # DTR & !L59CS~ & LALE & LWR & st2
             # DTR & !L51OCS~ & LALE & LWR & st2
             # DTR & LALE & !LEPROM~ & LWR & st2);

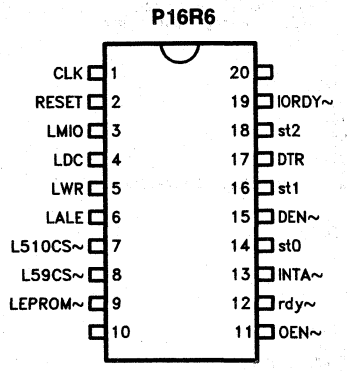

    !st2 := (!DTR & !IORDY~ & !RESET & st2);
```

240725-B9

PAL Codes: IO-3 (Continued)

ABEL™ 3.10—Document Generator 15-Feb-90 06:45 PM
IO BUS CONTROLLER—PAL 2, INTEL CORPORATION
Chip diagram for Module IO__CTRL__3

Device IO3



240725-71

end of module IO__CTRL__3

PAL Codes: IO-3 (Continued)

5

```
module     IO_CTRL_4  flag '-r3'

title      'IO BUS CONTROLLER - PAL 2, INTEL CORPORATION'

    IO4    device      'P16R6';

    x           =     .X.;              " ABEL 'don't care' symbol
    c           =     .C.;              " ABEL 'clocking input' symbol

" Inputs

    CLK         pin   1;   "Processor Clock
    RESET       pin   2;   "System Reset
    LMIO        pin   3;   "Latched M/IO#
    LDC         pin   4;   "Latched D/C#
    LWR         pin   5;   "Latched W/R#
    LALE        pin   6;   "Latched ALE
    delay       pin   7;   "Delay Signal for Wait State Generation
    unused_0    pin   8;   "
    unused_1    pin   9;   "
    OEN~        pin  11;   "PAL Output Enable
    unused_3    pin  12;   "
    unused_4    pin  19;   "

" Outputs

    IORDY~      pin  13;   "I/O-EPROM Ready
    rdy~        pin  14;   "I/O-EPROM Ready (n-1)
    rdy510~     pin  15;   "I/O-EPROM Ready (n-2)
    nc_0        pin  16;   "
    nc_1        pin  17;   "
    nc_2        pin  18;   "

    rstate      =  [IORDY~, rdy~, rdy510~];
    idle        =  [   1  ,  1  ,    1   ];
    rdy2        =  [   1  ,  1  ,    0   ];
    rdy1        =  [   1  ,  0  ,    1   ];
    rdy0        =  [   0  ,  1  ,    1   ];
    illegal_a   =  [   1  ,  0  ,    0   ];
    illegal_b   =  [   0  ,  0  ,    0   ];
    illegal_c   =  [   0  ,  0  ,    1   ];
    illegal_d   =  [   0  ,  1  ,    0   ];

state_diagram rstate

    state idle:         if (LMIO & !LDC & LWR & LALE) then rdy1 else
                        if !delay then rdy2 else idle;
    state rdy2:         if RESET then idle else rdy1;
    state rdy1:         if RESET then idle else
                        if !LALE then rdy0 else rdy1;
    state rdy0:         goto idle;
    state illegal_a:    goto idle;
    state illegal_b:    goto idle;
    state illegal_c:    goto idle;
```

240725-C0

PAL Codes: IO-4

```
    state illegal_d:    goto idle;

end IO_CTRL_4;
^Z
```

240725-C1

```
ABEL(tm) 3.10  -  Document Generator            15-Feb-90 06:55 PM
IO BUS CONTROLLER - PAL 2, INTEL CORPORATION
Equations for Module IO_CTRL_4

Device IO4


- Reduced Equations:

    !IORDY- := (IORDY- & !LALE & !RESET & rdy510- & !rdy-);

    !rdy- := (IORDY- & LALE & !RESET & rdy510- & !rdy-
              # IORDY- & !RESET & !rdy510- & rdy-
              # IORDY- & LALE & !LDC & LMIO & LWR & rdy510- & rdy-);


    !rdy510- := (IORDY- & !LALE & !delay & rdy510- & rdy-
                 # IORDY- & !LWR & !delay & rdy510- & rdy-
                 # IORDY- & LDC & !delay & rdy510- & rdy-
                 # IORDY- & !LMIO & !delay & rdy510- & rdy-);
```

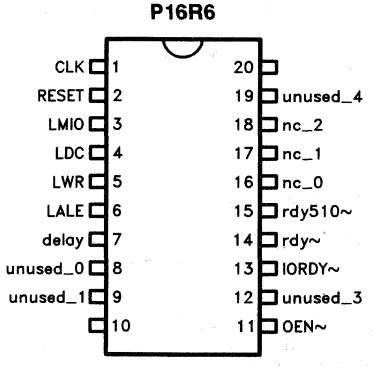
240725-C2

PAL Codes: IO-4 (Continued)

5

ABEL™ 3.10—Document Generator 15-Feb-90 06:55 PM
IO BUS CONTROLLER—PAL 2, INTEL CORPORATION
Chip diagram for Module IO__CTRL__4

Device IO4



240725-72

end of module IO__CTRL__4

PAL Codes: IO-4 (Continued)

```
module          LADDR_DEC flag '-r3'

title   'LOCAL_DECODE_LOGIC - INTEL CORPORATION'

    LADDR_PAL     device        'P16L8';

    x = .X.;              "ABEL don't care symbol
    c = .C.;              "ABEL clocking input symbol
    h = 1;                "logic 1
    l = 0;                "logic 0

" Inputs

    ADS~     pin   1;    "ADS#
    M_IO~    pin   2;    "M/IO#
    A31      pin   3;    "Addr bit 31
    A30            pin   4;    "Addr bit 30
    A29      pin   5;    "Addr bit 29

" Outputs

    X16~     pin  18;    "indicates a 16-bit access
    LBA~     pin  17;    "local bus access
    NCA~     pin  16;    "non-cache access

equations

    !X16~ =  !ADS~ & M_IO~ & A31 & A30 & A29;
    LBA~  =  h;
    NCA~  =  h;


end LADDR_DEC;
```

240725-C3

```
ABEL(tm) 3.10  -  Document Generator            14-Feb-90 09:51 AM
LOCAL_DECODE_LOGIC - INTEL CORPORATION
Equations for Module LADDR_DEC

Device LADDR_PAL


- Reduced Equations:

    !X16~ = (A29 & A30 & A31 & !ADS~ & M_IO~);

    !LBA~ = (0);

    !NCA~ = (0);
```

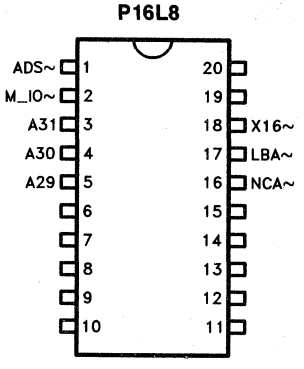
240725-C4

PAL Codes: Local Decoder

5

ABEL™ 3.10—Document Generator 14-Feb-90 09:51 AM
LOCAL_DECODE_LOGIC—INTEL CORPORATION
Chip diagram for Module LADDR_DEC

Device LADDR_PAL



240725-73

end of module LADDR_DEC

PAL Codes: Local Decoder (Continued)

```
module          READY flag '-r3'

title   'READY_LOGIC - INTEL CORPORATION'

    RDY     device       'P16L8';

" Inputs

    DRAMRDY-        pin   1;   "DRAM READY#
    IORDY-    pin   2;   "IO/EPROM READY#
    RDYEN-    pin   3;   "RDYEN# of 82385
    RDY385-   pin   4;   "READYO# OF 82385
    RDY387-   pin   5;   "READYO# OF 82387
    CACHE     pin   6;   "High if cache exits; otherwise, Low

" Outputs

    READY-  pin  12;   "READY# for 80386
    BREADY- pin  13;   "BREADY# for 82385

equations

    !BREADY- = !DRAMRDY- # !IORDY-;
    !READY- = (CACHE & !RDY385-) # !RDY387- #
              (CACHE & !RDYEN- & (!DRAMRDY- # !IORDY-) #
              !CACHE & (!DRAMRDY- # !IORDY-));

end READY;
```

240725-C5

```
ABEL(tm) 3.10  -  Document Generator            15-Feb-90 07:02 PM
READY_LOGIC - INTEL CORPORATION
Equations for Module READY

Device RDY


- Reduced Equations:

    !BREADY- = (!IORDY- # !DRAMRDY-);

    !READY- = (!CACHE & !IORDY-
               # !CACHE & !DRAMRDY-
               # !IORDY- & !RDYEN-
               # !DRAMRDY- & !RDYEN-
               # !RDY387-
               # CACHE & !RDY385-);
```

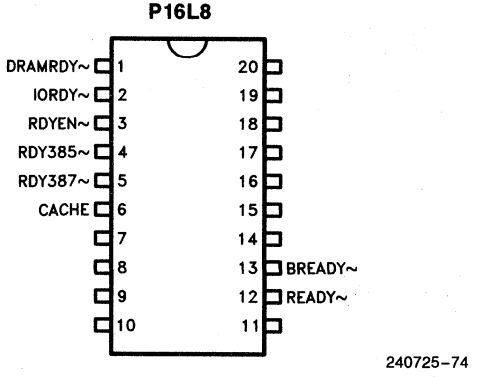
240725-C6

PAL Codes: Ready

5

ABEL™ 3.10—Document Generator 15-Feb-90 07:02 PM
READY_LOGIC—INTEL CORPORATION
Chip diagram for Module READY

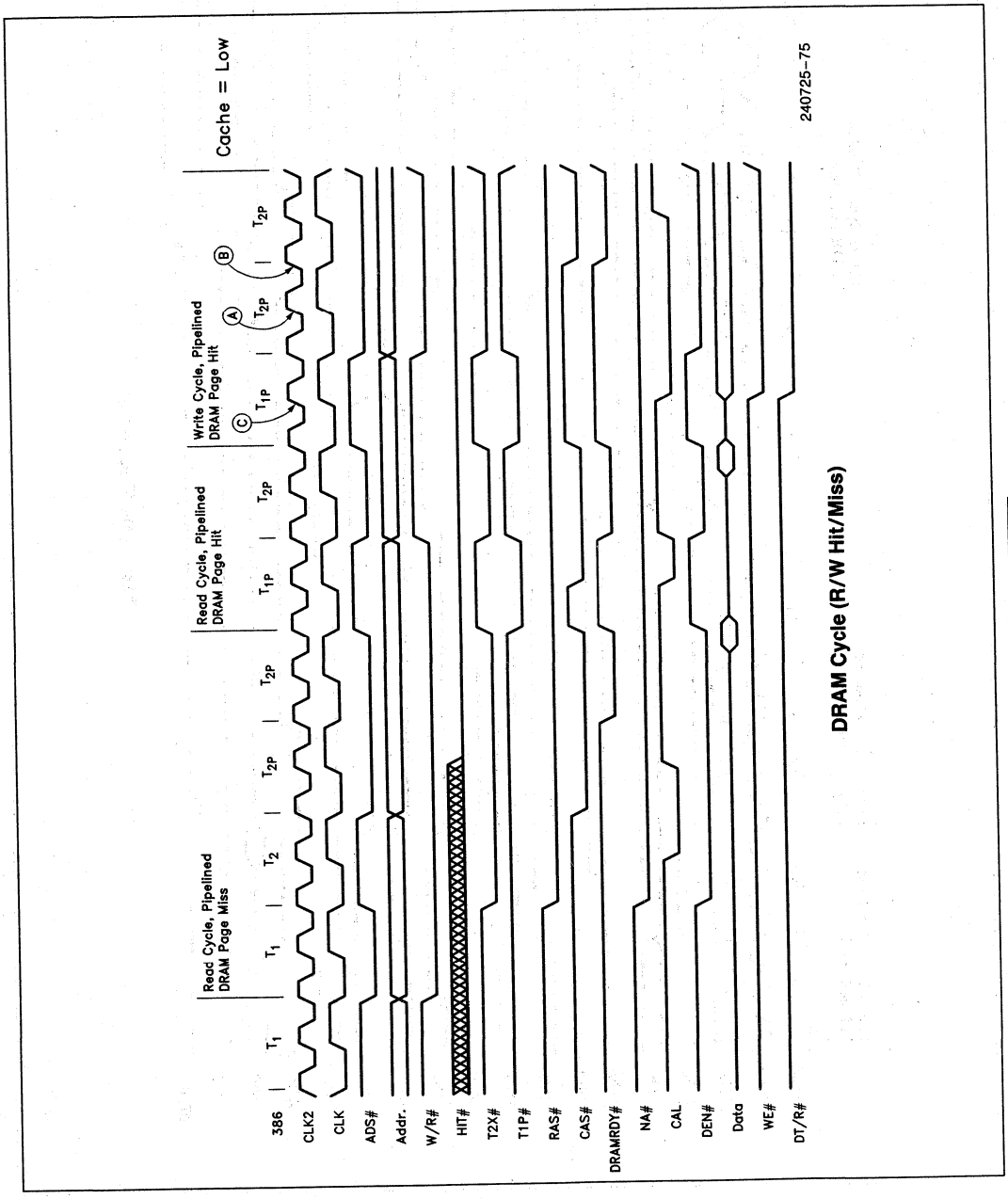
Device RDY



end of module READY

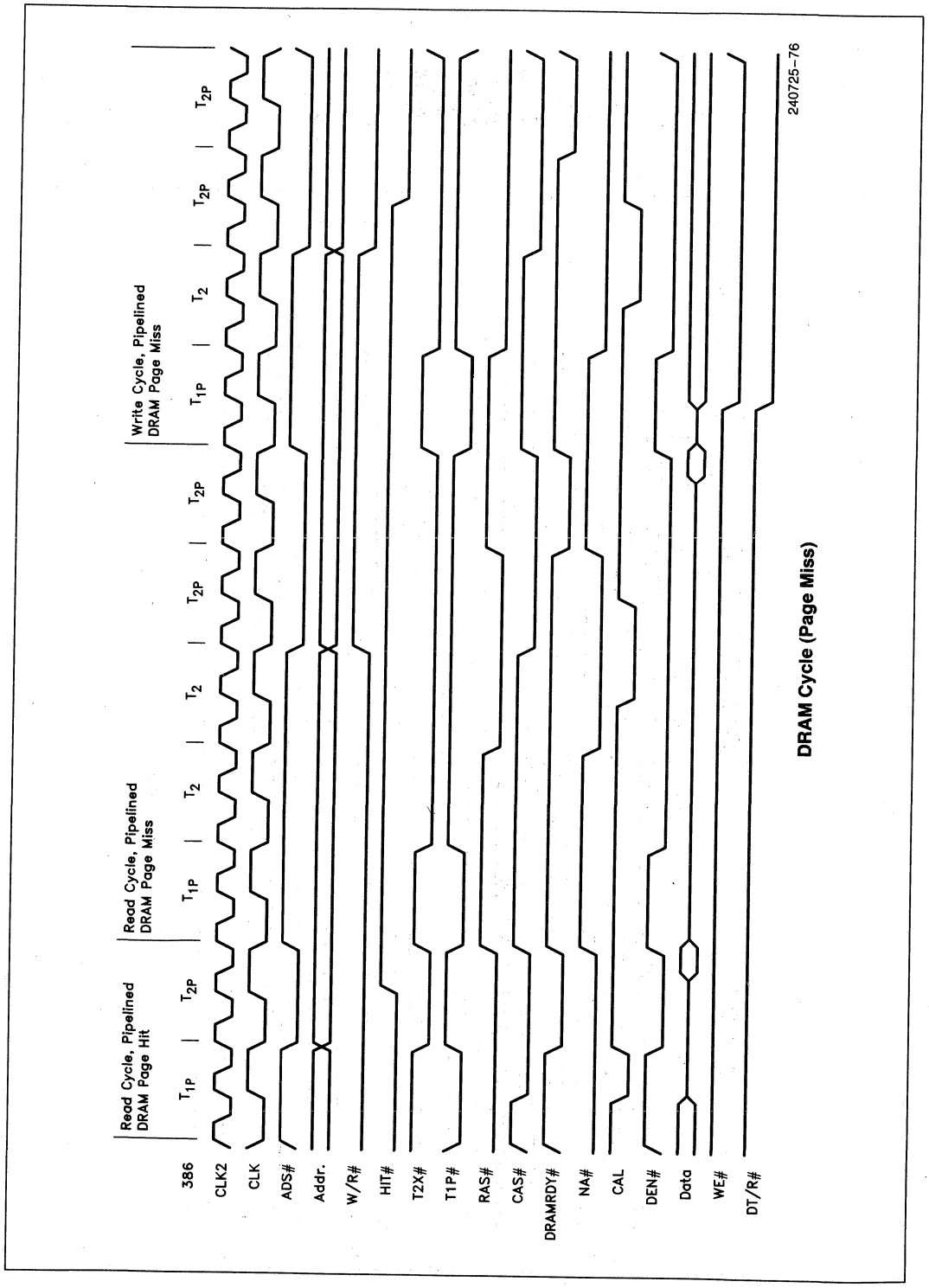
PAL Codes: Ready (Continued)

APPENDIX C
TIMING EQUATIONS

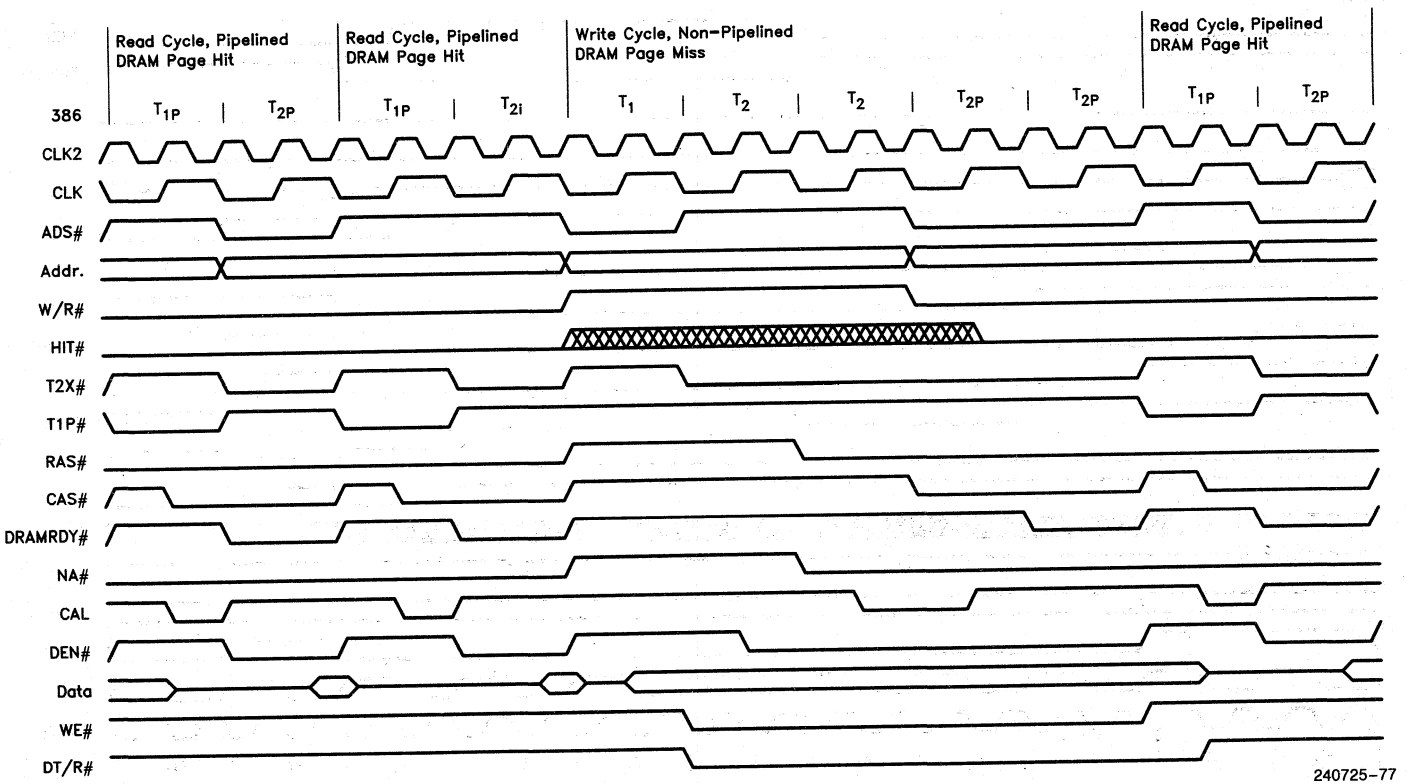


DRAM Cycle (R/W Hit/Miss)

5

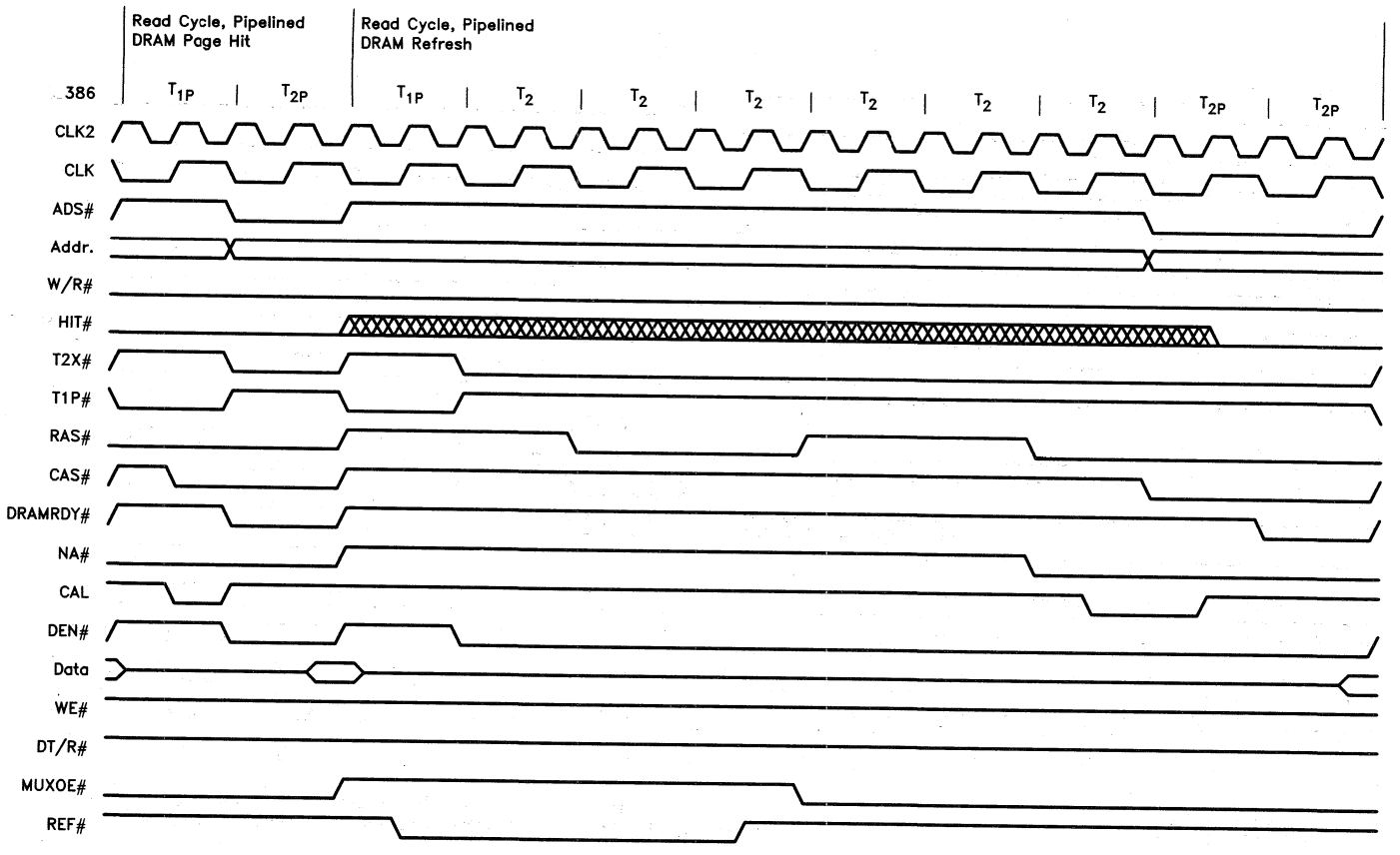


DRAM Cycle (Page Miss)

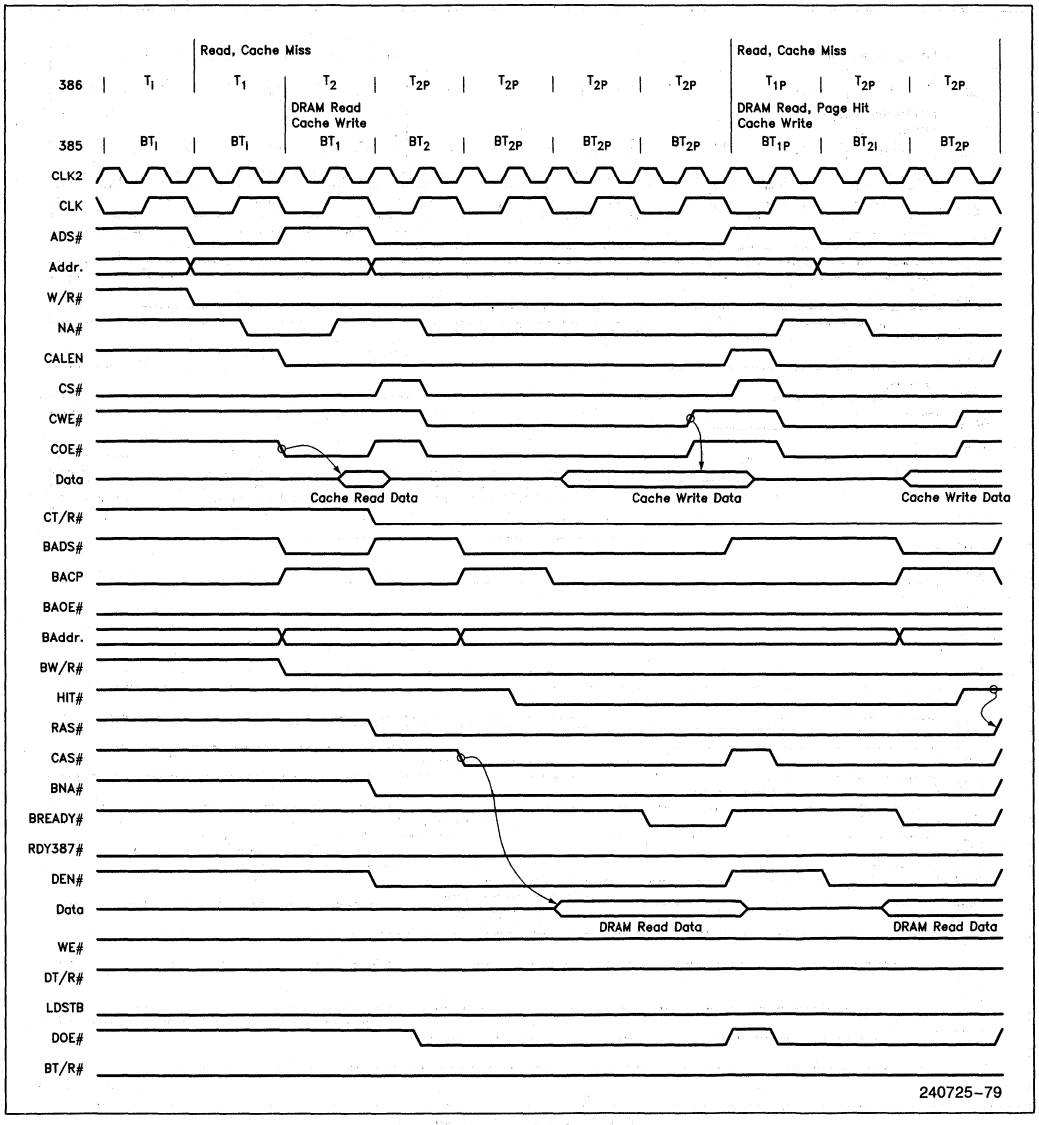

Read Cycle, Pipelined DRAM Page Hit
Read Cycle, Pipelined DRAM Page Hit
Write Cycle, Non-Pipelined DRAM Page Miss
Read Cycle, Pipelined DRAM Page Hit
386
T1P
T2P
T1P
T2i
T1
T2
T2
T2P
T2P
T1P
T2P
CLK2
CLK
ADS#
Addr.
W/R#
HIT#
T2X#
T1P#
RAS#
CAS#
DRAMRDY#
NA#
CAL
DEN#
Data
WE#
DT/R#
240725-77

DRAM Cycle

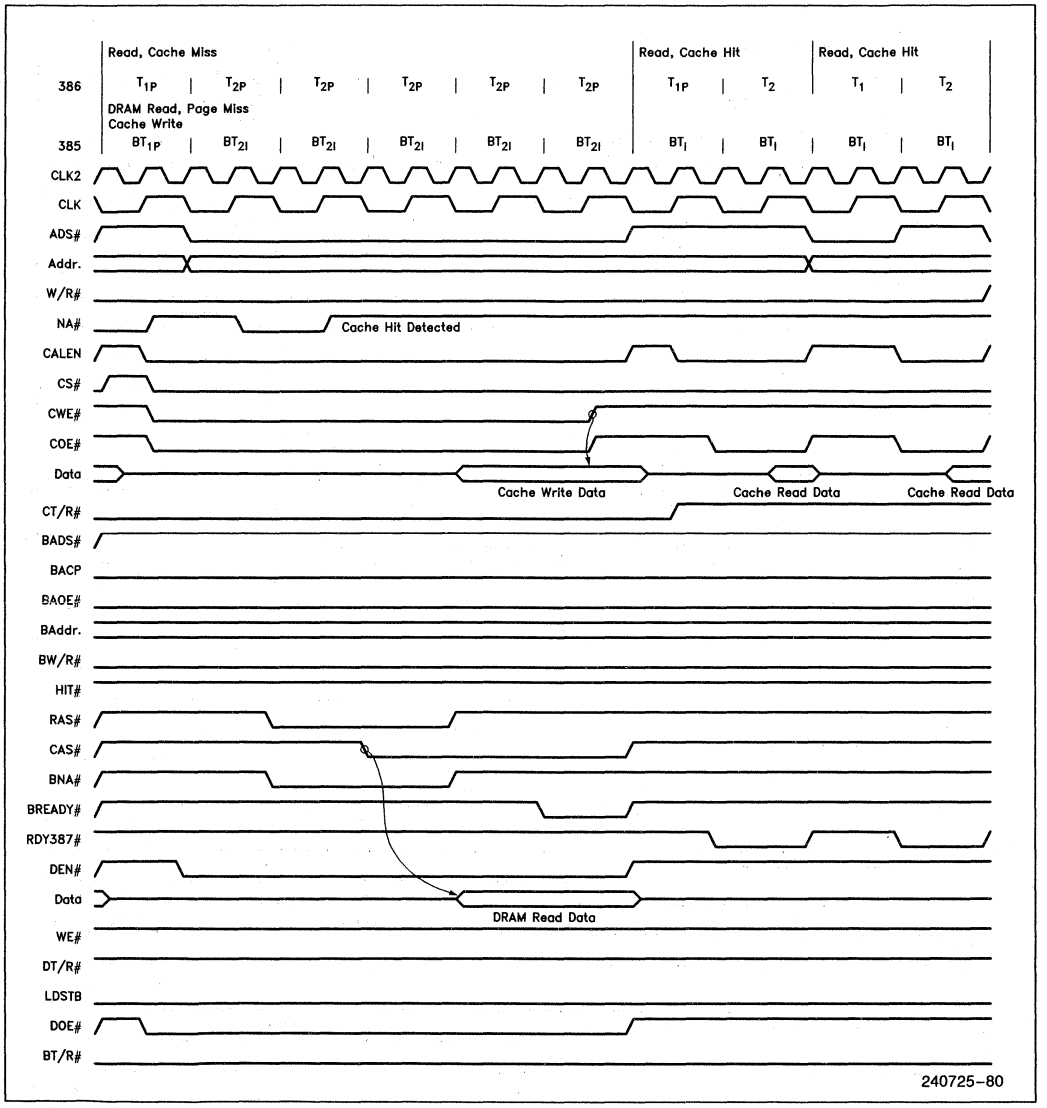
5



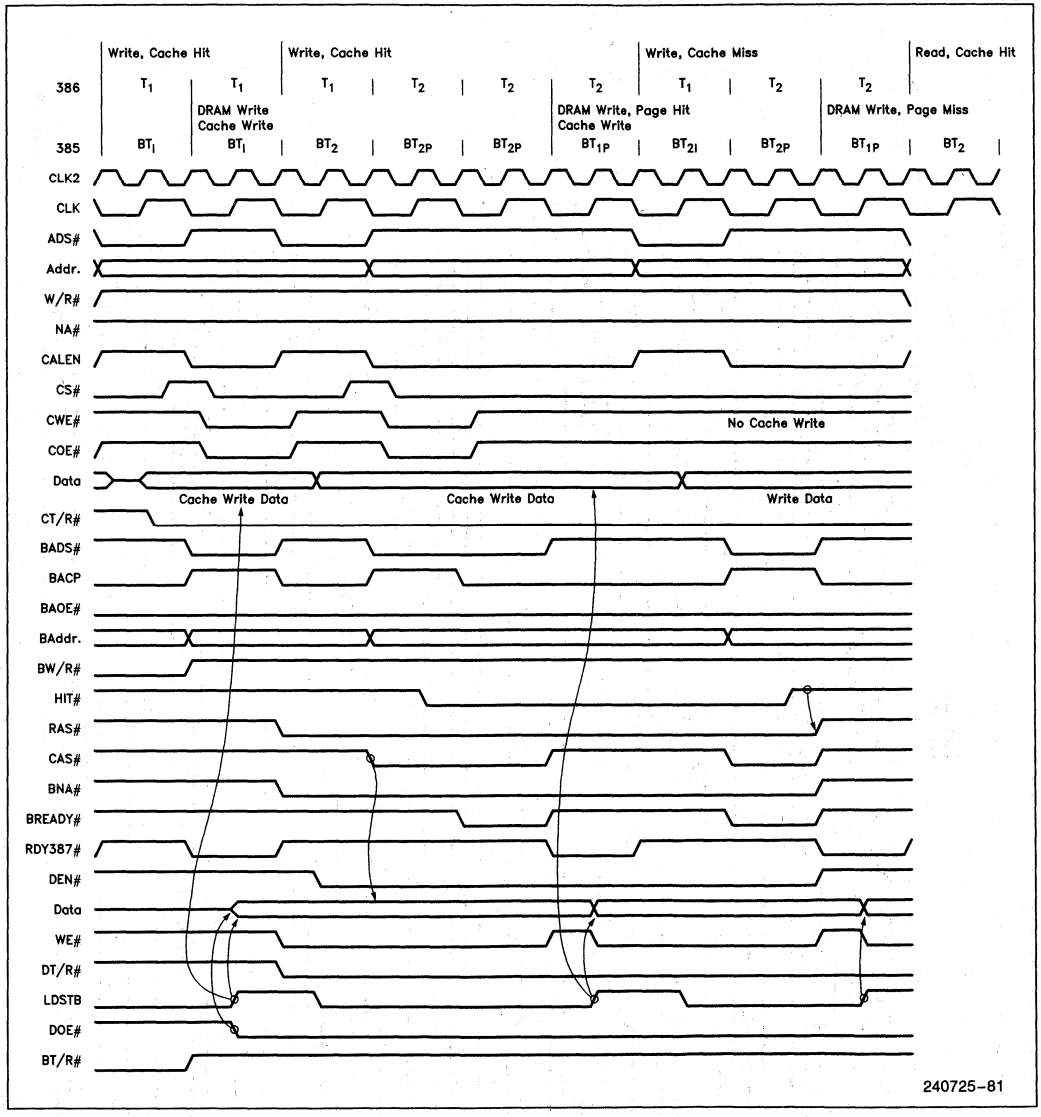
DRAM Refresh Cycle


Read, Cache Miss
Read, Cache Miss
386
385
DRAM Read
Cache Write
DRAM Read, Page Hit
Cache Write
CLK2
CLK
ADS#
Addr.
W/R#
NA#
CALEN
CS#
CWE#
COE#
Data
Cache Read Data
Cache Write Data
Cache Write Data
CT/R#
BADS#
BACP
BAOE#
BAddr.
BW/R#
HIT#
RAS#
CAS#
BNA#
BREADY#
RDY387#
DEN#
Data
DRAM Read Data
DRAM Read Data
WE#
DT/R#
LDSTB
DOE#
BT/R#
240725-79

Cache Cycle

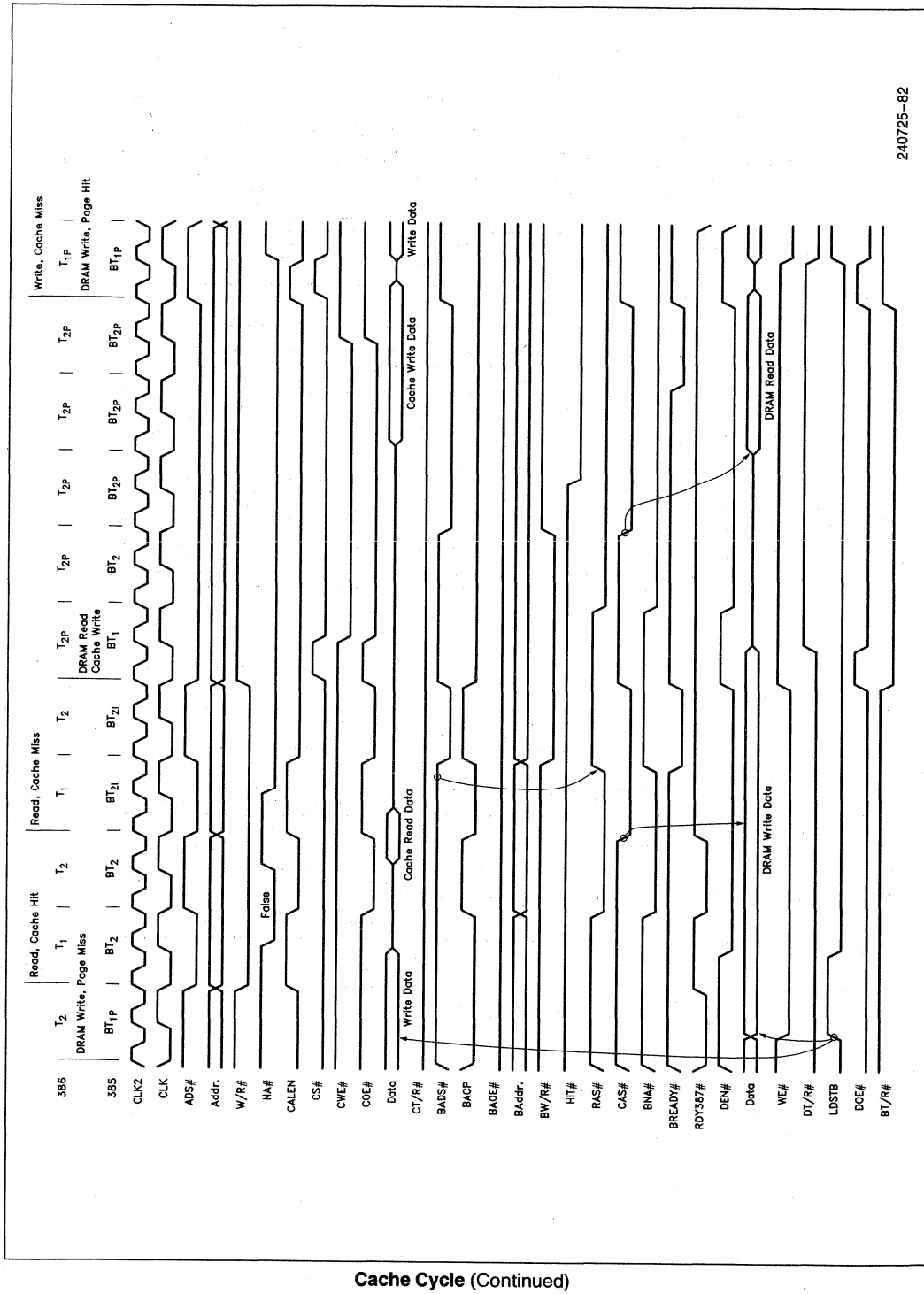


Cache Cycle (Continued)

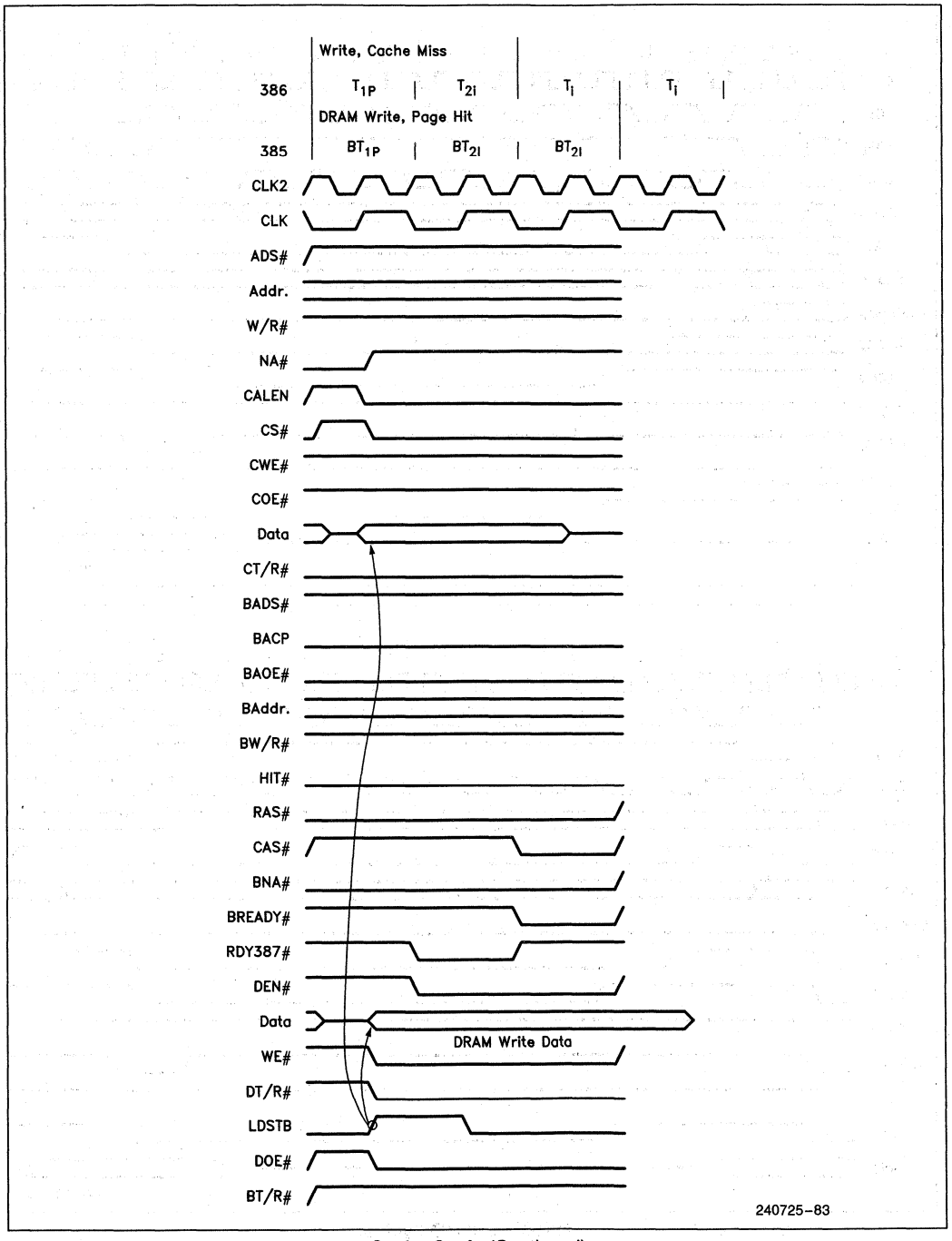


Cache Cycle (Continued)

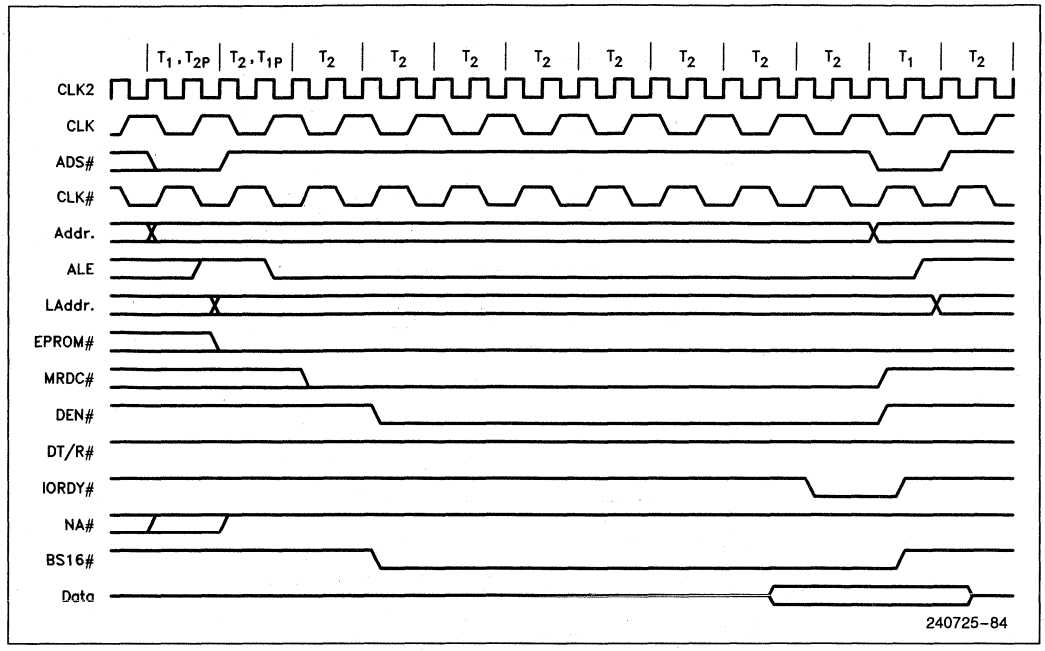
5

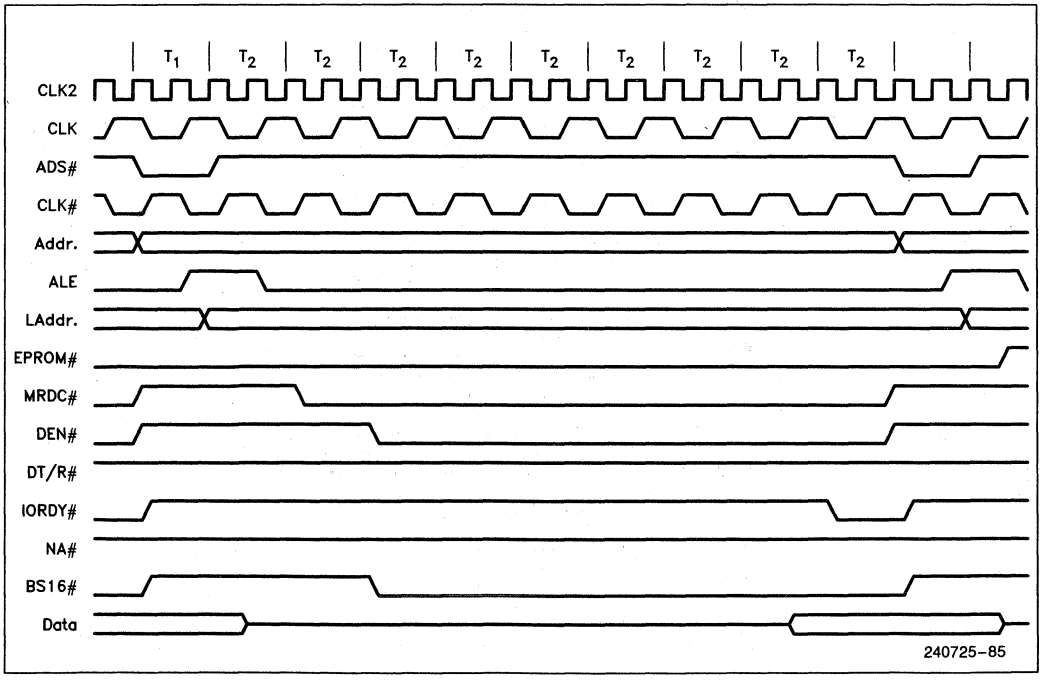


Cache Cycle (Continued)

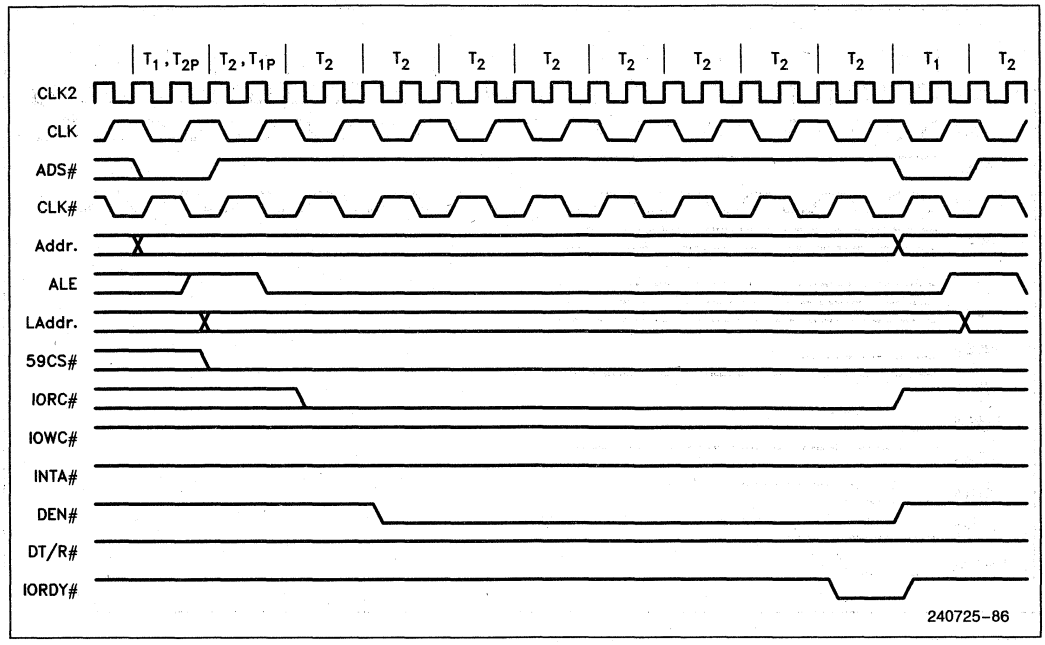


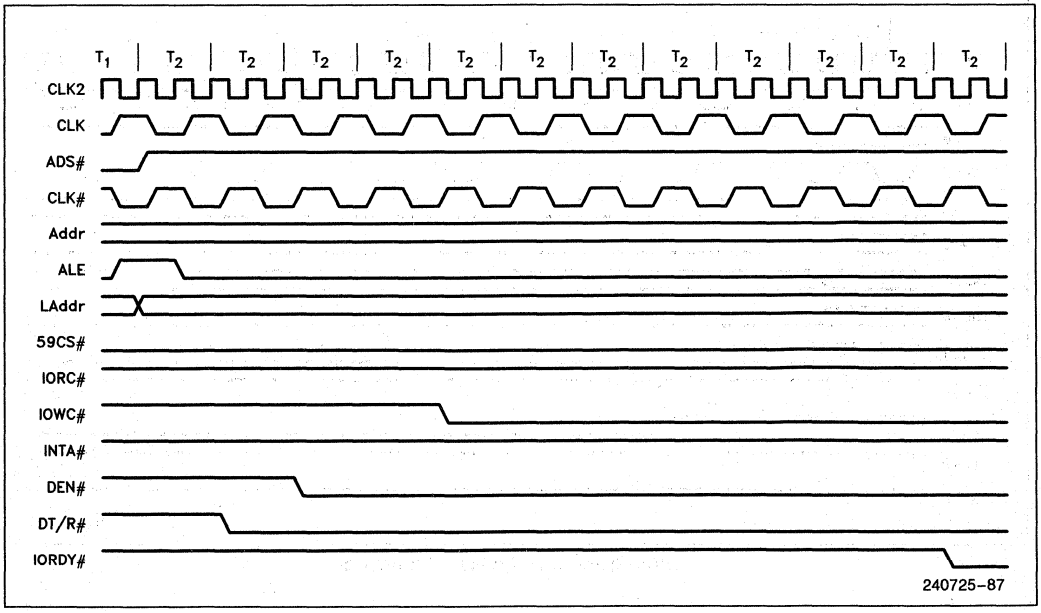
Cache Cycle (Continued)





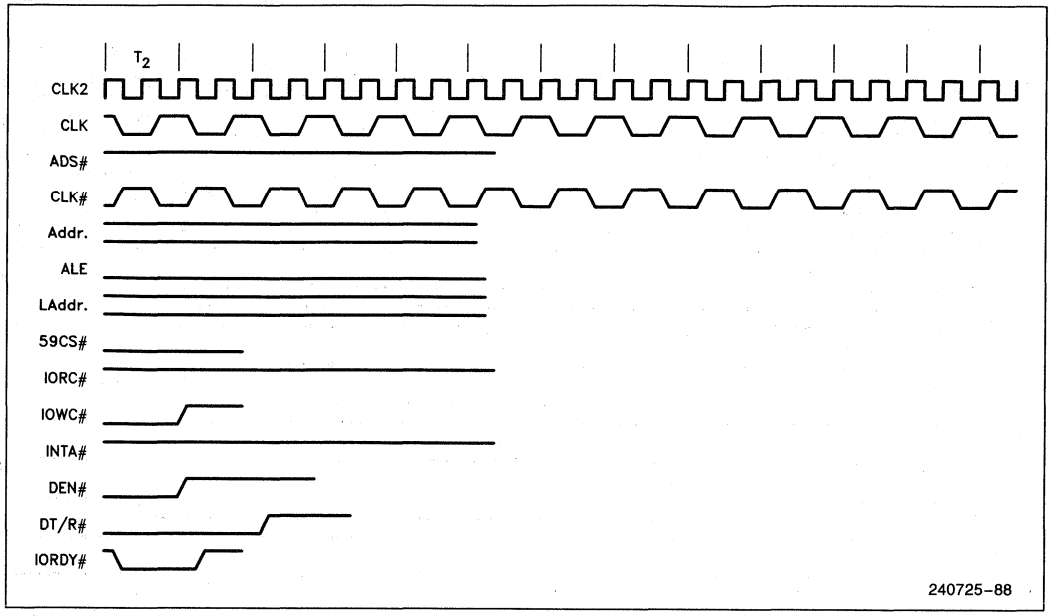
EPROM and I/O Cycles

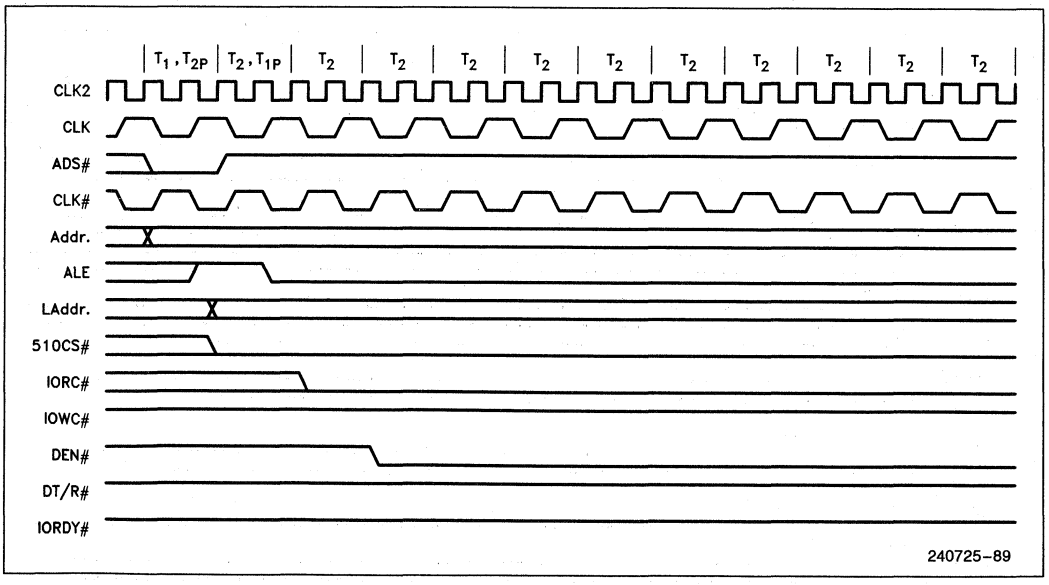




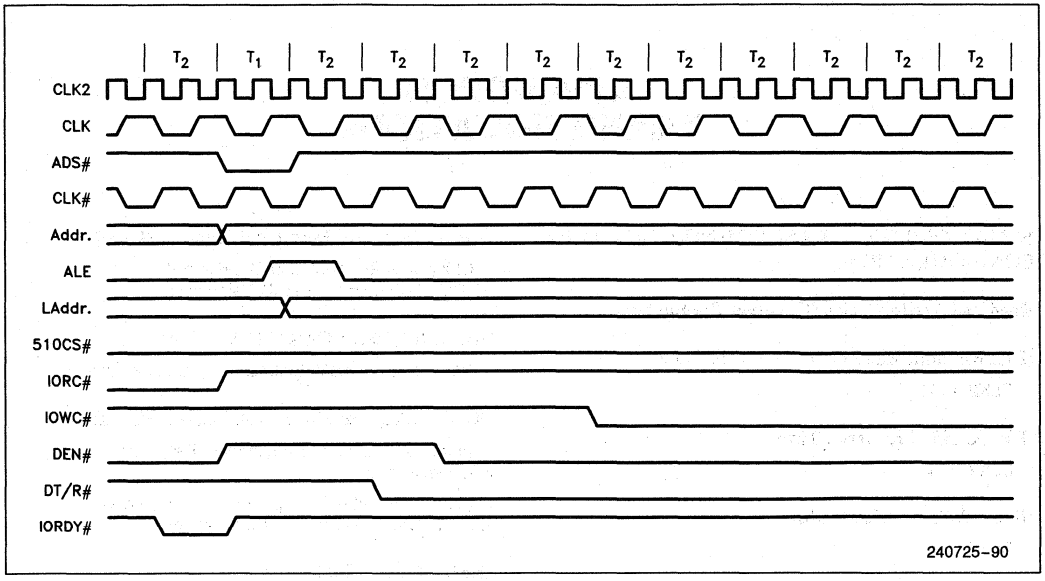
EPROM and I/O Cycles (Continued)

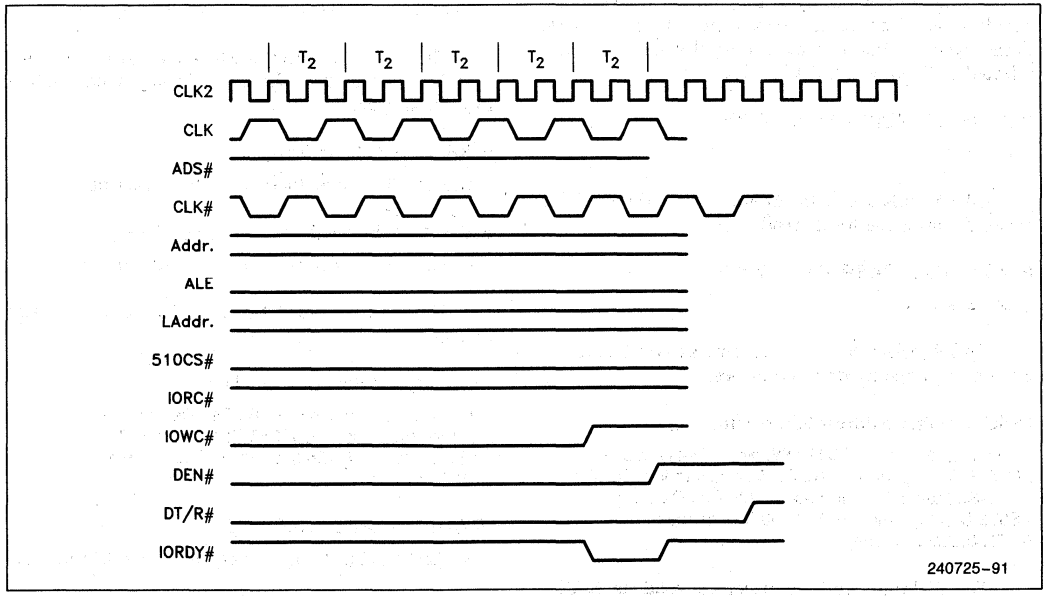
5





EPROM and I/O Cycles (Continued)





EPROM and I/O Cycles (Continued)

5

APPENDIX D
TIMING EQUATIONS

EQUATIONS FOR DRAM TIMINGS (NO CACHE CONFIGURATION):

Read and Write Cycles (Common Parameters):

tRC: Random Read or Write Cycle Time

CLK2 × 10

tRP: RAS# Precharge Time

CLK2 × 4

tRAS: RAS# Pulse Width

CLK2 × 4

A random DRAM cycle may have a RAS# pulse which is only four CLK2 periods wide. This is the case if the cycle is followed by Idle cycles (DRAMs not selected or Ti's) or a DRAM page miss.

tCAS (Read): CAS# Pulse Width

CLK2 × 3

CAS# pulses can be as narrow as three CLK2 cycles during Page Mode read cycles.

tCAS (Write): CAS# Pulse Width

CLK2 × 2

CAS# pulses can be as narrow as two CLK2 cycles during Page Mode write cycles.

tASC: Column Address Setup Time

min (CLK2 × 2 + AS32.tphl.min − Delay.max − ACT258.StoZ.tpl.max − ACT258.Cap.Derating, CLK2 × 3 + AS32.tphl.min − t6.max − 386.Cap.Derating − AS373.DtoO.tpd.max − ACT258.ItoZ.tpl.max − ACT258.Cap.Derating)

The Column Address becomes valid as RAS# switches from High to Low or as the 386 address becomes valid while RAS# is already Low (i.e., Page Mode, Pipelined cycles)

tCAH: Column Address Hold Time

CLK2 + AS373.GtoO.tpd.min + ACT258.ItoZ.tpl.min − AS32.tphl.max

The CAL (Column Address Latch) signal is activated one CLK2 period after the active-going edge of CAS#.

tAR: Column Address Hold Time to RAS#

CLK2 × 3 + AS373.GtoO.tpd.min + ACT258.ItoZ.tpl.min − RAS.Delay.max

tRCD: RAS# to CAS# Delay Time

CLK2 × 2 + AS32.tphl.min − RAS.Delay.max

tRAD: RAS# to Column Address Delay Time

(min) ACT258.StoZ.tphl.min + Delay.min − RAS.Delay.max

(max) ACT258.StoZ.tphl.max + Delay.max + ACT258.Cap.Derating − RAS.Delay.min

tRSH: RAS# Hold Time

CLK2 × 2 − AS32.tphl.max + RAS.Delay.min

The worst case occurs when a DRAM Page miss or Idle is detected at the end of the current DRAM Page miss cycle.

tCSH: CAS# Hold Time

CLK2 × 6 + AS32.tplh.min − RAS.Delay.max

tCRP: CAS# to RAS# Precharge Time

CLK2 × 2 + RAS.Delay.min − AS32.tplh.max

This is guaranteed by the DRAM control state machine.

tASR: Row Address Setup Time

CLK2 × 2 − t6.max − 386.Cap.Derating − ACT258.ItoZ.max − ACT258.Cap.Derating + H124.tpd.min + H125.tpd.min + PAL.tco.min + RAS.Delay.min

tRAH: Row Address Hold Time

ACT258.StoZ.tphl.min + Delay.min − RAS.Delay.max

tT: Transition Time (Rise and Fall)

tREF: Refresh Period

tREF2: Refresh Period

Read Cycles:

tRAC: Access Time

CLK2 × 6 − H124.tpd.max − H125.tpd.max − PAL.tco.max − t21.min − F245.max − RAS.Delay.max

tCAC: Access Time from CAS#

CLK2 × 3 − H124.tpd.max − H125.tpd.max − PAL.tco.max − AS32.tphl.max − t21.min − F245.max

tAA: Access Time from Address

CLK2 × 6 − t6.max − 386.Cap.Derating − AS373.DtoO.max − ACT258.ItoZ.tp.max − ACT258.Cap.Derating − t21.min − F245.max

tRCS: Read Command Setup Time

CLK2 + AS32.tphl.min

tRCH: Read Command Hold Time to CAS#

CLK2 − AS32.tplh.max

tRRH: Read Command Hold Time to RAS#

CLK2 − RAS.Delay.max

tOFF: Output Buffer Turn-off Time

CLK2 × 2 + F245.tzh.min

Write Cycles:

tWCS: Write Command Setup Time

CLK2 × 3 + AS32.tphl.min

tWCH: Write Command Hold Time

CLK2 × 2 − AS32.tplh.max

tWCR: Write Command Hold Time to RAS#

CLK2 × 6 − RAS.Delay.max

tWP: Write Command Pulse Width

CLK2 × 5

tRWL: Write Command to RAS# Lead Time

CLK2 × 5 + RAS.Delay.min

tCWL: Write Command to CAS# Lead Time

CLK2 × 5

tDS: Data-in Setup Time

CLK2 × 3 + H124.tp.min + H125.tp.min + AS32.tphl.min − T12.max − F245.tp.max

tDH: Data-in Hold Time

CLK2 × 2 + F245.tpz.min − AS32.tphl.max

tDHR: Data-in Hold Time to RAS#

CLK2 × 6 + F245.tpz.max + RAS.Delay.min

Page Mode Cycles:

tPC: Page Mode Cycle Time

CLK2 × 4

tRAPC: Page Mode RAS# Pulse Width

CLK2 × 4

tRSW: RAS# to Second WE# Delay Time

CLK2 × 7 − RAS.Delay.max

tCP: CAS# Precharge Time

CLK2

tWI: Write Invalid Time

CLK2

tCAP: Access Time from Column Precharge Time

CLK2 × 4 − H124.tp.max − H125.tp.max − PAL.tco.max − t21.min − F245.max

5

80386 A.C. SPECIFICATIONS

Symbol	Parameter	80386-33 Minimum	Maximum
	Operating Frequency	8.00	33.33
t1	CLK2 Period	15.00	62.50
t2a	CLK2 High Time	6.25	
t2b	CLK2 High Time	4.50	
t3a	CLK2 Low Time	6.25	
t3b	CLK2 Low Time	4.50	
t4	CLK2 Fall Time		4.00
t5	CLK2 Rise Time		4.00
t6	A2-A31 Valid Delay	4.00	15.00
t7	A2-A31 Float Delay	4.00	20.00
t8	BE0#-BE3#, LOCK# Valid Delay	4.00	15.00
t9	BE0#-BE3#, LOCK# Float Delay	4.00	20.00
t10	W/R#, M/IO#, D/C#, ADS# Valid Delay	4.00	15.00
t11	W/R#, M/IO#, D/C#, ADS# Float Delay	4.00	25.00
t12	D0-D31 Write Data Valid Delay	5.00	24.00
t13	D0-D31 Float Delay	4.00	17.00
t14	HLDA Valid Delay	4.00	20.00
t15	NA# Setup Time	5.00	
t16	NA# Hold Time	3.00	
t17	BS16# Setup Time	5.00	
t18	BS16# Hold Time	3.00	
t19	Ready# Setup Time	7.00	
t20	Ready# Hold Time	4.00	
t21	D0-D31 Read Setup Time	5.00	
t22	D0-D31 Read Hold Time	3.00	
t23	HOLD Setup Time	11.00	
t24	HOLD Hold Time	3.00	
t25	RESET Setup Time	8.00	
t26	RESET Hold Time	3.00	
t27	NMI, INTR Setup Time	5.00	
t28	NMI, INTR Hold Time	5.00	
t29	PEREQ, ERROR#, BUSY# Setup Time	5.00	
t30	PEREQ, ERROR#, BUSY# Hold Time	4.00	

PAL SPECIFICATIONS

Symbol	Parameter	Minimum	Maximum
ts	Input or Feedback Setup Time	7.00	
tco	Clock to Output	3.00	6.50

ROW ADDRESS LATCH SPECIFICATIONS
74FCT843B (IDT)

Symbol	Parameter	50 pF Minimum	Maximum
tplh	Dn to On Propagation Delay	3.00	6.50
tphl		3.00	6.50
tplh	G to On Propagation Delay	6.00	8.00
tphl		4.00	8.00
ts	Setup Time	2.00	
th	Hold Time	3.00	

240725-C7

Timings for No Cache Configuration

ROW ADDRESS COMPARATOR SPECIFICATIONS
74PCT521B (Performance)

Symbol	Parameter	Minimum	Maximum
tplh	An or Bn to Q Propagation Delay	1.50	5.50
tphl		1.50	5.50
tplh	I to Q Propagation Delay	1.50	4.60
tphl		1.50	4.60

DRAM ADDRESS MULTIPLEXER SPECIFICATIONS
74ACT258

Symbol	Parameter	Minimum	Maximum
tplh	S to Zn Propagation Delay	1.00	11.50
tphl		1.00	11.00
tplh	E# to Zn Propagation Delay	1.00	9.50
tphl		1.00	9.50
tplh	In to Zn Propagation Delay	1.00	9.50
tphl		1.00	8.00

DATA TRANSCEIVER SPECIFICATIONS
74F245

Symbol	Parameter	Minimum	Maximum
tplh	An to Bn or Bn to An Propagation Delay	2.50	7.00
tphl		2.50	7.00
tzh	Output Enable Time	3.00	8.00
tzl		3.50	9.00
tphz	Output Disable Time	3.00	7.50
tplz		2.00	7.50

COLUMN ADDRESS LATCH SPECIFICATIONS
74AS573

Symbol	Parameter	Minimum	Maximum
tplh	Dn to On Propagation Delay	3.00	6.00
tphl		3.00	6.00
tplh	G to On Propagation Delay	6.00	11.50
tphl		4.00	7.50
ts	Setup Time	2.00	
th	Hold Time	3.00	

RAS# DELAY

Symbol	Parameter	Minimum	Maximum
tp	Propagation Delay	0.00	0.00

240725-C8

Timings for No Cache Configuration (Continued)

OR SPECIFICATIONS
74AS32

Symbol	Parameter	Minimum	Maximum
tplh	Propagation Delay	1.00	5.80
tphl		1.00	5.80

DRAM TIMING REQUIREMENTS

Symbol	Parameter	For 80386-33 Minimum	For 80386-33 Maximum	Timing Margin (NMB 2801-06) Minimum	Timing Margin (NMB 2801-06) Maximum
Read and Write Cycles (Common Parameters):					
tRC	Random Read or Write Cycle Time	150.00		29.00	
tRP	RAS# Precharge Time	60.00		5.00	
tRAS	RAS# Pulse Width	60.00		0.00	
tCAS	CAS# Pulse Width (Read)	45.00		34.00	
tCAS	CAS# Pulse Width (Write)	30.00		25.00	
tASC	Column Address Setup Time	9.70		9.70	
tCAH	Column Address Hold Time	14.20		8.20	
tAR	Column Address Hold Time to RAS#	50.00		10.00	
tRCD	RAS# to CAS# Delay Time	31.00		25.00	14.00
tRAD	RAS# to Column Address Delay Time	5.00	21.30	1.00	6.70
tRSH	RAS# Hold Time	24.20		9.20	
tCSH	CAS# Hold Time	91.00		51.00	
tCRP	CAS# to RAS# Precharge Time	24.20		21.20	
tASR	Row Address Setup Time	5.45		3.45	
tRAH	Row Address Hold Time	5.00		3.00	
tT	Transition Time (Rise and Fall)				
tREF	Refresh Period				
tREF2	Refresh Period				
Read Cycles:					
tRAC	Access Time	68.25		8.25	
tCAC	Access Time from CAS#	17.45		6.45	
tAA	Access Time from Address	41.20		9.20	
tRCS	Read Command Setup Time	16.00		16.00	
tRCH	Read Command Hold Time to CAS#	9.20		9.20	
tRRH	Read Command Hold Time to RAS#	15.00		15.00	
tOFF	Output Buffer Turn-off Time		33.00		16.00
Write Cycles:					
tWCS	Write Command Setup Time	46.00		46.00	
tWCH	Write Command Hold Time	24.20		19.20	
tWCR	Write Command Hold Time to RAS#	90.00		50.00	
tWP	Write Command Pulse Width	75.00		70.00	
tRWL	Write Command to RAS# Lead Time	75.00		62.00	
tCWL	Write Command to CAS# Lead Time	75.00		70.00	
tDS	Data-in Setup Time	17.75		17.75	
tDH	Data-in Hold Time	26.20		21.20	
tDHR	Data-in Hold Time to RAS#	97.50		57.50	
Page Mode Cycles:					
tPC	Page Mode Cycle Time	60.00		23.00	
tRAPC	Page Mode RAS# Pulse Width	60.00			
tRSW	RAS# to Second WE# Delay Time	105.00			
tCP	CAS# Precharge Time	15.00		10.00	
tWI	Write Invalid Time	15.00			
tCAP	Access Time from Column Precharge Time		38.25		4.25

240725-C9

Timings for No Cache Configuration (Continued)

ADDRESS DECODER REQUIREMENTS

Symbol	Parameter	For 80386-33 Minimum	Maximum
tpd	Available Propagation Delay		8.75

ROW ADDRESS COMPARATOR REQUIREMENTS

Symbol	Parameter	For 80386-33 Minimum	Maximum
tpd	Available Propagation Delay		8.75

NA# SETUP TIME

Symbol	Parameter	Minimum	Maximum
tNA#	Available NA# Setup Time	5.25	

QUAD TTL TO 10KH-ECL TRANSLATOR
MC10H124

Symbol	Parameter	Minimum	Maximum
tpd	Propagation Delay	2.75	3.25

QUAD 10KH-ECL to TTL TRANSLATOR
MC10H125

Symbol	Parameter	Minimum	Maximum
tpd	Propagation Delay	0.00	0.00

DELAY ELEMENT

Symbol	Parameter	Minimum	Maximum
tpd	Propagation Delay	4.00	6.00

240725-D0

Timings for No Cache Configuration (Continued)

5

DRAM SPECIFICATIONS

Symbol	NMB 2801-06 Minimum	NMB 2801-06 Maximum	VITELIC V53C256 (70 ns) Minimum	VITELIC V53C256 (70 ns) Maximum
tRC	121.00		130.00	
tRP	55.00		50.00	
tRAS	60.00	100000	70.00	75000.00
tCAS	11.00		15, 20	75000.00
tCAS	5.00			
tASC	0.00		0.00	
tCAH	6.00		15.00	
tAR	40.00		55.00	
tRCD	6.00	45.00	25.00	55.00
tRAD	4.00	28.00	20.00	35.00
tRSH	15.00		15, 25	
tCSH	40.00		70.00	
tCRP	3.00		15.00	
tASR	2.00		0.00	
tRAH	2.00		15.00	
tT			3.00	25.00
tREF				
tREF2				
tRAC		60.00		70.00
tCAC		11.00		15.00
tAA		32.00		35.00
tRCS	0.00		0.00	
tRCH	0.00		5.00	
tRRH	0.00		5.00	
tOFF		17.00	0.00	15.00
tWCS	0.00		0.00	
tWCH	5.00		15.00	
tWCR	40.00		55.00	
tWP	5.00		15.00	
tRWL	13.00		20.00	
tCWL	5.00		20.00	
tDS	0.00		0.00	
tDH	5.00		15.00	
tDHR	40.00		55.00	
tPC	37.00		50.00	
tRAPC				
tRSW				
tCP	5.00		15.00	
tWI				
tCAP		34.00		45.00

240725-D1

CAPACITIVE LOAD TIMING DERATING FOR 74ACT258

Load Capacitance (pF)	Additional Propagation Delay (ns)	
60.00	0.26	(p = 0.02625q - 1.3125)
80.00	0.79	
100.00	0.89	(p = 0.022q - 1.3125)
120.00	1.33	
140.00	1.77	
160.00	2.21	
180.00	2.65	
200.00	3.09	
220.00	3.83	(p = 0.01666q + 0.1666)
240.00	4.17	
260.00	4.50	
280.00	4.83	
300.00	5.17	

DRAM ADDRESS BUS TIMING DERATING

Reason	Capacitive Load (pF)	Additional Propagation Delay (ns)
DRAM Address Inputs	160.00	
F258 Output Microstrip/Strip Lines	60.00	
TOTAL	220.00 ===>	3.80

240725-D2

Timings for No Cache Configuration (Continued)

EQUATIONS FOR DRAM TIMINGS (82385 Active):

Read and Write Cycles (Common Parameters):

tRC: Random Read or Write Cycle Time

CLK2 × 10

tRP: RAS# Precharge Time

CLK2 × 4

tRAS: RAS# Pulse Width

CLK2 × 4

A random DRAM cycle may have a RAS# pulse which is only four CLK2 periods wide. This is the case if the cycle is followed by Idle cycles (DRAMs not selected or Ti's) or a DRAM page miss.

tCAS (Read): CAS# Pulse Width

CLK2 × 5

CAS# pulses can be as narrow as five CLK2 cycles during Page Mode read cycles.

tCAS (Write): CAS# Pulse Width

CLK2 × 2

CAS# pulses can be as narrow as two CLK2 cycles during Page Mode write cycles.

tASC: Column Address Setup Time

min (CLK2 × 2 + AS32.tphl.min − Delay.max − ACT258.StoZ.tpl.max − ACT258.Cap.Derating, CLK2 × 3 + AS32.tphl.min − t6.max − 386.Cap.Derating − AS373.DtoO.tpd.max − ACT258.ltoZ.tpl.max − ACT258.Cap.Derating)

The Column Address becomes valid as RAS# switches from High to Low or as the 386 address becomes valid while RAS# is already Low (i.e., Page Mode, Pipelined cycles)

tCAH: Column Address Hold Time

CLK2 + AS373.GtoO.tpd.min + ACT258.ltoZ.tpl.min − AS32.tphl.max

The CAL (Column Address Latch) signal is activated one CLK2 period after the active-going edge of CAS#.

tAR: Column Address Hold Time to RAS#

CLK2 × 3 + AS373.GtoO.tpd.min + ACT258.ltoZ.tpl.min − RAS.Delay.max

tRCD: RAS# to CAS# Delay Time

CLK2 × 2 + AS32.tphl.min − RAS.Delay.max

tRAD: RAS# to Column Address Delay Time

(min) ACT258.StoZ.tphl.min + Delay.min − RAS.Delay.max

(max) ACT258.StoZ.tphl.max + Delay.max + ACT258.Cap.Derating − RAS.Delay.min

tRSH: RAS# Hold Time

CLK2 × 2 − AS32.tphl.max + RAS.Delay.min

The worst case occurs when a DRAM Page miss or Idle is detected at the end of the current DRAM Page miss cycle.

tCSH: CAS# Hold Time

CLK2 × 6 + AS32.tphl.min − RAS.Delay.max

tCRP: CAS# to RAS# Precharge Time

CLK2 × 2 + RAS.Delay.min − AS32.tplh.max

This is guaranteed by the DRAM control state machine.

tASR: Row Address Setup Time

CLK2 × 2 − t6.max − 386.Cap.Derating − ACT258.ltoZ.max − ACT258.Cap.Derating + H124.tpd.min + H125.tpd.min + PAL.tco.min + RAS.Delay.min

tRAH: Row Address Hold Time

ACT258.StoZ.tphl.min + Delay.min − RAS.Delay.max

tT: Transition Time (Rise and Fall)

tREF: Refresh Period

tREF2: Refresh Period

5

Read Cycles:

tRAC: Access Time

CLK2 × 8 − H124.tpd.max − H125.tpd.max − PAL.tco.max − − F245.max − AS646.tpd.max − F245.max − RAS.Delay.max − SRAM.tDW − CLK2 + 385.t22a.min

tCAC: Access Time from CAS#

CLK2 × 5 − H124.tpd.max − H125.tpd.max − PAL.tco.max − AS32.tphl.max − F245.max − AS646.tpd.max − F245.max − SRAM.tDW − CLK2 + 385.t22a.min

tAA: Access Time from Address

CLK2 × 8 − t6.max − 386.Cap.Derating − AS373.DtoO.max − ACT258.ItoZ.tp.max − ACT258.Cap.Derating − F245.max − AS646.tpd.max − F245.max − SRAM.tDW − CLK2 + 385.t22a.min

tRCS: Read Command Setup Time

CLK2 + AS32.tphl.min

tRCH: Read Command Hold Time to CAS#

CLK2 − AS32.tplh.max

tRRH: Read Command Hold Time to RAS#

CLK2 − RAS.Delay.max

tOFF: Output Buffer Turn-off Time

CLK2 × 2 + F245.tzh.min

Write Cycles:

tWCS: Write Command Setup Time

CLK2 × 3 + AS32.tphl.min

tWCH: Write Command Hold Time

CLK2 × 2 − AS32.tplh.max

tWCR: Write Command Hold Time to RAS#

CLK2 × 6 − RAS.Delay.max

tWP: Write Command Pulse Width

CLK2 × 5

tRWL: Write Command to RAS# Lead Time

CLK2 × 5 + RAS.Delay.min

tCWL: Write Command to CAS# Lead Time

CLK2 × 5

tDS: Data-in Setup Time

CLK2 × 3 + H124.tp.min + H125.tp.min + AS32.tphl.min − − 385.t43c.max − AS646.GotO.tp.max − F245.tp.max

tDH: Data-in Hold Time

CLK2 × 2 + F245.tpz.min − AS32.tphl.max

tDHR: Data-in Hold Time to RAS#

CLK2 × 6 + F245.tpz.max + RAS.Delay.min

Page Mode Cycles:

tPC: Page Mode Cycle Time

CLK2 × 6

tRAPC: Page Mode RAS# Pulse Width

CLK2 × 4

tRSW: RAS# to Second WE# Delay Time

CLK2 × 7 − RAS.Delay.max

tCP: CAS# Precharge Time

CLK2

tWI: Write Invalid Time

CLK2

tCAP: Access Time from Column Precharge Time

CLK2 × 6 − H124.tp.max − H125.tp.max − PAL.tco.max − − F245.max − AS646.tpd.max − F245.max − SRAM.tDW − CLK2 + 385.t22a.min

DRAM TIMING REQUIREMENTS

Symbol	Parameter	For 80386-33 Minimum	For 80386-33 Maximum	Timing Margin (NMB 2801-06) Minimum	Timing Margin (NMB 2801-06) Maximum
Read and Write Cycles (Common Parameters):					
tRC	Random Read or Write Cycle Time	150.00		29.00	
tRP	RAS# Precharge Time	60.00		5.00	
tRAS	RAS# Pulse Width	60.00		0.00	
tCAS	CAS# Pulse Width (Read)	75.00		64.00	
tCAS	CAS# Pulse Width (Write)	30.00		25.00	
tASC	Column Address Setup Time	9.70		9.70	
tCAH	Column Address Hold Time	14.20		8.20	
tAR	Column Address Hold Time to RAS#	50.00		10.00	
tRCD	RAS# to CAS# Delay Time	31.00		25.00	14.00
tRAD	RAS# to Column Address Delay Time	5.00	21.30	1.00	6.70
tRSH	RAS# Hold Time	24.20		9.20	
tCSH	CAS# Hold Time	91.00		51.00	
tCRP	CAS# to RAS# Precharge Time	24.20		21.20	
tASR	Row Address Setup Time	6.20		4.20	
tRAH	Row Address Hold Time	5.00		3.00	
tT	Transition Time (Rise and Fall)				
tREF	Refresh Period				
tREF2	Refresh Period				
Read Cycles:					
tRAC	Access Time	67.50		7.50	
tCAC	Access Time from CAS#	16.70		5.70	
tAA	Access Time from Address	37.70		5.70	
tRCS	Read Command Setup Time	20.80		20.80	
tRCH	Read Command Hold Time to CAS#	9.20		9.20	
tRRH	Read Command Hold Time to RAS#	15.00		15.00	
tOFF	Output Buffer Turn-off Time		33.00		16.00
Write Cycles:					
tWCS	Write Command Setup Time	46.00		46.00	
tWCH	Write Command Hold Time	24.20		19.20	
tWCR	Write Command Hold Time to RAS#	90.00		50.00	
tWP	Write Command Pulse Width	75.00		70.00	
tRWL	Write Command to RAS# Lead Time	75.00		62.00	
tCWL	Write Command to CAS# Lead Time	75.00		70.00	
tDS	Data-in Setup Time	9.00		9.00	
tDH	Data-in Hold Time	31.70		26.70	
tDHR	Data-in Hold Time to RAS#	97.50		57.50	
Page Mode Cycles:					
tPC	Page Mode Cycle Time	90.00		53.00	
tRAPC	Page Mode RAS# Pulse Width	60.00			
tRSW	RAS# to Second WE# Delay Time	105.00			
tCP	CAS# Precharge Time	15.00		10.00	
tWI	Write Invalid Time	15.00			
tCAP	Access Time from Column Precharge Time		37.50		3.50

240725-D3

Timings with Cache Active

5

APPENDIX E
REFERENCES

REFERENCES

Advanced CMOS Logic Designer's Handbook, Texas Instruments Inc., 1988.

Blood W., *MECL System Design Handbook*, Motorola Corp., 1983.

Keeler R., "High Speed Digital Printed Circuit Boards," *Electronic Packaging & Production*, pp. 140-145, Jan. 1986.

Tomlinson J., "Avoid The Pitfalls of High Speed Logic Design," *Electronic Design*, pp. 75-84, Nov. 9, 1989.

Pace C., "Terminate Bus Lines to Avoid Overshoot and Ringing," *EDN*, pp. 227-234, Sept. 17, 1987.

Royle D., "Rules Tell Whether Interconnections Act Like Transmission Lines," *EDN*, pp. 131-136, June 23, 1988.

Royle D., "Correct Signal Faults by Implementing Line-Analysis Theory," *EDN*, pp. 143-148, June 23, 1988.

Winchester E., "Guidelines Help You Design High-Speed PC Boards," *EDN*, pp. 221-226, Nov. 28, 1985.

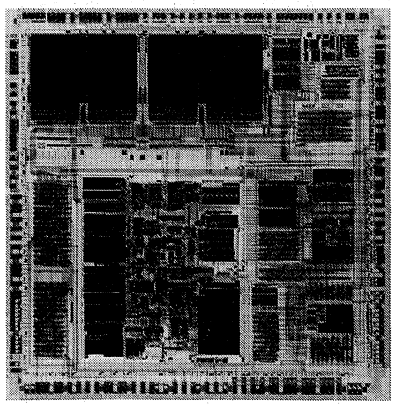
Yeargan J. R., Day R. L., and Nguyen T., "Effects of Printed Circuit Board Transmission Lines an Loading on Gate Performance," *IEEE Transactions on Industrial Electronics*, Vol. IE-34, no. 3, pp. 399-405, Aug. 1987.

386™ SL MICROPROCESSOR SuperSet
Highly-Integrated Static 386™ Microprocessor
Complete ISA Peripheral Subsystem
System-Wide Power Management

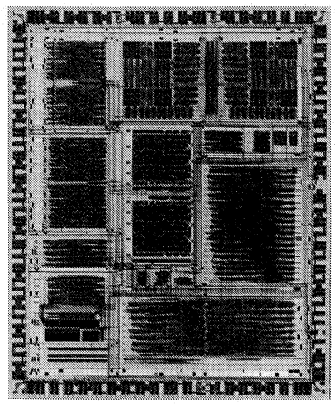
- **Static 386™ CPU Core**
  - **Runs MS-DOS*, WINDOWS*, OS/2* and UNIX***
  - **Object Code Compatible with Intel 8086, 80286 and 386™ Microprocessors**
- **Architecture Extension for Power Management Transparent to Operating Systems and Applications**
- **Complete ISA System, with Extended Support**
  - **Full ISA Bus Control, Status and Address and Data Interface Logic, with Full 24 mA Drive**
  - **Compatible ISA Bus Peripherals**
  - **System I/O Decoding, Programmable Chip Selects and Support Interfaces**
  - **High-Speed Peripheral Interface Bus (PI-Bus Support)**
  - **New ideaPort Interface for Hardware Expansion**
- **Integrated Cache Controller and Tag RAM**
  - **No-Glue Cache SRAM Interface**
  - **16k, 32k, or 64 kByte Cache Size**
  - **Direct, 2-Way or 4-Way Set Associative Organization**
- **Programmable Memory Control**
  - **No-Glue, Page-Mode DRAM Interface**
  - **SRAM Support for Lowest Power**
  - **512k to 32 MBytes**
  - **Full Hardware LIM EMS 4.0**

The 386™ SL Microprocessor SuperSet combines an ISA bus compatible personal computer's microprocessor, memory controller, cache controller and peripheral subsystems into just two Very Large Scale Integration (VLSI) devices. The product's high-integration and power conservation features reduce the size and power consumption typically associated with fully Industry Standard Architecture (ISA) bus compatible systems. In addition, new expandability and flexibility features offer the capability for continued innovation in battery-operated, space-constrained systems. The SL SuperSet brings 100% ISA-Bus compatibility to system designs ranging from the smallest palm-top and notebook PCs to expandable lap-top systems.

386 is a registered trademark of Intel Corporation.
*MS-DOS and WINDOWS are trademarks of Microsoft Corporation.
UNIX is a trademark of AT&T.
OS/2 is a trademark of International Business Machines Corporation.



240814-1



240814-2

Figure 1-1. Die Photograph of the 386™ SL Microprocessor (left) and 82360SL ISA Peripheral I/O (right)

October 1990
Order Number: 240814-001

386™ SL MICROPROCESSOR
386™ Microprocessor Core, with Integrated Bus Memory, and Cache Controllers; and System Power Management
Fully-Static CHMOS IV Technology

- Static 386™ CPU Core
  - Optimized and Compatible with Standard Operating System Software such as: MS-DOS*, WINDOWS*, OS/2* and UNIX*
  - Object Code Compatible with Intel 8086, 80286 and 386™ Microprocessors
  - Runs All Desk-Top Applications, 16- or 32-Bit
  - D.C. to 20 MHz Operation
  - 32 Megabytes Physical Memory/ 64 Terabytes Virtual Memory
  - 4 Gigabyte Maximum Segment Size
  - High Integration, Low Power Intel CHMOS IV Process Technology
- Transparent Power-Management System Architecture
  - System Management Mode Architecture Extension for Truly Compatible Systems
  - Power Management Transparent to Operating Systems and Application Programs
  - Programmable Hardware Supports Custom Power-Control Methods
- Direct Drive Bus Interfaces
  - Full ISA Bus Interface, with 24 mA Drive
  - High Speed Peripheral Interface Bus
- Integrated Cache Controller and Tag RAM
  - No-Glue Cache SRAM Interface
  - 16k, 32k, or 64 kByte Cache Size
  - Direct, 2-Way or 4-Way Set Associative Organization
  - Write Posting—Double Posted Writes in the Bus Controller
  - 16-Bit Line Size—Reduces Bus Utilization for Cache Line Fills
  - Write-Thru, with SmartHit Algorithm for Reduced Main Memory Power Consumption
- Programmable Memory Control
  - No-Glue, Page-Mode DRAM Interface
  - SRAM Support for Lowest Power
  - 1, 2, or 4 Banks Interleaved, with Programmable Wait States
  - 512k to 32 MBytes
  - Advanced, Flexible Address-Map Configuration
  - Full Hardware LIM EMS 4.0 Address Translation to 32 Megabytes without Waitstate Penalty

82360SL I/O Subsystem
Complete ISA Peripheral Subsystem
Integrated System Power Management
Fully-Static CHMOS IV Technology

- Complete ISA System, with Extended Support
  - — Full ISA Bus Control, Status and Address and Data Interface Logic, with Full 24 mA Drive
  - — Compatible ISA Bus Peripherals:
    - Two 8237 Direct Memory Access Controllers
    - Two 8254 Programmable Timer Counters (6 Timer/Counter Channels)
    - Two 8259A Programmable Interrupt Controllers (15 Channels)
    - Enhanced LS612 Page Memory Mapper
    - One 146818 Real Time Clock w/256-byte CMOS RAM
    - One 16450 Dual Serial Port Controller
    - One 8-Bit Parallel I/O Port (Centronics or Bi-Directional)
  - — Additional System I/O Decoding, Programmable Chip Selects and Support Interfaces:
    - Full Integrated Drive Electronics (I.D.E.) Hard Disk Interface
    - Floppy Disk Controller
- Keyboard Controller Chip Selects and Support Logic
  - — External Real Time Clock Support
  - — PS/2 and EISA Control/Status Ports
  - — Local Memory and ISA-Bus Memory Refresh Control
  - — New ideaPort Interface for Hardware Expansion
- Transparent Power-Management System Architecture
  - — Architecture Extension for Truly Compatible Systems
  - — Transparent to Operating Systems and Applications Programs
  - — Programmable Hardware Supports Custom Power-Control Methods
  - — Integrated Power Management Unit Manages Power-Events Safely

386™ SL Microprocessor SuperSet
386™ SL CPU and 82360SL I/O

1.0 INTRODUCTION 5-735

2.0 MECHANICAL PACKAGE SPECIFICATIONS, PIN ASSIGNMENT AND CHARACTERISTICS 5-742

3.0 SIGNAL DESCRIPTION 5-758

4.0 PACKAGE THERMAL SPECIFICATIONS 5-773

5.0 D.C. SPECIFICATIONS 5-774

6.0 TIMING SPECIFICATIONS 5-781

7.0 TIMING DIAGRAMS 5-801

8.0 CAPACITIVE DERATING INFORMATION 5-852

9.0 DAMPING RESISTOR REQUIREMENTS 5-857

10.0 MECHANICAL DETAILS OF LGA AND PQFP PACKAGES 5-858

11.0 REVISION HISTORY 5-863

1.0 INTRODUCTION

This document provides the pinouts, signal descriptions, and D.C./A.C. electrical characteristics of the 386™ SL CPU and 82360SL ISA I/O Peripheral device. Consult Intel for the most recent design-in information. For a thorough description of any functional topic, other than the parametric specifications, please consult the latest 386 SL Microprocessor SuperSet System Design Guide (Order No. 240816), and the 386 SL Microprocessor SuperSet Programmer's Guide (Order No. 240815).

Overview

The 386™ SL Microprocessor SuperSet is an extremely flexible pair of components marking a new milestone in microcomputer technology. Included in the pair are a 386 Architecture Central Processing Unit (CPU), several memory subsystem controllers, address translation and remapping logic, an optional cache memory controller, and an extensive collection of ISA bus compatible peripheral functions.

The SL SuperSet allows the personal computer designer to take advantage of the highest level of system integration, while preserving complete freedom in selecting system features, power/performance trade-offs, and value-added enhancements.

Essentially, all of the components needed to build an ISA bus compatible personal computer have been combined within just two components: the 386 SL microprocessor and memory control system, and the 82360SL ISA peripheral I/O and power management subsystem. The only other components needed for a complete personal computer are the main DRAM or optional static memory subsystem, optional cache SRAM and a graphics controller. A minimal amount of commodity Small Scale Integration (SSI) logic or Medium Scale Integration (MSI) logic buffers may be required for design-specific interface to peripheral devices on the ISA bus.

Systems based on the SL SuperSet typically include the functional blocks shown in Figure 1-2.

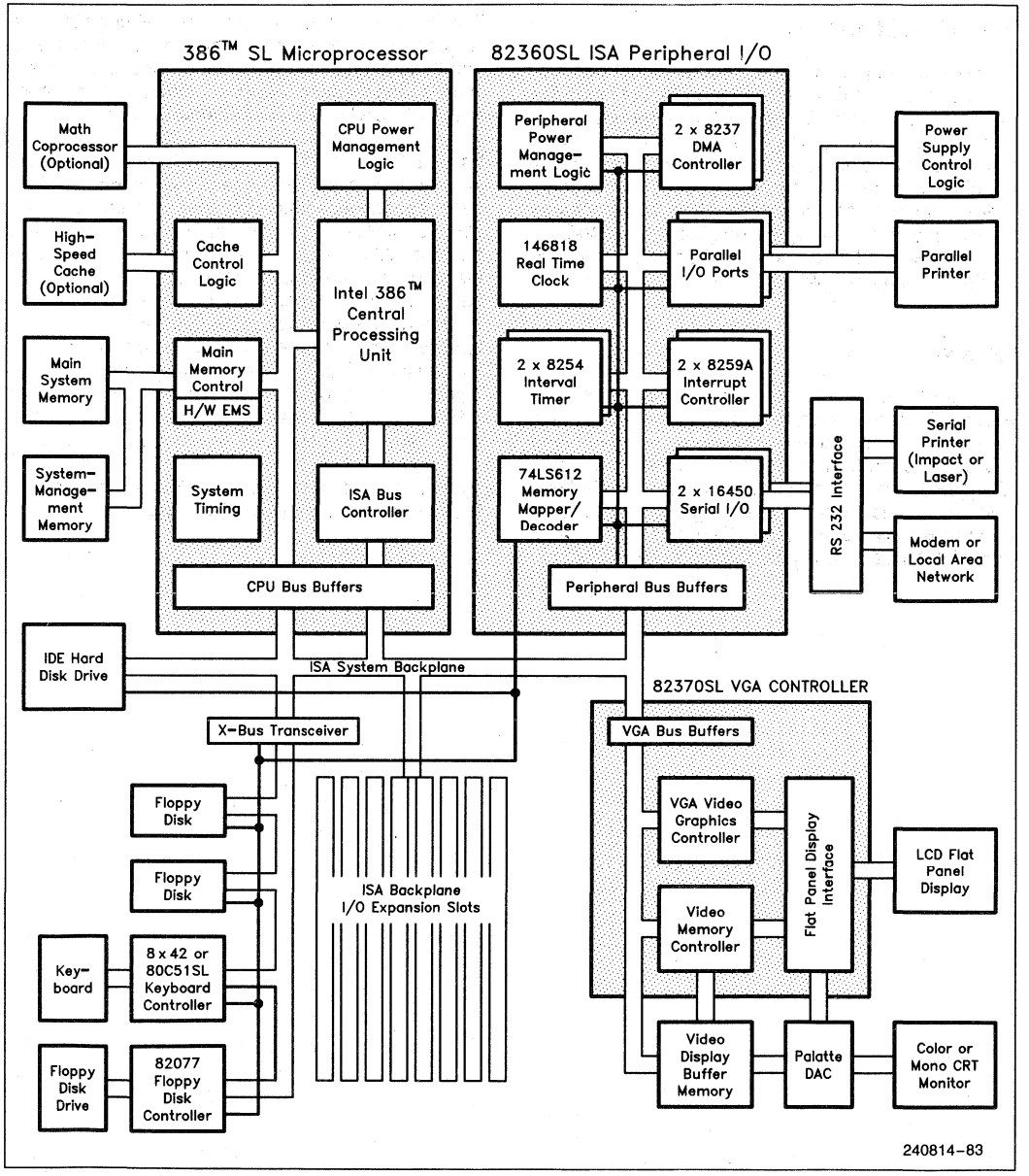


Figure 1-2. 386™ SL Microprocessor-Based System Functional Block Diagram

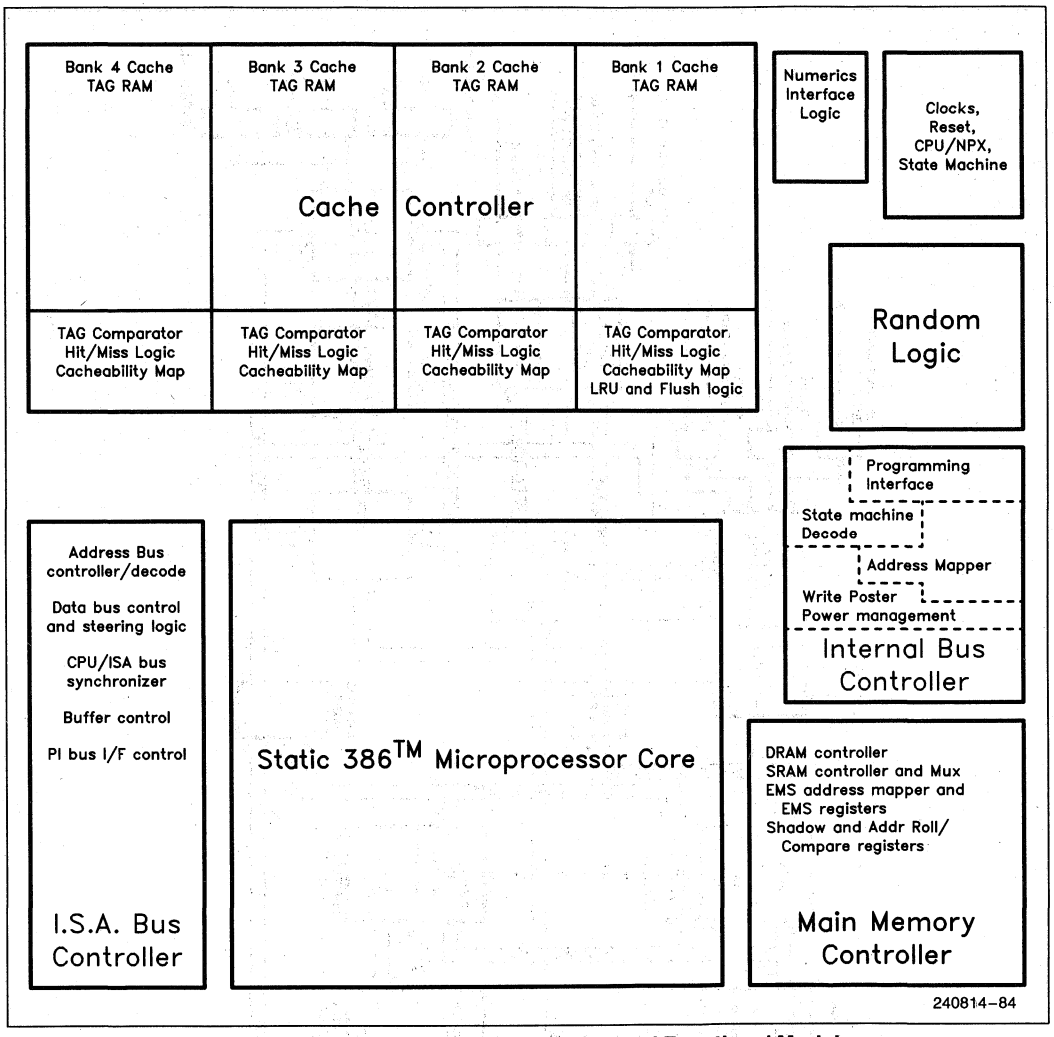


Figure 1-3a. 386™ SL Microprocessor Internal Functional Modules

5

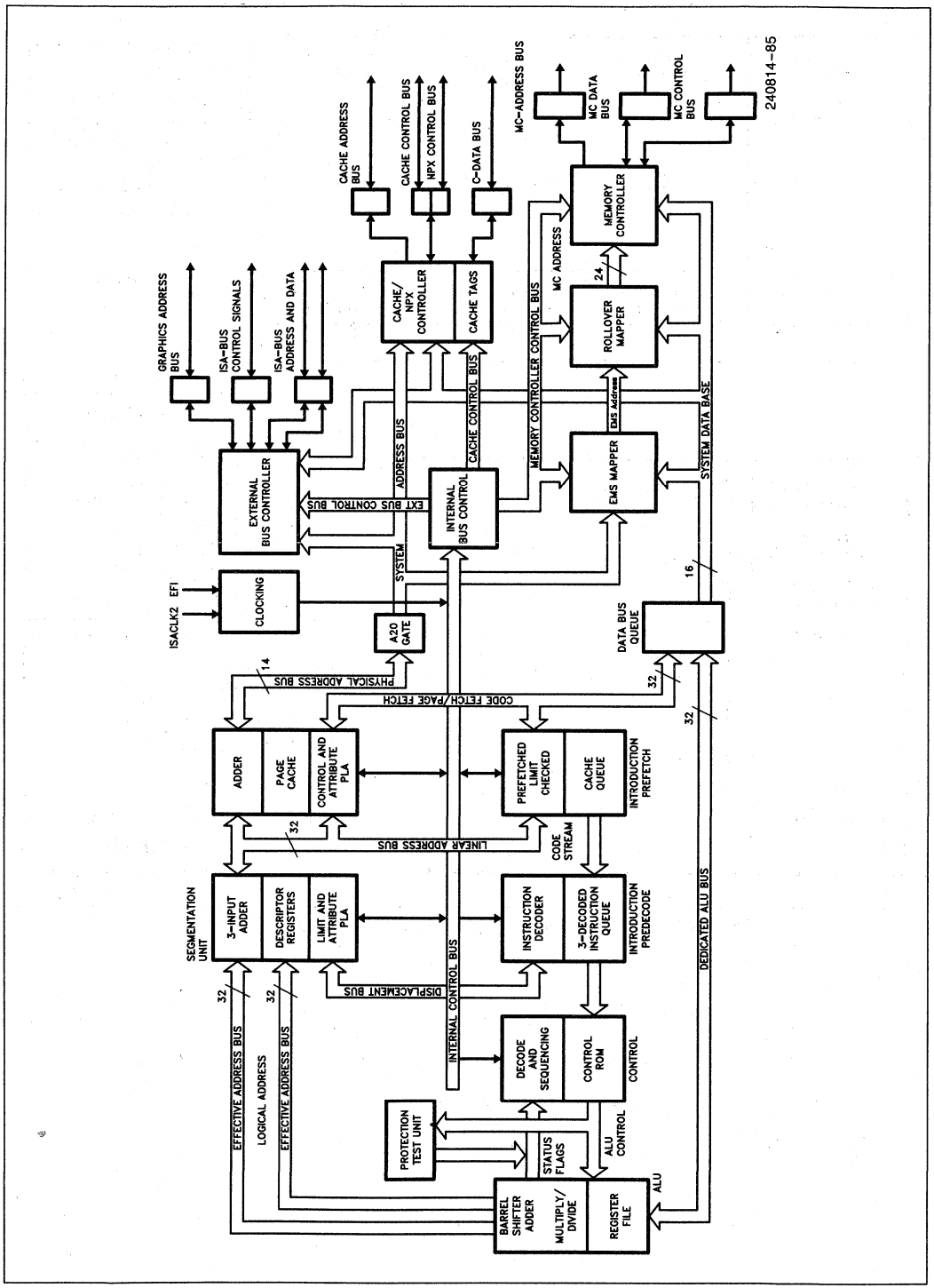


Figure 1-3b. 386™ SL Microprocessor Micro-Architecture

386™ SL Microprocessor: Central Processing Unit (CPU) and Memory Controller Subsystem

The 386 SL microprocessor is a highly-integrated, complete microprocessor and memory controller subsystem. At the heart of the 386 SL microprocessor is a CHMOS static 386 CPU core. The 386 CPU core has been fully optimized to reduce run-time power requirements, and includes a key architectural extension required by battery-operated systems.

The 386 SL processor is the first member of the 386 microprocessor product line to implement a CPU with the System Management Mode extension. The System Management Mode is a new CPU operating-mode which allows system vendors to rid their systems of the backwards-compatibility problems that plague battery-operated PCs. This 386 architecture extension eliminates portable-system conflicts by providing a safe, new operating level for the battery management firmware developed by system designers. With the 386™ SL CPU, firmware will execute transparently to every application, operating system and CPU mode, thus avoiding the compatibility conflicts which were once unavoidable.

The 386 SL microprocessor retains the paged-memory-management system, and all other key features which are common to the Intel386™ architecture. In addition, on-chip hardware implements the Expanded Memory Specification (E.M.S.) address translation compatible with the current Lotus/Intel/Microsoft (L.I.M.) E.M.S. 4.0 standard. Additional address-mapping and control logic integrated in the 386 SL CPU allows BIOS ROMs to be "shadowed" by faster memory devices, and supports a variety of common memory roll-over and back-fill schemes. The 386 SL CPU contains all of the control and interface logic needed to directly drive large main memory and an optional cache memory subsystem.

The 386 SL CPU contains bus drivers and control circuitry for two expansion interfaces. A Peripheral Interface Bus (PI-Bus) provides high-speed communication with devices which may reside on the same printed circuit board as the processor. The Industry Standard Architecture (ISA) bus provides a common interface for the wealth of third party ISA bus compatible I/O peripheral and expansion memory add-in boards. On-chip data-byte steering logic, address decoding and mapping logic automatically routes each memory or I/O operation to the appropriate local memory, cache, PI-Bus or ISA expansion bus.

All system configuration logic in the 386 SL processor subsystem is initialized under software control. The system designer only has to program the processor in order to support multiple system hardware designs where many devices of less flexibility were once required. System characteristics such as memory type, size, speed, organization, and mapping; cache size, organization and mapping; and peripheral selection, configuration and mapping are configured under software control. Thereafter, all memory and I/O transfer requests are automatically sent to the appropriate memory space or expansion bus, fully-transparent to existing operating system software and application programs.

Figure 1-3a shows the functional blocks and Figure 1.3b shows the microarchitecture of the 386 SL processor.

82360SL I/O: Integrated ISA Peripheral and Power Management Device

The 82360SL ISA Peripheral I/O contains dedicated logic to perform a number of CPU, memory, and peripheral support functions. The 82360SL device also contains an extensive set of programmable power management facilities which allow minimized system energy requirements for battery-powered portable computers.

The 82360SL includes a complete set of on-chip peripheral device functions including two 16450 compatible serial ports, one 8-bit Centronics interface or bi-directional parallel port, two 8254 compatible timer counters, two 8259 compatible interrupt controllers, two 8237 compatible DMA controllers, one 74LS612 compatible DMA page register, one 146818 compatible Real-time clock/calendar with 256 bytes of battery backed CMOS RAM and an integrated drive electronics (IDE) hard-disk-drive interface. The Intel 82360SL also contains highly programmable chip selects and complete peripheral interface logic for direct keyboard, FLASH memory and floppy disk controller support. The peripheral registers and functions behave exactly as the discrete components commonly found in industry-standard personal computers. The peripheral logic is enhanced for static operation by supporting write only registers as read/write.

The processor and memory support functions contained in the 82360SL device eliminate most of the external random-logic "glue" that might otherwise be required. The 82360SL device provides internal programmable-frequency clock generators for the CPU, backplane, and video subsystems. A programmable, low-power DRAM refresh timer is also provided to maintain system memory integrity during the power saving system stand-by and suspend states.

The 82360SL also contains a flexible set of hardware functions to support the growing sophistication in power management schemes required by portable systems. Numerous hardware timers, event monitors and I/O interfaces can programmably monitor and control system activity. Firmware developed by the system designer allocates and directs the hardware to fulfill the unique power management needs of a given system configuration.

All of the standard peripheral registers, clock-generation logic, and power-management facilities have been designed to ensure complete compatibility with existing operating systems and applications software.

Figure 1-4 shows the functional blocks and microarchitecture of the 82360SL I/O subsystem.

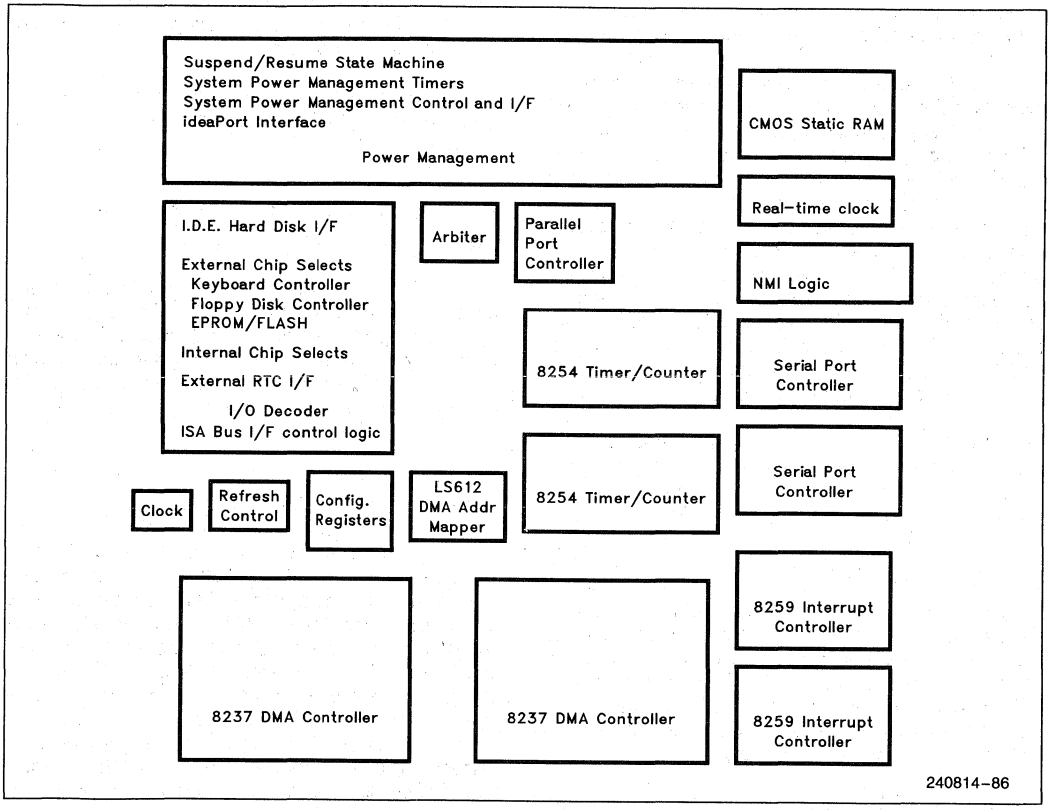


Figure 1-4a. 82360SL ISA Peripheral I/O Internal Functional Modules

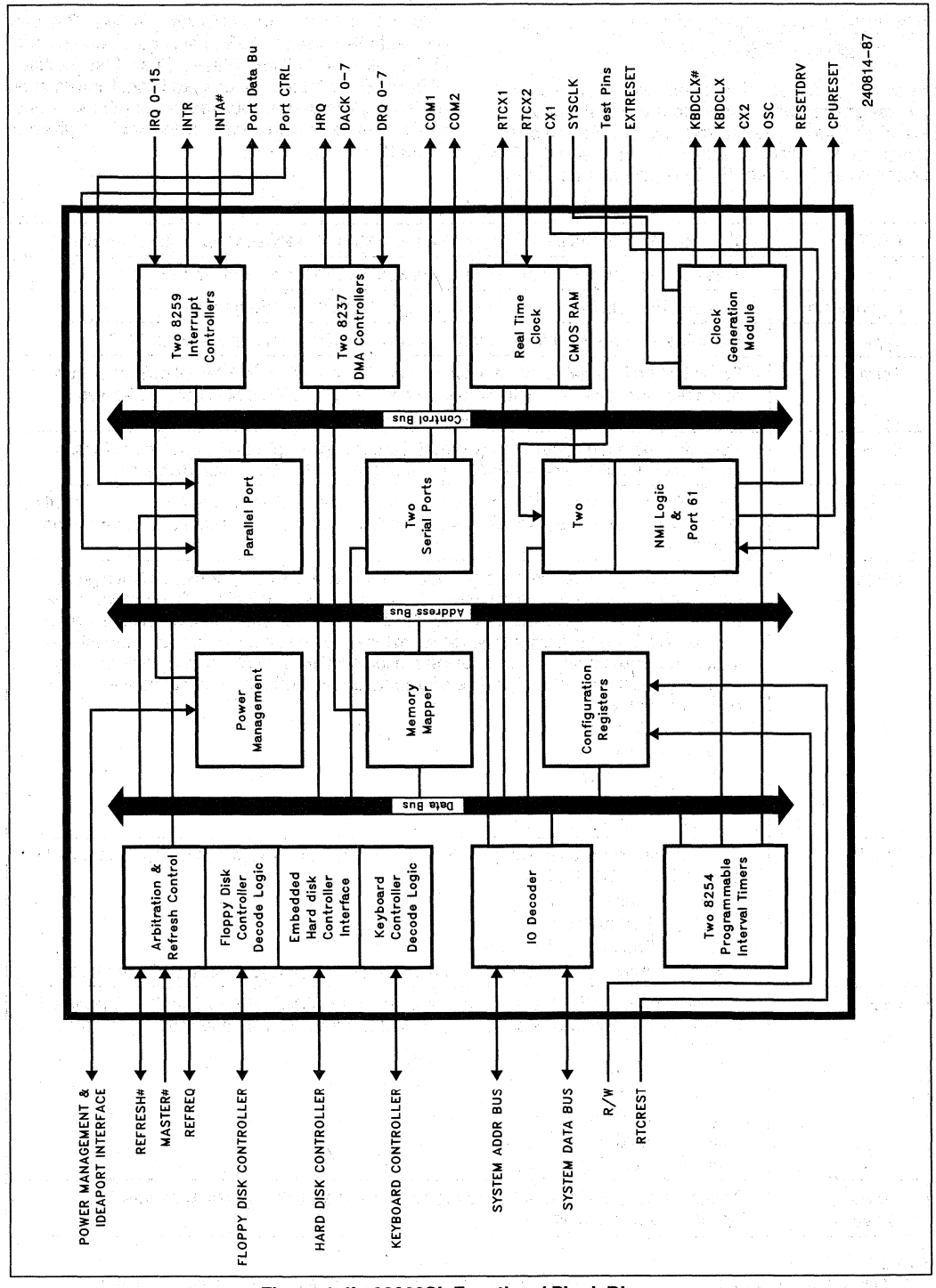


Figure 1-4b. 82360SL Functional Block Diagram

2.0 PIN ASSIGNMENTS AND SIGNAL CHARACTERISTICS

Section 2 provides information for the SL SuperSet pin assignment with respect to the signal mnuemonics. In addition to the package pin out diagrams, two tables are provided for easy location of signals. The first table lists the 386 SL CPU package device pinouts in the 227 pin Land Grid Array (LGA). The second table lists the 82360SL package device pinouts in the 196 lead JEDEC Plastic Quad Flat Package (PQFP). Both tables include additional information for the signals and associated pin numbers. A brief explanation of each column of the table is given in Table 2-1.

Table 2-1. Description of the Columns of Tables 2-2 and 2-3

PQFP	This column lists the pin numbers of the 82360SL in a Plastic Quad Flat Package.
LGA	This column lists the pin numbers of the 386 SL CPU in a Land Grid Array.
Signal Name	This column lists the signal name associated with the package pins.
Type	Indicates whether the pin is an Input (I), an Output (O) or an Input-Output (IO).
Term	Specifies the internal terminator on the pin. This could be an internal pull-up or pull-down resistor value or a hold circuit. To find out whether a pull-up or a pull-down is provided, use the STPCK (Stop Clock) column.
Drive	Specifies the drive current I_{OH} (Current Output Logic High) and I_{OL} (Current Output Logic Low) in milli-Amperes (mA) for output (O), and bi-directional (IO), pins.
Load	This column lists the maximum specified capacitive load which the buffer can directly drive in pico-Farads (pF) for each signal. This is specified for output and input-output pins only.
Susp.	This column specifies the state of the pin during a suspend operation. Input signals have the representation Tri/x where x is either a logic 0 or logic 1. This indicates that the input is internally isolated and that the internal termination on the pin is tri-stated or disabled. When in Suspend Mode an external logic value x is forced to the internal logic. The input can be driven to the same logic HIGH or LOW state by external logic with no current source or sink. The additional output buffer abbreviations are explained below. Tri - Tristated Actv - Active 0 - held low 1 - held high Hold - held at last state
Stpck.	This column specifies the state of the pin when the clock signal CPUCLK is internally stopped in the 386 SL CPU. Pu - Pulled up Pd - Pulled down Drv - Driven high, low or at the last state Actv - Active (Signal is driven and continues to operate or change logic states)
ONCE	This column specifies the state of the pin when the ONCE# (On Circuit Emulator) pin is asserted, allowing in-circuit testing while the device is still populated on the logic board. Tri - Floats Actv - Active 0 - held low 1 - held high Hold - held at last state
Derating Curve	This column specifies which derating curve(1) is used for each output buffer associated with the pin.

NOTE:
1. For more information on derating curves and how to use them, see Section 8 (Capacitive Derating Information).

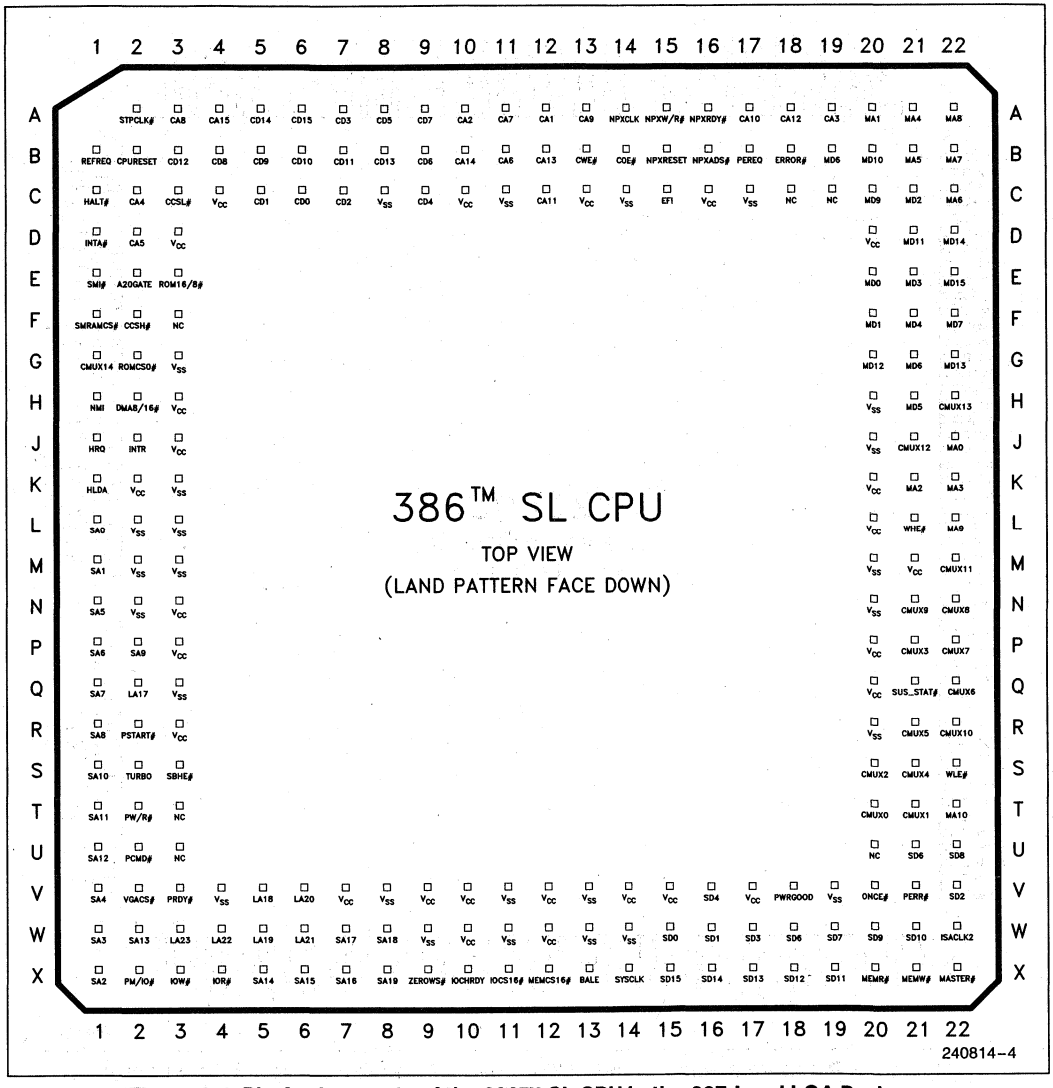


Figure 2-1. Pin Assignments of the 386™ SL CPU in the 227-Lead LGA Package (Top View—Land Pattern Facing Down, Component Marking Facing Up)

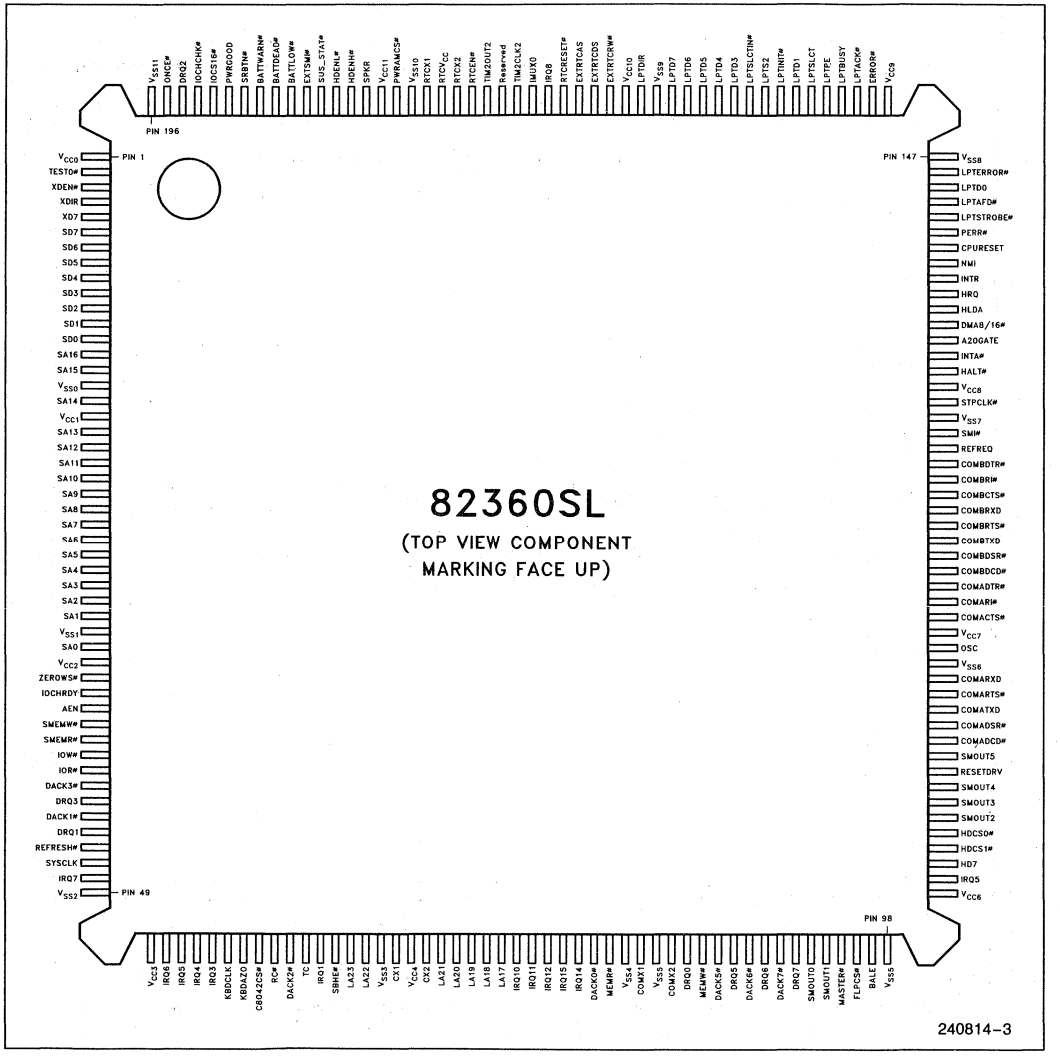


Figure 2-2. Pin Assignments for the 82360SL in a 196-Lead Plastic Quad Flat Package

Table 2-2. 386™ SL CPU Pin Characteristics

LGA Pin#	Signal Name	Type	Term	Drive	Load	Susp	Stpck	ONCE	Derating Curve
A02	STPCLK#	I	60K			Tri/1	Pu	Tri/1	
A03	CA8	O	Hold	4, 2	45	Hold	Drv	Hold	F
A04	CA15	O	Hold	4, 2	45	Hold	Drv	Hold	F
A05	CD14	IO	Hold	4, 2	50	Hold	Drv	Hold	I
A06	CD15	IO	Hold	4, 2	50	Hold	Drv	Hold	I
A07	CD3	IO	Hold	4, 2	50	Hold	Drv	Hold	I
A08	CD5	IO	Hold	4, 2	50	Hold	Drv	Hold	I
A09	CD7	IO	Hold	4, 2	50	Hold	Drv	Hold	I
A10	CA2	O	Hold	4, 2	60	Hold	Drv	Hold	F
A11	CA7	O	Hold	4, 2	45	Hold	Drv	Hold	F
A12	CA1	O	Hold	4, 2	45	Hold	Drv	Hold	F
A13	CA9	O	Hold	4, 2	45	Hold	Drv	Hold	F
A14	NPXCLK	O	Hold	4, 2	20	Hold	Drv	Hold	F
A15	NPXW/R#	O	Hold	4, 2	30	Hold	Drv	Hold	F
A16	NPXRDY#	I	60K			Tri/1	Pu	Tri/1	
A17	CA10	O	Hold	4, 2	45	Hold	Drv	Hold	F
A18	CA12	O	Hold	4, 2	45	Hold	Drv	Hold	F
A19	CA3	O	Hold	4, 2	45	Hold	Drv	Hold	F
A20	MA1	O	Hold	4, 2	300	Hold	Drv	Hold	E
A21	MA4	O	Hold	4, 2	300	Hold	Drv	Hold	E
A22	MA8	O	Hold	4, 2	300	Hold	Drv	Hold	E
B01	REFREQ	I	Hold			Actv	Actv	Tri/0	
B02	CPURESET	I	20K			Tri/0	Pd	Tri/0	
B03	CD12	IO	Hold	4, 2	50	Hold	Drv	Hold	I
B04	CD8	IO	Hold	4, 2	50	Hold	Drv	Hold	I
B05	CD9	IO	Hold	4, 2	50	Hold	Drv	Hold	I
B06	CD10	IO	Hold	4, 2	50	Hold	Drv	Hold	I
B07	CD11	IO	Hold	4, 2	50	Hold	Drv	Hold	I
B08	CD13	IO	Hold	4, 2	50	Hold	Drv	Hold	I
B09	CD6	IO	Hold	4, 2	50	Hold	Drv	Hold	I
B10	CA14	O	Hold	4, 2	45	Hold	Drv	Hold	F
B11	CA6	O	Hold	4, 2	45	Hold	Drv	Hold	F
B12	CA13	O	Hold	4, 2	45	Hold	Drv	Hold	F

NOTES:
1. Tri/1 indicates a tri-stateable output with pull-up.
2. Tri/0 indicates a tri-stateable output with pull-down.
3. CMUX 8–11 (RASxx#) are ACTIVE when the 386™ SL CPU Memory Controller is programmed in the DRAM controller mode with Suspend Refresh enabled. Otherwise, these signals are HOLD.

Table 2-2. 386™ SL CPU Pin Characteristics (Continued)

LGA Pin#	Signal Name	Type	Term	Drive	Load	Susp	Stpck	ONCE	Derating Curve
B13	CWE#	O	Hold	4, 2	45	Hold	Drv	Hold	I
B14	COE#	O	Hold	4, 2	45	Hold	Drv	Hold	F
B15	NPXRESET#	O	Hold	4, 2	20	Hold	Drv	Hold	F
B16	NPXADS#	O	Hold	4, 2	30	Hold	Drv	Hold	F
B17	PEREQ	I	20K			Tri/0	Pd	Tri/0	
B18	ERROR#	I	60K			Tri/1	Pu	Tri/1	
B19	MD8	IO	Hold	4, 2	68	Hold	Drv	Hold	H
B20	MD10	IO	Hold	4, 2	68	Hold	Drv	Hold	H
B21	MA5	O	Hold	4, 2	300	Hold	Drv	Hold	E
B22	MA7	O	Hold	4, 2	300	Hold	Drv	Hold	E
C01	HALT#	O	Hold	4, 2	65	Hold	Drv	Hold	G
C02	CA4	O	Hold	4, 2	45	Hold	Drv	Hold	F
C03	CCSL#	O	Hold	4, 2	35	Hold	Drv	Hold	F
C04	V_{CC}								
C05	CD1	IO	Hold	4, 2	50	Hold	Drv	Hold	I
C06	CD0	IO	Hold	4, 2	50	Hold	Drv	Hold	
C07	CD2	IO	Hold	4, 2	50	Hold	Drv	Hold	I
C08	V_{SS}								
C09	CD4	IO	Hold	4, 2	50	Hold	Drv	Hold	I
C10	V_{CC}								
C11	V_{SS}								
C12	CA11	O	Hold	4, 2	45	Hold	Drv	Hold	F
C13	V_{CC}								
C14	V_{SS}								
C15	EFI	I							
C16	V_{CC}								
C17	V_{SS}								
C18	BUSY#	I	60K			Tri/1	Pu	Tri/1	
C19	nc								
C20	MD9	IO	Hold	4, 2	68	Hold	Drv	Hold	H
C21	MD2	IO	Hold	4, 2	68	Hold	Drv	Hold	H
C22	MA6	O	Hold	4, 2	300	Hold	Drv	Hold	E
D01	INTA#	O	Hold	4, 2	65	Hold	Drv	Hold	G

Table 2-2. 386™ SL CPU Pin Characteristics (Continued)

LGA Pin #	Signal Name	Type	Term	Drive	Load	Susp	Stpck	ONCE	Derating Curve
D02	CA5	O	Hold	4, 2	45	Hold	Drv	Hold	F
D03	V_{CC}								
D20	V_{CC}								
D21	MD11	IO	Hold	4, 2	68	Hold	Drv	Hold	H
D22	MD14	IO	Hold	4, 2	68	Hold	Drv	Hold	H
E01	SMI#	I	60K			Tri/1	Pu	Tri/1	
E02	A20GATE	I	20K			Tri/0	Pd	Tri/0	
E03	ROM16/8#	I	60K			Tri/1	Pu	Tri/1	
E20	MD0	IO	Hold	4, 2	68	Hold	Drv	Hold	H
E21	MD3	IO	Hold	4, 2	68	Hold	Drv	Hold	H
E22	MD15	IO	Hold	4, 2	68	Hold	Drv	Hold	H
F01	SMRAMCS#	O	Hold	4, 2	65	Drv	Drv	Hold	G
F02	CCSH#	O	Hold	4, 2	35	Hold	Drv	Hold	F
F03	NC								
F20	MD1	IO	Hold	4, 2	68	Hold	Drv	Hold	H
F21	MD4	IO	Hold	4, 2	68	Hold	Drv	Hold	H
F22	MD7	IO	Hold	4, 2	68	Hold	Drv	Hold	H
G01	CMUX14	O	Hold	4, 2	65	Hold	Drv	Hold	G
G02	ROMCS0#	O	Hold	4, 2	65	Hold	Drv	Hold	G
G03	V_{SS}								
G20	MD12	IO	Hold	4, 2	68	Hold	Drv	Hold	H
G21	MD6	IO	Hold	4, 2	68	Hold	Drv	Hold	H
G22	MD13	IO	Hold	4, 2	68	Hold	Drv	Hold	H
H01	NMI	I	20K			Tri/0	Pd	Tri/0	
H02	DMA8/16#	I	60K			Tri/1	Pu	Tri/1	
H03	V_{CC}								
H20	V_{SS}								
H21	MD5	IO	Hold	4, 2	68	Hold	Drv	Hold	H
H22	CMUX13	IO	Hold	4, 2	68	Hold	Drv	Hold	H
J01	HRQ	I	20K			Tri/0	Pd	Tri/0	
J02	INTR	I	20K			Tri/0	Pd	Tri/0	
J03	V_{CC}								
J20	V_{SS}								

Table 2-2. 386™ SL CPU Pin Characteristics (Continued)

LGA Pin #	Signal Name	Type	Term	Drive	Load	Susp	Stpck	ONCE	Derating Curve
J21	CMUX12	IO	Hold	4, 2	68	Hold	Drv	Hold	H
J22	MA0	O	Hold	4, 2	300	Hold	Drv	Hold	E
K01	HLDA	O	Hold	4, 2	65	Hold	Drv	Hold	G
K02	V_{CC}								
K03	V_{SS}								
K20	V_{CC}								
K21	MA2	O	Hold	4, 2	300	Hold	Drv	Hold	E
K22	MA3	O	Hold	4, 2	300	Hold	Drv	Hold	E
L01	SA0	IO	Hold	24, 4	240	Hold	Drv	Hold	C
L02	V_{SS}								
L03	V_{SS}								
L20	V_{CC}								
L21	WHE#	O	Hold	4, 2	300	Tri/1	Drv	Hold	D
L22	MA9	O	Hold	4, 2	300	Hold	Drv	Hold	E
M01	SA1	IO	Hold	24, 4	240	Hold	Drv	Hold	C
M02	V_{SS}								
M03	V_{SS}								
M20	V_{SS}								
M21	V_{CC}								
M22	CMUX11	O	Hold	4, 2	144	Actv(3)	Drv	Hold	A
N01	SA5	IO	Hold	24, 4	240	Hold	Drv	Hold	C
N02	V_{SS}								
N03	V_{CC}								
N20	V_{SS}								
N21	CMUX9	O	Hold	4, 2	144	Actv(3)	Drv	Hold	A
N22	CMUX8	O	Hold	4, 2	144	Actv(3)	Drv	Hold	A
P01	SA6	IO	Hold	24, 4	240	Hold	Drv	Hold	C
P02	SA9	IO	Hold	24, 4	240	Hold	Drv	Hold	C
P03	V_{CC}								
P20	V_{CC}								
P21	CMUX3	O	Hold	4, 2	72	Tri/0	Drv	Hold	H
P22	CMUX7	O	Hold	4, 2	72	Hold	Drv	Hold	H
Q01	SA7	IO	Hold	24, 4	240	Hold	Drv	Hold	C

Table 2-2. 386™ SL CPU Pin Characteristics (Continued)

LGA Pin#	Signal Name	Type	Term	Drive	Load	Susp	Stpck	ONCE	Derating Curve
Q02	LA17	IO	Hold	24, 4	240	Hold	Drv	Hold	C
Q03	V_{SS}								
Q20	V_{CC}								
Q21	SUS_STAT#	I	60K			Actv	Actv	Tri/1	
Q22	CMUX6	O	Hold	4, 2	72	Tri/0	Drv	Hold	H
R01	SA8	IO	Hold	24, 4	240	Hold	Drv	Hold	C
R02	PSTART#	O	Hold	4, 2	65	Hold	Drv	Hold	G
R03	V_{CC}								
R20	V_{SS}								
R21	CMUX5	O	Hold	4, 2	72	Tri/1	Drv	Hold	H
R22	CMUX10	O	Hold	4, 2	144	Actv(3)	Drv	Hold	A
S01	SA10	IO	Hold	24, 4	240	Hold	Drv	Hold	C
S02	TURBO	I	60K			Tri/1	Pu	Tri/1	
S03	SBHE#	IO	Hold	24, 4	240	Hold	Drv	Hold	C
S20	CMUX2	O	Hold	4, 2	72	Tri/0	Drv	Hold	H
S21	CMUX4	O	Hold	4, 2	72	Tri/0	Drv	Hold	H
S22	WLE#	O	Hold	4, 2	300	Tri/1	Drv	Hold	D
T01	SA11	IO	Hold	24, 4	240	Hold	Drv	Hold	C
T02	PW/R#	O	Hold	4, 2	65	Hold	Drv	Hold	G
T03	NC								
T20	CMUX0	O	Hold	4, 2	72	Tri/0	Drv	Hold	H
T21	CMUX1	O	Hold	4, 2	72	Tri/0	Drv	Hold	H
T22	MA10	O	Hold	4, 2	300	Hold	Drv	Hold	E
U01	SA12	IO	Hold	24, 4	240	Hold	Drv	Hold	C
U02	PCMD#	O	Hold	4, 2	65	Hold	Drv	Hold	G
U03	NC								
U20	NC								
U21	SD5	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
U22	SD8	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
V01	SA4	IO	Hold	24, 4	240	Hold	Drv	Hold	C
V02	VGACS#	O	Hold	4, 2	65	Hold	Drv	Hold	G
V03	PRDY#	I	60K			Tri/1	Pu	Tri/1	

Table 2-2. 386™ SL CPU Pin Characteristics (Continued)

LGA Pin #	Signal Name	Type	Term	Drive	Load	Susp	Stpck	ONCE	Derating Curve
V04	V_{SS}								
V05	LA18	IO	Hold	24, 4	240	Hold	Drv	Hold	C
V06	LA20	IO	Hold	24, 4	240	Hold	Drv	Hold	C
V07	V_{CC}								
V08	V_{SS}								
V09	V_{CC}								
V10	V_{CC}								
V11	V_{SS}								
V12	V_{CC}								
V13	V_{SS}								
V14	V_{CC}								
V15	V_{CC}								
V16	SD4	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
V17	V_{CC}								
V18	PWRGOOD	I				Actv	Actv	Tri/0	
V19	V_{SS}								
V20	ONCE#	I	60K			Tri/1	Pu	Actv	
V21	PERR#	O	Hold	4, 2	68	Hold	Drv	Hold	H
V22	SD2	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
W01	SA3	IO	Hold	24, 4	240	Hold	Drv	Hold	C
W02	SA13	IO	Hold	24, 4	240	Hold	Drv	Hold	C
W03	LA23	IO	Hold	24, 4	240	Hold	Drv	Hold	C
W04	LA22	IO	Hold	24, 4	240	Hold	Drv	Hold	C
W05	LA19	IO	Hold	24, 4	240	Hold	Drv	Hold	C
W06	LA21	IO	Hold	24, 4	240	Hold	Drv	Hold	C
W07	SA17	O	Hold	24, 4	240	Hold	Drv	Hold	C
W08	SA18	O	Hold	24, 4	240	Hold	Drv	Hold	C
W09	V_{SS}								
W10	V_{CC}								
W11	V_{SS}								
W12	V_{CC}								
W13	V_{SS}								
W14	V_{SS}								

Table 2-2. 386™ SL CPU Pin Characteristics (Continued)

LGA Pin#	Signal Name	Type	Term	Drive	Load	Susp	Stpck	ONCE	Derating Curve
W15	SD0	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
W16	SD1	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
W17	SD3	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
W18	SD6	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
W19	SD7	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
W20	SD9	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
W21	SD10	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
W22	ISACLK2	I							
X01	SA2	IO	Hold	24, 4	240	Hold	Drv	Hold	C
X02	PM/IO#	O	Hold	4, 2	65	Hold	Drv	Hold	G
X03	IOW#	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	B
X04	IOR#	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	B
X05	SA14	IO	Hold	24, 4	240	Hold	Drv	Hold	C
X06	SA15	IO	Hold	24, 4	240	Hold	Drv	Hold	C
X07	SA16	IO	Hold	24, 4	240	Hold	Drv	Hold	C
X08	SA19	O	Hold	24, 4	240	Hold	Drv	Hold	C
X09	ZEROWS#	I	300			Tri/1	Pu	Tri/1	
X10	IOCHRDY	IO	300	24, 4	240	Tri/1	Pu	Tri/1	D
X11	IOCS16#	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
X12	MEMCS16#	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
X13	BALE	O	Hold	24, 4	240	Hold	Drv	Hold	D
X14	SYSCLK	O	Hold	4, 2	120	Hold	Drv	Hold	G
X15	SD15	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
X16	SD14	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
X17	SD13	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
X18	SD12	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
X19	SD11	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	D
X20	MEMR#	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	B
X21	MEMW#	IO	60K	24, 4	240	Tri/1	Pu	Tri/1	B
X22	MASTER#	I	60K			Tri/1	Pu	Tri/1	

Table 2-3. 82360SL Pin Characteristics

PQFP Pin #	Signal Name	Type	Term	Drive	Load	Susp	ONCE	Derating Curve
B1	V_{CC}							
B2	TESTO#	I	60K			Tri	Pu	
B3	XDEN#	O		12	50	Tri	Tri	M
B4	XDIR	O		12	50	Tri	Tri	M
B5	XD7	IO	60K	24	100	Tri	Tri/1	J
B6	SD7	IO		24	240	Tri	Tri	J
B7	SD6	IO		24	240	Tri	Tri	J
B8	SD5	IO		24	240	Tri	Tri	J
B9	SD4	IO		24	240	Tri	Tri	J
B10	SD3	IO		24	240	Tri	Tri	J
B11	SD2	IO		24	240	Tri	Tri	J
B12	SD1	IO		24	240	Tri	Tri	J
B13	SD0	IO		24	240	Tri	Tri	J
B14	SA16	IO		24	240	Tri	Tri	J
B15	SA15	IO		24	240	Tri	Tri	J
B16	V_{SS}							
B17	SA14	IO		24	240	Tri	Tri	J
B18	V_{CC}							
B19	SA13	IO		24	240	Tri	Tri	J
B20	SA12	IO		24	240	Tri	Tri	J
B21	SA11	IO		24	240	Tri	Tri	J
B22	SA10	IO		24	240	Tri	Tri	J
B23	SA9	IO		24	240	Tri	Tri	J
B24	SA8	IO		24	240	Tri	Tri	J
B25	SA7	IO		24	240	Tri	Tri	J
B26	SA6	IO		24	240	Tri	Tri	J
B27	SA5	IO		24	240	Tri	Tri	J
B28	SA4	IO		24	240	Tri	Tri	J
B29	SA3	IO		24	240	Tri	Tri	J
B30	SA2	IO		24	240	Tri	Tri	J
B31	SA1	IO		24	240	Tri	Tri	J
B32	V_{SS}							

Table 2-3. 82360SL Pin Characteristics (Continued)

PQFP Pin #	Signal Name	Type	Term	Drive	Load	Susp	ONCE	Derating Curve
B33	SA0	IO		24	240	Tri	Tri	J
B34	V_{CC}							
B35	ZEROWS#	OD		24	240	Tri	Tri	K
B36	IOCHRDY	OD		24	240	Tri	Tri	K
B37	AEN	O		24	240	Tri	Tri	K
B38	SMEMW#	O	60K	24	240	Tri	Tri	K
B39	SMEMR#	O	60K	24	240	Tri	Tri	K
B40	IOW#	IO		24	240	Tri	Tri	K
B41	IOR#	IO		24	240	Tri	Tri	K
B42	DACK3#	O		12	50	Tri	Tri	M
B43	DRQ3	I	20K			Pd	Pd	
B44	DACK1#	O		12	50	Tri	Tri	M
B45	DRQ1	I	20K			Pd	Pd	
B46	REFRESH#	OD	300	16	240	Tri	Pu	K
B47	SYSCLK	I	100K			Pd	Pd	
B48	IRQ7	I	10K			Tri	Pu	
B49	V_{SS}							
B50	V_{CC}							
B51	IRQ6	I	10K			Tri	Pu	
B52	IRQ5	I	10K			Tri	Pu	
B53	IRQ4	I	10K			Tri	Pu	
B54	IRQ3	I	10K			Tri	Pu	
B55	KBDCLK	O		12	50	Tri	Tri	M
B56	KBDA20	I	20K			Pd	Pd	
B57	C8042CS#	O		12	50	Tri	Tri	M
B58	RC#	I	60K			Tri	Pu	
B59	DACK2#	O		12	50	Tri	Tri	M
B60	TC	O		24	240	Tri	Tri	K
B61	IRQ1	I	10K			Tri	Pu	
B62	SBHE#	O		24	240	Tri	Tri	K
B63	LA23	IO		24	240	Tri	Tri	J
B64	LA22	IO		24	240	Tri	Tri	J
B65	V_{SS}							

Table 2-3. 82360SL Pin Characteristics (Continued)

PQFP Pin#	Signal Name	Type	Term	Drive	Load	Susp	ONCE	Derating Curve
B66	CX1	I					Actv	
B67	V_{CC}							
B68	CX2	O					Actv	
B69	LA21	IO		24	240	Tri	Tri	J
B70	LA20	IO		24	240	Tri	Tri	J
B71	LA19	IO		24	240	Tri	Tri	J
B72	LA18	IO		24	240	Tri	Tri	J
B73	LA17	IO		24	240	Tri	Tri	J
B74	IRQ10	I	10K			Tri	Pu	
B75	IRQ11	I	10K			Tri	Pu	
B76	IRQ12	I	10K			Tri	Pu	
B77	IRQ15	I	10K			Tri	Pu	
B78	IRQ14	I	10K			Tri	Pu	
B79	DACK0#	O		12	50	Tri	Tri	M
B80	MEMR#	IO		24	240	Tri	Tri	K
B81	V_{SS}							
B82	COMX1	I					Actv	
B83	V_{CC}							
B84	COMX2	O					Actv	
B85	DRQ0	I	20K			Pd	Pd	
B86	MEMW#	IO		24	240	Tri	Tri	K
B87	DACK5#	O		12	50	Tri	Tri	M
B88	DRQ5	I	20K			Pd	Pd	
B89	DACK6#	O		12	50	Tri	Tri	M
B90	DRQ6	I	20K			Pd	Pd	
B91	DACK7#	O		12	50	Tri	Tri	M
B92	DRQ7	I	20K			Pd	Pd	
B93	SMOUT0	O		12	50	Tri	Tri	M
B94	SMOUT1	O		12	50	Tri	Tri	M
B95	MASTER#	I				Tri	Tri	
B96	FLPCS#	O		12	50	Tri	Tri	M
B97	BALE	I	100K			Pd	Pd	
B98	V_{SS}							
B99	V_{CC}							
B100	IRQ9	I	10K			Tri	Pu	

Table 2-3. 82360SL Pin Characteristics (Continued)

PQFP Pin #	Signal Name	Type	Term	Drive	Load	Susp	ONCE	Derating Curve
B101	HD7	IO	60K	24	100	Tri	Tri/1	J
B102	HDCS1#	O		12	50	Tri	Tri	M
B103	HDCS0#	O		12	50	Tri	Tri	M
B104	SMOUT2	O		12	50	Tri	Tri	M
B105	SMOUT3	O		12	50	Tri	Tri	M
B106	SMOUT4	O		12	50	Tri	Tri	M
B107	RESETDRV	O		24	240	Tri	Tri	K
B108	SMOUT5	O		12	50	Tri	Tri	M
B109	COMADCD#	I	60K			Tri	Pu	
B110	COMADSR#	I	60K			Tri	Pu	
B111	COMATXD	O		12	50	Tri	Tri	M
B112	COMARTS#	O		12	50	Tri	Tri	M
B113	COMARXD	I	20K			Pd	Pd	
B114	V_{SS}							
B115	OSC	O		24	240	Tri	Tri	K
B116	V_{CC}							
B117	COMACTS#	I	60K			Tri	Pu	
B118	COMARI#	I	60K			Actv	Pu	
B119	COMADTR#	O		12	50	Tri	Tri	M
B120	COMBDCD#	I	60K			Tri	Pu	
B121	COMBDSR#	I	60K			Tri	Pu	
B122	COMBTXD	O		12	50	Tri	Tri	M
B123	COMBRTS#	O		12	50	Tri	Tri	M
B124	COMBRXD	I	20K			Pd	Pd	
B125	COMBCTS#	I	60K			Tri	Pu	
B126	COMBRI#	I	60K			Actv	Pu	
B127	COMBDTR#	O		12	50	Tri	Tri	M
B128	REFREQ	O		12	50	Actv(1)	Actv	M
B129	SMI#	O		12	50	Tri	Tri	M
B130	V_{SS}							
B131	STPCLK#	O		12	50	Tri	Tri	M
B132	V_{CC}							

NOTE:
1. Programmable, active only when suspend refresh is enabled.

Table 2-3. 82360SL Pin Characteristics (Continued)

PQFP Pin #	Signal Name	Type	Term	Drive	Load	Susp	ONCE	Derating Curve
B133	HALT#	I	60K			Tri	Pu	
B134	INTA#	I	60K			Tri	Pu	
B135	A20GATE	O		12	50	Tri	Tri	M
B136	DMA8/16#	O		12	50	Tri	Tri	M
B137	HLDA	I	20K			Pd	Pd	
B138	HRQ	O		12	50	Tri	Tri	M
B139	INTR	O		12	50	Tri	Tri	M
B140	NMI	O		12	50	Tri	Tri	M
B141	CPURESET	O		12	50	Tri	Tri	M
B142	PERR#	I	60K			Tri	Pu	
B143	LPTSTROBE#	OD	4K7	12	100	Tri	Tri	L
B144	LPTAFD#	OD	4K7	12	100	Tri	Tri	L
B145	LPTD0	IO	20K	8	100	Pd	Tri/0	L
B146	LPTERROR#	I	60K			Tri	Pu	
B147	V_{SS}							
B148	V_{CC}							
B149	ERROR#	I	60K			Tri	Pu	
B150	LPTACK#	I	60K			Tri	Pu	
B151	LPTBUSY	I	20K			Pd	Pd	
B152	LPTPE	I	20K			Pd	Pd	
B153	LPTSLCT	I	20K			Pd	Pd	
B154	LPTD1	IO	20K	8	100	Pd	Tri/0	L
B155	LPTINIT#	OD	4K7	12	100	Tri	Tri	L
B156	LPTD2	IO	20K	8	100	Pd	Tri/0	L
B157	LPTSLCTIN#	OD	4K7	12	100	Tri	Tri	L
B158	LPTD3	IO	20K	8	100	Pd	Tri/0	L
B159	LPTD4	IO	20K	8	100	Pd	Tri/0	L
B160	LPTD5	IO	20K	8	100	Pd	Tri/0	L
B161	LPTD6	IO	20K	8	100	Pd	Tri/0	L
B162	LPTD7	IO	20K	8	100	Pd	Tri/0	L
B163	V_{SS}							
B164	LPTDIR	OD	4K7	12	100	Tri	Tri	L

Table 2-3. 82360SL Pin Characteristics (Continued)

PQFP Pin #	Signal Name	Type	Term	Drive	Load	Susp	ONCE	Derating Curve
B165	V_{CC}							
B166	EXTRTCRW#	O	60K	12	50	Pu	Tri/1	M
B167	EXTRTCDS	O	60K	12	50	Pu	Tri/1	M
B168	EXTRTCAS	O	60K	12	50	Pu	Tri/1	M
B169	RTCRESET#	I	60K			Pu	Pu	
B170	IRQ8	I	60K			Pu	Pu	
B171	IMUX0	I	20K			Pd	Pd	
B172	TIM2CLK2	I	20K			Pd	Pd	
B173	Reserved							
B174	TIM2OUT2	O		12	50	Tri	Tri	M
B175	RTCEN#	I	60K			Pu	Pu	
B176	RTCX2	O					Actv	
B177	RTCVCC							
B178	RTCX1	I					Actv	
B179	V_{SS}							
B180	SMRAMCS#	I	60K			Tri	Pu	
B181	V_{CC}							
B182	SPKR	O		12	50	Tri	Tri	M
B183	HDENH#	O		12	50	Tri	Tri	M
B184	HDENL#	O		12	50	Tri	Tri	M
B185	SUS_STAT#	O		12	50	Drv	Actv	M
B186	EXTSMI#	I	60K			Tri	Pu	
B187	BATTLOW#	I	60K			Pu	Pu	
B188	BATTDEAD#	I	60K			Pu	Pu	
B189	BATTWARN#	I	60K			Tri	Pu	
B190	SRBTN#	I	60K			Pu	Pu	
B191	PWRGOOD	I	60K			Pu	Pu	
B192	IOCS16#	I						
B193	IOCHCK#	I						
B194	DRQ2	I	20K			Pd	Pd	
B195	ONCE#	I	60K			Tri	Pu	
B196	V_{SS}							

3.0 SIGNAL DESCRIPTIONS

386™ SL Microprocessor

The following table provides a brief description of the signals of the 386 SL CPU. Signal names which end with the character "#" indicate that the corresponding signal is low when active.

<table>
<tr><th>Symbol</th><th>Name and Function</th></tr>
<tr><td>A20GATE</td><td>A20 Gate: This active HIGH input signal controls the 386 SL CPU A20 address line. When HIGH this signal forces the 386 SL CPU to mask off (force LOW) the internal physical address signal A20. When this signal is LOW, the internal physical address signal A20 is available on the System Address (SA) bus. When A20 gate is inactive this allows emulation of the 8086 1 Mbyte address "wrap-around".</td></tr>
<tr><td>BALE</td><td>Bus Address Latch Enable (ISA bus signal): This active HIGH output signal is used for two purposes. BALE is used to latch the address lines on the LA bus (LA17–LA23) on the falling edge of BALE. BALE is also used to qualify ISA bus cycles for signals on the Peripherial Interface (PI) bus (PM/IO# and PW/R#). On the falling edge of BALE, PM/IO# and PW/R# can be sampled to determine the type of ISA bus cycle that is going to occur. BALE may be used to qualify and generate buffered control and status signals to the ISA expansion bus. The PI bus signal decoding is as follows:
<table>
<tr><th>Type of Bus Cycle</th><th>PM/IO#</th><th>PW/R#</th></tr>
<tr><td>Memory Read</td><td>1</td><td>0</td></tr>
<tr><td>Memory Write</td><td>1</td><td>1</td></tr>
<tr><td>I/O Read</td><td>0</td><td>0</td></tr>
<tr><td>I/O Write</td><td>0</td><td>1</td></tr>
<tr><td>Interrupt Acknowledge</td><td>0</td><td>1</td></tr>
<tr><td>HALT (address = 2)*</td><td>1</td><td>1</td></tr>
<tr><td>Shutdown (address = 0)*</td><td>1</td><td>1</td></tr>
</table>
*Note that BALE is not generated for these cycles, however the PM/IO# and PW/R# will reflect these states during HALT and Shutdown bus cycles where BALE is driven in typical ISA bus systems. Memory read/write, IO read/write and interrupt/interrupt acknowledge cycles correspond to the standard ISA bus cycle.</td></tr>
<tr><td>BUSY#</td><td>BUSY: This active LOW input signal indicates a busy condition from a math co-processor (MCP).</td></tr>
<tr><td>CA[15:1]</td><td>Cache Address Bus: This is the address bus output used to select the memory cell in the cache memory. The CA2 signal is also connected to the CMD0# input of the MCP indicating Opcode (when high) or Data (when low) during a write cycle and control/status register (high) or data register (low) during a read. CA2 is used to address the upper or lower DWORD port of the MCP.</td></tr>
<tr><td>CCSH#</td><td>Cache Chip Select High Byte: This active LOW output is used to enable the upper byte of the cache SRAMs. This signal should be connected to the upper byte cache SRAM chip-select input.</td></tr>
<tr><td>CCSL#</td><td>Cache Chip Select Low Byte: This active LOW output is used to enable the lower byte of the cache SRAMs. This signal should be connected to the lower byte cache SRAM chip-select input.</td></tr>
<tr><td>CD[15:0]</td><td>Cache Data Bus: This is the bi-directional data bus used to transfer data between the cache SRAMs and the 386 SL CPU. The Cache Data bus is also used to transfer data between the MCP and the 386 SL CPU.</td></tr>
</table>

386™ SL Microprocessor Signal Descriptions (Continued)

Symbol	Name and Function
CMUX0	**CPU Multiplexed Pin Zero:** This output signal has two functions. When the 386 SL CPU Memory Controller Unit is configured as a DRAM controller then this pin becomes "CASL3#" and should be connected to the lower byte of DRAM bank 3 CAS# input. When the 386 SL CPU Memory Controller Unit is configured as an SRAM controller this signal becomes the direction control (DIR) and should be connected to the direction control input of the SRAM data transceiver.
CMUX1	**CPU Multiplexed Pin One:** This output signal has two functions. When the 386 SL CPU Memory Controller Unit is configured as a DRAM controller then this pin becomes "CASH3#" and should be connected to the upper byte of DRAM bank 3 CAS# input. When the 386 SL CPU Memory Controller Unit is configured as a SRAM controller this signal becomes "LE" and should be connected to the latch enable input of the SRAM address latch. This pin is disabled when SUS_STAT# is active (LOW) and the system is not performing a suspend refresh operation. When the pin is disabled the output is sustained at the previous state by internal "keepers".
CMUX2	**CPU Multiplexed Pin Two:** This output signal has two functions. When the 386 SL CPU Memory Controller Unit is configured as a DRAM controller this pin becomes "CASL2#" and should be connected to the lower byte of DRAM bank 2 CAS# input. When the 386 SL CPU Memory Controller Unit is configured as a SRAM controller this pin becomes "DEN3#" and should be connected to the data transceiver enable input for bank 3 of the SRAM memory subsystem. This pin is disabled when SUS_STAT# is active (LOW) and the system is not performing a suspend refresh operation. When the pin is disabled the output is sustained at the previous state by internal "keepers".
CMUX3	**CPU Multiplexed Pin Three:** This output signal has two functions. When the 386 SL CPU Memory Controller Unit is configured as a DRAM controller this pin becomes "CASH2#" and should be connected to the upper byte of DRAM bank 2 CAS# input. When the 386 SL CPU Memory Controller Unit is configured as a SRAM controller this pin becomes "DEN2#" and should be connected to the data transceiver enable input for bank 2 of the SRAM memory subsystem. This pin is disabled when SUS_STAT# is active (LOW) and the system is not performing a suspend refresh operation. When the pin is disabled the output is sustained at the previous state by internal "keepers".
CMUX4	**CPU Multiplexed Pin Four:** This output signal has two functions. When the 386 SL CPU Memory Controller Unit is configured as a DRAM controller this pin becomes "CASL1#" and should be connected to the lower byte of DRAM bank 1 CAS# input. When the 386 SL CPU Memory Controller Unit is configured as a SRAM controller this pin becomes "DEN1#" and should be connected to the data transceiver enable input for bank 1 of the SRAM memory subsystem. This pin is disabled when SUS_STAT# is active (LOW) and the system is not performing a suspend refresh operation. When the pin is disabled the output is sustained at the previous state by internal "keepers".
CMUX5	**CPU Multiplexed Pin Five:** This output signal has two functions. When the 386 SL CPU Memory Controller Unit is configured as a DRAM controller this pin becomes "CASH1#" and should be connected to the upper byte of DRAM bank 1 CAS# input. When the 386 SL CPU Memory Controller Unit is configured as a SRAM controller this pin becomes "DEN1#" and should be connected to the data transceiver enable input for bank 1 of the SRAM memory subsystem. This pin is disabled when SUS_STAT# is active (LOW) and the system is not performing a suspend refresh operation. When the pin is disabled the output is sustained at the previous state by internal "keepers".

386™ SL Microprocessor Signal Descriptions (Continued)

Symbol	Name and Function
CMUX6	**CPU Multiplexed Pin Six:** This output signal has two functions. When the 386 SL CPU Memory Controller Unit is configured as a DRAM controller this pin becomes "CASL0#" and should be connected to the lower byte of DRAM bank 0 CAS# input. When the 386 SL CPU Memory Controller Unit is configured as a SRAM controller this pin becomes "DEN0#" and should be connected to the data transceiver enable input for bank 0 of the SRAM memory subsystem. This pin is disabled when SUS_STAT# is active (LOW) and the system is not performing a suspend refresh operation. When the pin is disabled the output is sustained at the previous state by internal "keepers".
CMUX7	**CPU Multiplexed Pin Seven:** This output signal has two functions. When the 386 SL CPU Memory Controller Unit is configured as a DRAM controller this pin becomes "CASH0#" and should be connected to the upper byte of DRAM bank 0 CAS# input. When the 386 SL CPU Memory Controller Unit is configured as a SRAM controller this pin becomes "DEN0#" and should be connected to the data transceiver enable input for bank 0 of the SRAM memory subsystem. This pin is disabled when SUS_STAT# is active (LOW) and the system is not performing a suspend refresh operation. When the pin is disabled the output is sustained at the previous state by internal "keepers".
CMUX8	**CPU Multiplexed Pin Eight:** This output signal has two functions. When the 386 SL CPU Memory Controller Unit is configured as a DRAM controller this pin becomes "RAS3#" and should be connected to the upper and lower byte of DRAM bank 3 RAS# inputs. When the 386 SL CPU Memory Controller Unit is configured as a SRAM controller then this pin becomes "CE3#" and should be connected to the upper and lower byte of the SRAM chip-select, or to the chip-select decode logic for bank 3 of the SRAM memory subsystem. This pin is disabled when SUS_STAT# is active (LOW) and the system is not performing a suspend refresh operation. When the pin is disabled the output is sustained at the previous state by internal "keepers".
CMUX9	**CPU Multiplexed Pin Nine:** This output signal has two functions. When the 386 SL CPU Memory Controller Unit is configured as a DRAM controller this pin becomes "RAS2#" and should be connected to the upper and lower byte of DRAM bank 2 RAS# inputs. When the 386 SL CPU Memory Controller Unit is configured as a SRAM controller this pin becomes "CE2#" and should be connected to the upper and lower byte of the SRAM chip-select, or to the chip-select decode logic for bank 2 of the SRAM memory subsystem. This pin is disabled when SUS_STAT# is active (LOW) and the system is not performing a suspend refresh operation. When the pin is disabled the output is sustained at the previous state by internal "keepers".
CMUX10	**CPU Multiplexed Pin Ten:** This output signal has two functions. When the 386 SL CPU Memory Controller Unit is configured as a DRAM controller this pin becomes "RAS1#" and should be connected to the upper and lower byte of DRAM bank 1 RAS# inputs. When the 386 SL CPU Memory Controller Unit is configured as a SRAM controller this pin becomes "CE1#" and should be connected to the upper and lower byte of the SRAM chip-select, or to the chip-select decode logic for bank 1 of the SRAM memory subsystem. This pin is disabled when SUS_STAT# is active (LOW) and the system is not performing a suspend refresh operation. When the pin is disabled the output is sustained at the previous state by internal "keepers".

386™ SL Microprocessor Signal Descriptions (Continued)

Symbol	Name and Function
CMUX11	**CPU Multiplexed Pin Eleven:** This output signal has two functions. When the 386 SL CPU Memory Controller Unit is configured as a DRAM controller this pin becomes "RAS0#" and should be connected to the upper and lower byte of DRAM bank 0 RAS# inputs. When the 386 SL CPU Memory Controller Unit is configured as a SRAM controller this pin becomes "CE0#" and should be connected to the upper and lower byte of the SRAM chip-select, or to the chip-select decode logic for bank 0 of the SRAM memory subsystem. This pin is disabled when SUS_STAT# is active (LOW) and the system is not performing a suspend refresh operation. When the pin is disabled the output is sustained at the previous state by internal "keepers".
CMUX12	**CPU Multiplexed Pin Twelve:** This output signal has two functions. When the 386 SL CPU Memory Controller Unit is configured as a DRAM controller this pin becomes "PARL" and should be connected to the lower byte of DRAM bank 0 data parity bit. When the 386 SL CPU Memory Controller Unit is configured as a SRAM controller this pin becomes "OLE#" and should be connected to the lower byte of the SRAM output enable input of the SRAM memory subsystem. This pin is disabled when SUS_STAT# is active (LOW) and the system is not performing a suspend refresh operation. When the pin is disabled the output is sustained at the previous state by internal "keepers".
CMUX13	**CPU Multiplexed Pin Thirteen:** This output signal has two functions. When the 386 SL CPU Memory Controller Unit is configured as a DRAM controller this pin becomes "PARH" and should be connected to the upper byte of DRAM bank 0 data parity bit. When the 386 SL CPU Memory Controller Unit is configured as a SRAM controller this pin becomes "OHE#" and should be connected to the upper byte of the SRAM output enable input of the SRAM memory subsystem. This pin is disabled when SUS_STAT# is active (LOW) and the system is not performing a suspend refresh operation. When the pin is disabled the output is sustained at the previous state by internal "keepers".
CMUX14	**CPU Multiplexed Pin 14:** This output signal has two functions. The 386 SL CPU can be configured to use this pin as either a BIOS ROM chip-select (ROMCS1#), or a FLASH disk chip-select signal (FLSHDCS#). In either case, the signal is driven LOW when an access to the selected interface occurs.
COE#	**Cache Output Enable:** This active LOW output signal is used to indicate a read access to the CACHE SRAMs, and is used to enable the cache SRAMs' output buffers. This signal should be connected to the output enable signals of the upper and lower byte cache SRAMs.
CPURESET	**CPU Reset:** This active HIGH input forces the 386 SL CPU to execute a reset to the internal CPU core and state machines. The configuration registers are not reset.
CWE#	**Cache Write Enable:** This active LOW output is used to indicate a read (HIGH) or write (LOW) access to the cache SRAMs. This signal should be connected to the write enable signal of the upper and lower cache SRAMs.
DMA8/16#	**DMA 8-bit or 16-bit Cycle:** This input, in conjunction with HRQ, indicates to 386 SL CPU if an 8-bit or 16-bit DMA access is occurring. If an 8-bit DMA access is occurring, the 386 SL CPU will swap the upper byte of data to the lower data byte for upper byte accesses.
EFI	**External Frequency Input.** This is an oscillator input. This clock controls all CPU core and memory controller timings and is equal to twice the desired processor frequency (CLK2 vs CPUCLK).

386™ SL Microprocessor Signal Descriptions (Continued)

Symbol	Name and Function
ERROR#	**Numerics ERROR:** This active LOW input to the 386 SL CPU is generated from a math co-processor (MCP). It also indicates to the 82360SL that an unmasked exception has occurred in the MCP. ERROR# is provided to allow numerics error handling compatible with the ISA bus compatible Personal Computer.
HALT#	**HALT:** This active LOW output indicates to external devices that the 386 SL CPU has executed a HALT instruction (address = 2) or a shutdown condition (address = 0). This can be used as an indicator for devices to assert the STPCLK# signal.
HLDA	**HoLD Acknowledge:** This active HIGH output indicates to external devices that the 386 SL CPU has relinquished control of the ISA bus. At this time the 386 SL CPU has floated the address and control signals of the ISA bus.
HRQ	**Hold ReQuest:** This active HIGH input indicates to the 386 SL CPU that an external device wishes to take control of the ISA bus.
INTA#	**INTerrupt Acknowledge:** This active LOW output indicates that the 386 SL CPU is executing an interrupt acknowledge bus cycle. During this process an external interrupt device will pass an interrupt vector to the 386 SL CPU.
INTR	**Interrupt Request:** This active HIGH input indicates to the 386 SL CPU that an external device is requesting the execution of an interrupt service routine.
IOCHRDY	**I/O CHannel ReaDY:** This active HIGH input indicates that the I/O Channel, (ISA expansion bus), is ready to terminate the bus cycle. The ISA expansion bus is a normally ready bus and IOCHRDY is active HIGH. When an ISA bus peripheral needs to extend the standard 3 SYSCLK, 16-bit ISA bus cycle the peripheral device asserts IOCHRDY LOW.
IOCS16#	**I/O Chip Select 16:** This active LOW input indicates that an ISA bus peripheral wishes to execute a 16-bit I/O cycle. This signal has an active pull-up, when not driven the default I/O bus cycle is 8 bits.
IOR#	**I/O Read:** This active LOW signal indicates that the ISA bus is executing an I/O read cycle.
IOW#	**I/O Write:** This active LOW signal indicates that the ISA bus is executing an I/O write cycle.
ISACLK2	**ISA Clock Two:** This is an oscillator input. This clock controls all of the ISA bus timings and is equal to twice the SYSCLK frequency. Normally the ISA bus SYSCLK is 8 MHz, and the ISACLK2 oscillator is 16 MHz.
LA[23:17]	**Latchable local Address bus:** This is the unlatched local address of the ISA bus for access to memory above 1 megabyte. The LA bus is also used by the Peripherial Interface (PI) Bus.
MA[10:0]	**Memory controller Multiplexed Address bus:** This is the address bus output for the Memory Controller Unit. The 22-bit address is output in a row/column fashion for both DRAM and SRAM memory subsystems. The Memory Controller Unit places the ROW address out first and qualifies it by the RASx# signal going active in DRAM mode or the LE signal going active in the SRAM mode. The column address is then placed on the Memory Address bus and is qualified by the CASXx# signals going active for the DRAM mode. This pin is disabled when SUS_STAT# is active (LOW). When the pin is disabled the output is sustained at the previous state by internal "keepers".

386™ SL Microprocessor Signal Descriptions (Continued)

Symbol	Name and Function
MASTER#	**Master:** This active LOW input indicates that an ISA bus peripheral is controlling the bus. The peripheral device asserts this signal in conjunction with a DMA request (DRQ) line or the HRQ (hold request) to gain control of the bus. When the MASTER# signal is asserted LOW along with HRQ being asserted HIGH or a DRQ line being asserted HIGH, the 386 SL CPU will float all address, data and control signals on the ISA bus.
MD[15:0]	**Memory controller local Memory Data bus:** This is the bi-directional data bus of the Memory Controller Unit. All accesses by the Memory Controller Unit that transfer data between the 386 SL CPU and SRAM or DRAM use the Memory Data Bus. This pin is disabled when SUS_STAT# is active (low) and the system is not performing a suspend refresh operation. When the pin is disabled the output is sustained at the previous state by internal "keepers".
MEMCS16#	**MEMory Chip Select 16:** This active LOW input indicates that an ISA bus peripheral wishes to execute a 16-bit memory cycle. This signal has an active pull-up, when not driven the default memory bus cycle is 8 bits.
MEMR#	**MEMory Read:** This bi-directional active LOW signal indicates when a memory read access is taking place on the ISA bus. When the 386 SL CPU is performing a memory read to the ISA bus it is an output, when the DMA or Bus Master is accessing memory on the ISA bus, the DMA device or Master drives MEMR#.
MEMW#	**MEMory Write:** This bi-directional active LOW signal indicates when a memory write access is taking place on the ISA bus. When the 386 SL CPU is performing a memory write to the ISA bus it is an output, when the DMA or Bus Master is accessing memory on the ISA bus, the DMA controller or Bus Master drives MEMW#.
N/C	**No connection:** These pins must not be connected to any voltage, but must be left floating in order to guarantee proper operation of the 386 SL CPU and to maintain compatibility with future Intel Processors.
NMI	**Non-Maskable Interrupt:** This rising edge sensitive input will latch a request to the 386 SL CPU for a non-maskable interrupt on a LOW-to-HIGH transition.
NPXADS#	**Numerics ADdress Strobe:** This active LOW output signal indicates the start of a math co-process (MCP or NPX, numerics processor extension) data transfer cycle.
NPXCLK	**Numerics Clock:** This output signal is used to drive the MCP clock input.
NPXRDY#	**Numerics Ready:** This active LOW input is used to terminate a MCP (or NPX, numerics processor extension) bus cycle. This signal is low for I/O and data operand MCP cycles.
NPXRESET	**Numerics Reset:** This active HIGH output signal is used to reset the MCP.
NPXW/R#	**Numerics Write or Read:** This output signal indicates the type of data transfer that is being performed between the 386 SL CPU and the MCP. When high this signal indicates a MCP write, when low this signal indicates a MCP read.
ONCE#	**ON-board Circuit Emulation:** This active LOW input signal floats the neccessary outputs from the 386 SL CPU allowing an in-circuit emulation (ICE™-386™ SL) module to drive the 386 SL CPU signals. This allows an emulator to be used for system testing and developement while the 386 SL CPU and the 82360SL are still physically populated on the system motherboard. The state of all 386 SL CPU and 82360SL signals when ONCE# is asserted low is summarized in section 2, (386 SL CPU and 82360SL signal characteristics).

386™ SL Microprocessor Signal Descriptions (Continued)

Symbol	Name and Function
PCMD#	**PI-BUS Command:** This active LOW output indicates that valid write data is on the System data bus (SD[15:0]) signals, or that the 386 SL CPU is ready to sample valid read data from the PI bus for Peripherial Interface bus cycles.
PEREQ	**Processor Extension Request:** This active HIGH output signal indicates that the 386 SL CPU has data to transfer to or from the MCP data FIFO.
PERR#	**Parity ERRor:** This active LOW output indicates to an external device that the 386 SL CPU Memory Controller Unit has detected a memory parity error. The PERROR# signal is used by the 82360SL to generate NMI back to the 386 SL CPU.
PM/IO#	**PI-BUS Memory or I/O:** This output indicates the type of bus cycle the 386 SL CPU is executing on the Peripherial Interface Bus (PI-bus): Either a Memory (HIGH) or I/O (LOW) cycle.
PRDY#	**PI-BUS Ready:** This active LOW input is used to terminate Peripherial Interface bus cycles. The Peripheral Interface Bus is a normally not-ready bus, and will continue the bus cycle until the PRDY# is activated or a Peripherial Interface Time-out occurs.
PSTART#	**PI-BUS START:** This active LOW output indicates that the address (SA[19:0], LA[23:17] and SBHE#), command signals (PM/IO# and PW/R#) and chip-selects (VGACS# or FLSHDCS#) are valid for a Peripheral Interface Bus cycle.
PW/R#	**PI-BUS Write or Read:** This output indicates the type of bus cycle the 386 SL CPU is executing on the Peripheral Interface Bus: Either a Write (HIGH) or Read (LOW) cycle.
PWRGOOD	**Power Good:** This active HIGH input indicates that power to the system is good. This signal is generated by the power supply circuitry, and a LOW level on this signal causes the 386 SL to totally reset: The CPU core is reset, internal state machines are reset, all configuration registers are reset. Power Good should be low for a specified minimum number of CPU clocks for valid recognition in order to perform a global 386 SL CPU reset.
REFREQ	**REFresh REQuest:** This active HIGH input indicates that the 386 SL CPU should execute an internal DRAM refresh cycle to the on-board local memory.
ROM16/8#	**ROM 16-bits or 8-bits:** This input configuration signal pin selects if the BIOS interface is a 16-bit (when high) or 8-bit interface (when low). This pin has an internal pull-up resistor defaulting to a 16-bit wide BIOS EPROM.
ROMCS0#	**ROM Chip Select 0:** This LOW true output provides the chip select for the System BIOS EPROM.
SA[19:0]	**System Address Bus:** This is the bi-directional system address of the ISA bus, as well as the Peripheral Interface Bus. SA[16:0] are inputs during DMA and Master operation. SA[19:17] are outputs only since a 8237 compatible DMA controller accesses up to 64 kBytes at a time. The 74LS612 module in the 82360SL is used to furnish the DMA upper addresses for DMA access to 16 Megabyte.
SBHE#	**System Byte High Enable:** When this output signal is LOW, it indicates that data is being transferred on the upper byte of the 16-bit data bus (SD[15:8]).
SD[15:0]	**System Data Bus:** This 16-bit bi-directional data bus is used to transfer data between the 386 SL CPU and the ISA bus. The system data bus is also used to transfer data between the 386 SL CPU and the Peripherial Interface (PI-BUS).
SMI#	**System power Management Interrupt:** This falling edge sensitive input latches a Power Management interrupt request with a High-to-Low edge. The SMI# is the highest priority interrupt in the 386 SL processor.

386™ SL Microprocessor Signal Descriptions (Continued)

Symbol	Name and Function
SMRAMCS#	**System power Management RAM Chip Select:** This active LOW output is used to select an external system power management SM-RAM, and to indicate to the 82360SL device when accesses to the system power management SM-RAM are occurring.
STPCLK#	**Stop Clock:** This active LOW input stops the clock to the internal 386 CPU core. (This signal is functionally tested by the execution of HALT or I/O read instructions.)
SYSCLK	**System Clock:** This is a clock output equal to one half of the ISACLK2 input frequency.
SUS_STAT#	**SUSpend STATus:** This active LOW input indicates to the 386 SL CPU that system power is being turned off. The 386 SL CPU will respond by electrically isolating selected pins as indicated in Section 2, (386 SL CPU signal characteristics).
TURBO	**Turbo:** This active HIGH input signal indicates to 386 SL CPU when to enter "Turbo Mode". Turbo Mode is defined as the CPU executing at full speed, the default speed for the system. When this signal is forced inactive LOW, the 386 SL CPU executes from a divide by two or a divide by four clock as defined by the De-turbo bit in the CPUPWRMODE register. When this signal is HIGH, the CPU executes from a clock as defined by the Fast CPU clock field in the CPUPWRMODE register.
V_{CC}	**System Power:** Provides the +5V nominal D.C. supply inputs.
VGACS#	**VGA Chip-select:** This active LOW output is asserted anytime an access occurs to the user defined VGA address space.
V_{SS}	**System Ground:** Provides the 0V connection from which all inputs and outputs are referenced.
WHE#	**Write High Enable:** This active LOW output indicates that a write access to the upper byte of the 386 SL CPU memory bus is occurring when the Memory Controller Unit is configured for SRAM mode. When in DRAM mode, the signal is active anytime a write access occurs. This output should be connected to the write enable of the upper byte for either DRAM or SRAM memory subsystems. This pin is driven during a suspend operation.
WLE#	**Write Low Enable:** This active LOW output indicates that a write access to the lower byte of the 386 SL CPU memory bus is occurring when the Memory Controller Unit is configured for SRAM mode. When in DRAM mode, the signal is active anytime a write access occurs. This output should be connected to the write enable of the lower byte for either DRAM or SRAM memory subsystems. This pin is driven during a suspend operation.
ZEROWS#	**ZERO Wait State (ISA bus signal):** This active LOW input indicates that an ISA bus peripheral wishes to execute a zero wait state bus cycle (the normal default 16-bit ISA bus memory or I/O cycle is 3 SYSCLKs or one PC/AT equivalent wait state). When ZEROWS# is driven low, a 16-bit bus cycle will occur in two SYSCLKs. When ZEROWS# is driven low for an 8-bit memory cycle the default 6 SYSCLK bus cycle is shortened to 3 SYSCLKs.

3.0 SIGNAL DESCRIPTIONS (Continued)

82360SL ISA Peripheral I/O

The following table provides a brief description of the signals of the 82360SL I/O. Signal names which end with the character "#" indicate that the corresponding signal is low true when active.

Symbol	Name and Function
A20GATE	**A20 Gate (direct to CPU):** This active HIGH output signal forces the 386 SL CPU to mask off A20 on the system address bus (internal to the 386 SL CPU), to allow emulation of an 8086.
AEN	**Address ENabled (ISA-bus signal):** This active HIGH output indicates a DMA access, refresh or I/O access to a non-standard ISA peripheral I/O address location. The 82360SL drives this signal high to signify a valid DMA address. It is used by bus slaves to decode I/O ports. All ports must be decoded for AEN low. There are no DMA cycles to addressed I/O ports.
BALE	**Buffered Address Latch Enable (ISA-bus signal):** This active HIGH input to the 82360SL is driven by the 386 SL CPU during standard ISA bus cycles. During ISA bus memory and I/O cycles BALE is used to indicate valid addresses at the start of a bus cycle. SA[19:0] are valid on the falling edge and LA[23:17] are valid while BALE is high. BALE is is also driven high by the 386 SL CPU and remains high during DMA cycles.
BATTDEAD#	**BATTery DEAD:** This active LOW input indicates that the battery does not have enough power to resume or reset. This signal will prevent a system reset if asserted LOW.
BATTLOW#	**BATTery LOW:** This active LOW input indicates that the battery power is low. BATTLOW# is typically driven by a D.C. to D.C. power converter associated with the battery power supply. A thermal power monitor indicates that the main battery power is dropping below the adequate charge level to sustain operation. If this signal is asserted LOW with BATTWRN# asserted LOW a SMI request will be generated. The feature is enabled via S/W control. The signal will also prevent a resume operation if asserted LOW.
BATTWARN#	**BATTery WARNing:** This active LOW input indicates the battery has minimal charge left (eg. one half an hour of full power use remaining).
C8042CS#	**Keyboard controller Chip Select:** This active LOW output is driven when there is an I/O read or write to the Keyboard Controller Ports 60 or 64 hex.
COM(A,B)CTS#	**Clear To Send:** This active LOW input indicates to the Serial Port Controller for COMA or COMB that a serial device is clear to accept data. This signal is typically used for a modem control function. A change in the state of this signal generates a modem status interrupt. The modem or data set asserts this signal when it is ready to accept data for transmission.
COM(A,B)DCD#	**Data Carrier Detect:** This active HIGH input indicates that the Serial Port Controller COMA or COMB has detected a data carrier from the data set of a serial device. Typically this signal is from a modem.
COM(A,B)DSR#	**Data Set Ready:** This active LOW input signal is used by the modem or data set to indicate that the modem or data set is ready to establish the communication link and transfer data with the Serial Port Controller.
COM(A,B)DTR#	**Data Terminal Ready:** This active LOW output signal informs the modem or data set that the Serial Port Controller is ready to communicate.
COM(A,B)RXD	**Serial data Receive:** This input signal is used to receive serial data. Each character can consist of from five to eight bits of data with one start bit and one, one and a half or two stop bits. The least significant bit is received first.

82360SL ISA Peripheral I/O Signal Descriptions (Continued)

Symbol	Name and Function
COM(A,B)RI#	**Ring Indicator:** This active LOW input signal is used for a modem control function. A change in the state (either from high to low or from low to high) of this signal generates a modem status interrupt. The modem or data set asserts this signal to indicate that it has detected a telephone ring. This will cause the 82360SL to wake the 386 SL CPU from a suspended state if modem ring is enabled as a wake-up event.
COM(A,B)RTS#	**Request To Send:** This active LOW output signal informs the modem or data set that the Serial Port Controller is ready to send data.
COM(A,B)TXD	**Serial data transmission:** This output signal is used to transmit data serially between the Serial Port Controller and serial device. Each character can consist of five to eight bits of data with one start bit and either one, one and a half, or two stop bits. The least significant bit is transmitted first. The control of the format of a character is defined under S/W control via the Line Control Register. Please consult the 386 SL Microprocessor SuperSet Programmer's Reference Manual for additional information. Information regarding the functional timing specifications of transmitted and recieved serial data may be found in sections 6 and 7 (A.C. timing specifications and timing diagrams).
COMX1,COMX2	**Crystal oscillator input and output pins:** The crystal attached to these signals should be tuned to 1.8432 Mhz. The on-chip oscillator uses an external crystal and tank circuit to generate an internal clock. This clock is used to generate the various baud rates for the serial ports. Optionally an external oscillator may be connected to the COMX1 input.
CPURESET	**CPU RESET:** This active HIGH output is connected directly to the 386 SL CPU to provide a reset of the 386 CPU core. CPURESET always occurs during a PWRGOOD reset. CPURESET may also be generated by RC# from a keyboard controller, Fast Reset from I/O Port 92 or other programmable Reset, or a resume from suspend.
CX1,CX2	**Crystal oscillator input and output pins:** The crystal should be tuned to 14.31818 Mhz. It is used for the ISA bus signal OSC signal and is internally divided by 12 to clock the timer counters. The oscillator input may be directly driven from an external source.
DACK[7:5], [3:0]#	**DMA ACKnowledge channel n (ISA bus signal):** The 82360SL DMA controller drives the respective DMA acknowledge signal low after a device has requested DMA service. The corresponding output signal indicates that the DMA channel transfer may begin.
DMA8/16#	**DMA 8-bit or 16-bit cycle:** This output signal is directly connected to the 386 SL CPU. When the signal is HIGH it indicates that the current DMA cycle is 8-bit. When this signal is low it indicates that the DMA cycle is using a 16-bit channel.
DRQ[7:5], [3:0]	**DMA ReQuest channel n (ISA bus signal):** These input signals are used to request DMA service from devices residing on the ISA bus. An ISA bus device drives this signal to request service from the appropriate DMA channel by asserting this signal high.
ERROR#	**MCP ERROR:** This signal is an active LOW input to the 82360SL. The math coprocessor error signal generates a IRQ13 through the 82360SL.
EXTSMI#	**EXTernal System Management Interrupt request:** This active low input will generate a SMI request if the function is enabled.
EXTRTCAS	**EXTernal RTC Address Strobe:** This output signal is active HIGH when there is a write access to the RTC I/O address port and when an external RTC is selected.
EXTRTCDS	**EXTernal RTC read Data Strobe:** This output signal is active LOW when there is a read access to an external RTC I/O data port and when an external RTC is selected.
EXTRTCRW#	**EXTernal RTC (Real Time Clock) Read/Write:** This low true output signal is active when there is a write access to an external RTC I/O data port and when an external RTC is selected.

82360SL ISA Peripheral I/O Signal Descriptions (Continued)

Symbol	Name and Function
FLPCS#	**FLoPpy Chip Select:** This LOW true output signal is the chip select for the floppy disk controller I/O ports 03F0–03F5 and 3F7 hex.
HALT#	**HALT:** This LOW true input signal is driven by the 386 SL CPU and indicates when the CPU has executed a HLT instruction (address = 2) or is in a shutdown condition (address = 0).
HD7	**HD-bus Data bit HD7:** The bi-directional System Data Bit 7 is controlled separately for the Integrated Drive Electronics (I.D.E.) hard disk drive and floppy disk drive. This is provided to accommodate the I/O address 3F7 hex which is split between the floppy disk drive controller and I.D.E. hard disk. Data transfer between storage peripherals connected to the I.D.E. Hard Disk and Floppy Disk and the 82360SL are on separate busses. Data bit 7 has to be separated from data bits [6:0]. The 82360SL controls and buffers data bit 7 seperately.
HDCS[1:0]#	**Hard Disk Chip Select:** These LOW true output signals are the I.D.E. hard disk drive chip selects decoded from the I/O address ports 01F0–01F7h (HDCS0#) and 03F6–03F7h (HDCS1#).
HDEN(H,L)#	**Hard Disk buffer ENable:** These LOW true output signals control the I.D.E. hard disk data buffers, high and low bytes.
HLDA	**HoLD Acknowledge (direct to CPU):** This HIGH true input signal indicates that the 386 SL CPU has released the ISA bus for refresh, DMA or master cycles.
HRQ	**Hold ReQuest (direct to CPU):** This active HIGH output signal indicates a request to the 386 SL CPU to release the ISA bus when the 82360SL requests the bus for ISA bus style refresh, DMA or master mode cycles.
IMUX0	This pin is multiplexed. It can be used as Timer 2 gate 2 input or a speaker input from the modem.
INTA#	**INTerrupt Acknowledge (direct to CPU):** This active LOW input to the 82360SL indicates that the 386 SL CPU has recognized an interrupt and will initiate an interrupt acknowledge bus cycle. The INTA bus cycle is comprised of two eight-bit I/O cycles in which the interrupt vector transferred on the second eight-bit I/O write of the INTA cycle.
INTR	**INTerrupt Request (direct to CPU):** This active HIGH output requests a standard maskable interrupt to the 386 SL CPU.
IOCHCK#	**IO CHannel CheckK (ISA bus signal):** This maskable active LOW input is driven by a device on the ISA bus typically used to indicate a parity error on the ISA bus. This signal is one of the possible sources which may generate an NMI. NMI generation via IO Channel Check may be enabled or disabled using PORT 61 (IOCKEN). NMI may be masked using the ISA bus compatible NMI control port at I/O 70 hex bit 7.
IOCHRDY	**I/O CHannel ReaDY (ISA bus signal):** This active HIGH input is used by the 82360SL DMA controller to extend ISA bus cycles. IOCHRDY is also used to extend bus cycles for I/O device trapping. Additional wait states extend the bus cycle, allowing for start up during Resume mode. The ISA bus is a normally ready bus, an external device can extend a DMA cycle or ISA bus cycle by deasserting this signal (driven low). This signal is normally high on the ISA bus.

82360SL ISA Peripheral I/O Signal Descriptions (Continued)

Symbol	Name and Function
IOCS16#	**16-bit I/O Chip Select (ISA bus signal):** This active LOW input signal to the 82360SL is used to indicate a 16-bit I/O bus cycle. The I.D.E. hard disk high byte buffer enable is generated when IOCS16# is driven low during an I.D.E. 16-bit I/O access. IOCS16# is also an input to the 386 SL CPU driven by devices residing on the ISA bus to indicate a 16-bit I/O bus cycle.
IOR#	**I/O Read (ISA bus signal):** This bi-directional active LOW signal is an input during normal accesses to I/O ports. When low this signal indicates an I/O read. This signal is an output from the 82360SL during DMA bus cycles for I/O to memory transfers.
IOW#	**I/O Write (ISA bus signal):** This bi-directional active LOW signal is an input during normal accesses to I/O ports. When low this signal indicates and I/O write. This signal is an output from the 82360SL during DMA bus cycles for memory to I/O transfers.
IRQ[15, 14, 12–3, 1]	**Interrupt ReQuest n (ISA bus signal):** These active HIGH input signals are used to request interrupt service. The interrupt request lines are driven by devices on the ISA bus which have a corresponding interrupt service routine associated with the interrupt vector and interrupt request.
KBDA20	**KeyBoarD A20 gate:** This active HIGH input is "ORed" with internal bits to produce A20GATE which goes to the 386 SL CPU. The bit is connected to port 2, bit 1 of an 8042 in a standard ISA bus compatible system.
KBDCLK	**KeyBoarD CLocK:** This output signal is used to drive the clock input to the keyboard controller. It is derived from the 8 MHz SYSCLK and can be divided by 1, 2, 4 or stopped.
LA[23:17]	**Local Address bus (ISA bus signal):** These are input signals to the 82360SL during memory transfers (decoding for X-bus buffer controls) and output signals during DMA accesses and refresh. The latchable address lines allow access to physical memory on the ISA bus to 16 megabytes.
LPTACK#	**Line PrinTer ACKnowledge:** Active LOW input signal which is part of the parallel port data handshake. The line printer asserts this signal to show that data transfer was complete and that it is ready for the next transfer.
LPTAFD#	**Line Printer Auto line FeeD:** This signal is an active LOW output from 82360SL to a printer. When asserted, it instructs the printing device to insert a line feed at the end of every line.
LPTBUSY	**Line PrinTer BUSY:** This signal is an active HIGH input to 82360SL. The printer asserts this signal when it is not ready to accept further data from 82360SL.
LPTD[7:0]	**Line printer Data bus:** These signals are the 8-bit bi-directional data bus for the parallel port. In PC/AT mode these signals are output only. The 82360SL also supports a bidirectional mode for the PS/2 style parallel port.
LPTDIR	**Line PrinTer DIRection:** This active HIGH output signal is only valid in bidirectional mode for data transfer using the parallel port.
LPTERROR#	**Line PrinTer ERROR:** This active LOW input signal is driven by a peripheral device to flag an error condition.
LPTINIT#	**Line PrinTer InITialize:** This active LOW output from 82360SL instructs the peripheral to initialize itself.
LPTPE	**Line PrinTer Paper End:** This active HIGH input to 82360SL signals that the printer has run out of paper when asserted.
LPTSLCT	**Line PrinTer SeLeCTed:** This active HIGH input signal is asserted by the printer to confirm that it has been selected.

82360SL ISA Peripheral I/O Signal Descriptions (Continued)

Symbol	Name and Function
LPTSLCTIN#	**Line PrinTer SeLeCT IN:** This active LOW output signal is asserted to select the printer interfaced to the parallel port.
LPTSTROBE#	**Line PrinTer STROBE:** This active LOW output signal is used to strobe data into the peripheral device. The parallel port controls are read and written through I/O registers.
MASTER#	**ISA bus MASTER (ISA bus signal):** This active LOW input signal is used with a DRQ line to gain control of the system bus. Upon receiving DACK# the 82360SL may pull MASTER# active (low), which will allow the 82360SL control of the system address, data and control busses. The 386 SL CPU will have tri-stated these lines one clock after receiving the MASTER# signal.
MEMR#	**MEMory cycle Read (ISA bus signal):** This bi-directional active LOW signal indicates a read cycle anywhere in the 16 Mbyte memory address space. During memory read cycles to memory on the ISA bus, this signal is an input into the 82360SL. MEMR# is driven by the 82360SL during DMA cycles.
MEMW#	**MEMory cycle Write (ISA bus signal):** This bi-directional active LOW signal indicates a write cycle anywhere in the 16 Mbyte memory address space. During memory write cycles to memory on the ISA bus, this signal is an input. MEMW# is an output from the 82360SL during DMA cycles.
N/C	**No Connection:** These signals must not be connected to any voltage. The No Connection signals must be left floating in order to guarantee proper operation of the 82360SL and compatibility with future Intel processors.
NMI	**Non Maskable Interrupt (direct to CPU):** This active HIGH output is directly connected to the 386 SL CPU. The 82360SL asserts NMI to request the 386 SL CPU to service a high priority non-maskable interrupt. The low to high transition of this signal is recognized by the 386 SL CPU.
ONCE#	**ON-board Circuit Emulation:** This active LOW input pin floats the appropriate outputs of the 82360SL as indicated in Section 2 pin assignments. When ONCE# is driven active the 82360SL allows an In-Circuit emulator (ICE™-386™ SL) module to drive its signals. This allows the system to be tested while the 82360SL is still physically populated on the motherboard.
OSC	**OSCillator (ISA bus signal):** This is the 14.31818 Mhz output signal with a 50% duty cycle and is asynchronous to SYSCLK.
PERR#	**Parity ERRor (direct from CPU):** This active LOW input signal is connected to the output of the 386 SL CPU. When the 386 SL CPU detects a parity error from the local DRAM subsystem it drives this signal to the 82360SL. The system memory parity error will generate a NMI via the 82360SL when NMI is enabled via I/O port 70 hex bit 7.
SMI#	**System Management Interrupt (direct to CPU):** This active LOW output is directly connected to the 386 SL CPU. When the falling edge of SMI# is detected by the 386 SL CPU it generates the highest priority interrupt when enabled. The typical use of SMI# is for power management.
SMRAMCS#	**System Management RAM Chip Select:** This active LOW output is driven whenever the 386 SL CPU is accessing the System Management SM-RAM. It is active even when SM-RAM is part of the 386 SL CPU system memory RAM. The 82360SL uses the SMRAMCS# to determine when the SMI code is being executed on the ISA bus, and enables the X-bus control signals.
PWRGOOD	**PoWeR GOOD:** This active HIGH input is typically supplied by the power supply. When Power good is activated high this indicates that the supply voltage is stable. Power Good low is also used to generate System Reset, RESETDRV, and CPURESET.

82360SL ISA Peripheral I/O Signal Descriptions (Continued)

Symbol	Name and Function
RC#	**Reset CPU:** This active low input is typically driven by the keyboard controller. RC# is "ORed" with internal bits to produce a programmable pulse width CPURESET signal. It is connected to port 2, bit 0 of an 8042 in a standard ISA bus compatible system.
REFREQ	**REFresh REQuest (direct to CPU):** This active HIGH output signal is directly connected to the 386 SL CPU. When Refresh Request is asserted it indicates that the 386 SL CPU should refresh the local DRAM subsystem.
REFRESH#	**System REFRESH (ISA bus signal):** This active LOW input signal indicates a refresh cycle. It is driven for the duration of the cycle. It is an input during master generated refresh bus cycles.
RESETDRV	**RESET DRiVe (ISA bus signal):** This active HIGH output is the main system cold reset, generated from the power supply "power good" signal and by system resume.
RTCEN#	**RTC ENable:** This active LOW input signal should be strapped high or low depending on whether an internal (LOW) or external (HIGH) RTC is used in the system. The 82360SL on-chip real time clock and CMOS RAM are enabled by this signal when LOW.
RTCRESET#	**Internal RTC RESET input:** This active LOW input signal is used to reset the internal RTC status and flag registers, (typically when the RTC battery has been changed).
RTCVCC	This is a separate power supply input for the internal RTC. It should be connected to a 3V battery when the system is fully off and 5V during active operation.
RTCX1,RTCX2	**RTC Crystal oscillator input and output pins:** The crystal should be tuned to 32.768 Khz. It is used for the RTC and system power management state machines. The oscillator may be driven directly from the input signal.
SA[16:0]	**System Address bus (ISA bus signal):** The bi-directional system address bus is an input for decoding internal I/O registers and an output during DMA and refresh cycles.
SBHE#	**System Byte High Enable (ISA bus signal):** The active LOW output signal indicates when there is valid data on the upper data byte of the system data bus.
SD[7:0]	**System Data bus (ISA bus signal):** This is the bidirectional system data bus. The 82360SL directly drives the ISA bus system data bits [7:0] without external transceivers or buffers. 8-bit data is transferred to and from the 82360SL with these signals.
SMEMR#	**System MEMory Read (ISA bus signal):** This signal is driven by the 82360SL to signify a memory read cycle to the bottom 1 Mbyte address range. It is used by ISA bus compatible slaves which decode SA[19:0] during memory cycles.
SMEMW#	**System MEMory Write (ISA bus signal):** This signal is driven by the 82360SL to signify memory write cycle to the bottom 1 Mbyte address range. It is used by ISA bus compatible slaves which decode SA[19:0] during memory cycles.
SMOUT[5:0]	**System Management OUTput control:** These six outputs can be connected to control the power circuits for various devices in the system. These output pins are directly controlled by the SM_OFF_CNTRL register.
SPKR	**SPeaKeR output:** This is the output of the 8254 megacell, timer/counter #1, channel 2, or directly driven through IMUX0, or from the 8254 megacell, timer counter #2, channel 1. This output signal is typically connected to an external speaker. There is additional circuitry to ensure that the signal is low when not being used.
SRBTN#	**Suspend/Resume BuTtoN:** This active LOW input generates a SMI requesting a system suspend or resume.

82360SL ISA Peripheral I/O Signal Descriptions (Continued)

Symbol	Name and Function
STPCLK#	**SToP CLocK:** This active LOW output signal stops the clock to the 386 CPU core of the 386 SL Microprocessor. Stop clock is directly connected to the 386 SL CPU from the 82360SL. The 82360SL activates this signal upon detection of a halt bus cycle or when an I/O read to the stop clock register in the 82360SL occurs.
SYSCLK	**SYStem CLocK (ISA bus signal):** This signal is an output from the 386 SL CPU and an input to the 82360SL. The SYSCLK signal is used to clock the ISA bus state machines and is also used to derive the internal DMA clock signal in the 82360SL. The SYSCLK is the 8 MHz typical clock which is one half of the frequency of ISACLK2.
SUS_STAT#	**SUSpend STATus:** The 82360SL power management controls this active low output signal to switch the power off to all non-critical devices during a suspend.
TC	**Terminal Count (ISA bus signal):** This active HIGH output signal is used to indicate the termination of a DMA transfer.
TIM2CLK2	**TIMer 2 CLK:** This is the input clock for timer/counter #2 when it is programmed to be used in the General Purpose (GP) mode.
TIM2OUT2	**TIMer 2 OUTput:** This signal is the frequency output from timer/counter #2 and can be used as a general purpose timer/counter output.
V_{CC}	**System Power:** Provides the +5V nominal D.C. supply inputs for the 82360SL.
V_{SS}	**System Ground:** Provides the 0V connection from which all inputs and outputs are referenced.
XD7	**X-bus Data bit XD7:** I/O port 3F7h is split between the floppy and hard disk and the storage peripherals which transfer data reside on separate busses. Data bit XD7 is separated from bits XD[6:0]. The 82360SL separately controls and buffers bit XD7 to isolate data bit 7 from the floppy disk and I.D.E. hard disk.
XDEN#	**X-bus Data ENable:** This active LOW output signal is used to control the X-bus data transceiver. It is only activated by the 82360SL on valid accesses to X-bus peripherals.
XDIR	**X-bus data DIRection:** This active HIGH output signal controls the direction of the X-bus and HD-bus data transceivers. XDIR is high for read cycles.
ZEROWS#	**ZERO Wait State (ISA-bus signal):** This active LOW output signal is driven by the 82360SL when it can accept a zero wait state write cycle.

4.0 PACKAGE THERMAL SPECIFICATIONS

The SL SuperSet is specified for functional operation with a temperature range from 0 to 90 degrees Celcius for the 386 SL CPU and the 82360SL. The case temperature should be measured in the operating environment to determine whether the SL SuperSet is within the specified operating temperature range. The case temperature should be measured at the center of the top surface of the package. When the SL SuperSet devices have a supply voltage applied the operating temperature range is applicable rather than the storage temperature.

The following definitions and assumptions are used to determine the recommended maximum case temperature for the 386 SL CPU and 82360SL:

T_A = Ambient Temperature in degrees Celcius

T_C = Case temperature in degrees Celcius

θ_{JC} = Package thermal resistance between junction and case

θ_{JA} = Package thermal resistance between junction and ambient

T_J = Junction Temperature

P = Power Consumption in Watts

The ambient temperature can be evaluated by using the values of thermal resistance between junction and case, θ_{JC} and the thermal resistance between junction and ambient θ_{JA} in the following equations:

$$T_J = T_C + P*\theta_{JC}$$

$$T_A = T_J - P*\theta_{JA}$$

$$T_C = T_A + P*[\theta_{JA} - \theta_{JC}]$$

Values for θ_{JA} and θ_{JC} are given in Table 4-1 for the 196-lead PQFP 82360SL and the 227-lead LGA 386™ SL CPU.

Table 4-1. Thermal Resistances (°C/W) θ_{JC} and θ_{JA}

Package	θ_{JC} °C/W	θ_{JA} (°C/W) versus Airflow—ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
196L PQFP	5	21	18	13.5	11.8	10.5	9.5
227L LGA							

5

ABSOLUTE MAXIMUM RATINGS

Table 4.3 provides environmental stress ratings for the packaged SL SuperSet devices. Functional operation at the storage maximum and minimum ratings is not implied or guaranteed.

Extended exposure to maximum ratings may affect device reliability. Further, precautions should be taken to avoid high static voltages and electric fields to prevent static electric discharge.

Other system components such as the memory subsystem (DRAM/SRAM), storage peripherals (hard disk/floppy disk), I/O and display subsystem may reduce the absolute maximum storage temperature conditions due to the inherent physical characteristics of the other components.

Table 4-3. Maximum Ratings

Parameter	Maximum Rating
1. Storage Temperature	−65°C to +150°C
2. Case Temperature under Bias	0°C to +90°C(1)
3. Supply Voltage with Respect to V_{SS}	−0.5V to +6.5V
4. Voltage on Other Pins	−0.5V to (V_{CC} + 0.5V)

NOTE:
1. Case temperature under Bias maximum rating also includes the case where the 386 SL CPU and 82360SL are in suspend or standby mode. In standby mode and in specific cases in suspend mode, power is applied to the SL SuperSet for operation of the Real-Time Clock and DRAM refresh. It is assumed in these cases that the SL SuperSet devices are not in normal or full-speed operation. Typically at these extreme minimum and maximum temperature ranges the external oscillators are stopped or diasbled with the exception of the 32 kHz Real-Time Clock oscillator. The limiting factor for minimum and maximum case temperature under Bias is the operational temperature range supported by the RTC crystal and 82360SL on-chip oscillator. It is also assumed that main system memory is not being accessed (only slow refresh for DRAM) or the SRAM is in standby mode, and all other components used in the system are also capable of operating at these maximum and minimum temperature values.

5.0 D.C. SPECIFICATIONS

386™ SL CPU D.C. Specifications

Functional operating range: V_{CC} = 5V ± 10%; T_{CASE} = 0°C to 90°C

Table 5-1. D.C. Voltage Specifications

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Inpt Low Voltage	−0.3	0.8	V	At 8 MHz
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	At 8 MHz
V_{ILC}	EFI/ISACLK2 Input Low Voltage	−0.3	0.8	V	At 8 MHz, CMOS Logic Levels
V_{IHC}	EFI/ISACLK2 Input High Voltage	V_{CC} − 0.8	V_{CC} − 0.3	V	At 8 MHz, CMOS Logic Levels
V_{OL}	Output Low Voltage				
	I_{OL} = 4 mA		0.5	V	At 8 MHz(1)
	I_{OL} = 24 mA		0.5	V	At 8 MHz(2)
V_{OH}	Output High Voltage				
	I_{OH} = −2 mA	2.4		V	At 8 MHz(1)
	I_{OH} = −0.2 mA	V_{CC} − 0.5		V	At 8 MHz(2)
	I_{OH} = −4 mA	2.4		V	At 8 MHz(2)
	I_{OH} = −0.18 mA	V_{CC} − 0.5		V	At 8 MHz(1)

ADVANCE INFORMATION SEE INTEL FOR DESIGN-IN INFORMATION

Table 5-2. Leakage Current and Sustaining Current Specifications

Symbol	Parameter	Min	Max	Unit	Notes
I_{IL}	Input Leakage Current Condition 1: When SUS_STAT# and/or ONCE# not active. Pins with internal 60k PU Pins with internal 20k PD Pins with internal 300 PU Other Input Pins		 −150 300 −24 ±15	 µA µA mA µA	 V_{IL} = 0.45V V_{IH} = 2.4V V_{IL} = 0.45V 0V < V_{IN} < V_{CC}
	Condition 2: When SUS_STAT# and/or ONCE# active. Pins with internal 60k PU Pins with internal 20k PD Pins with internal 300 PU Other Input Pins		 ±15 ±15 ±15 ±15	 µA µA µA µA	 0V < V_{IN} < V_{CC} 0V < V_{IN} < V_{CC} 0V < V_{IN} < V_{CC} 0V < V_{IN} < V_{CC}
I_{OL}	Output Leakage Current Condition 1: When SUS_STAT# and/or ONCE# not active Pins with internal 60k PU Pins with internal 300 PU Other Output Pins		 −150 −24 ±15	 µA mA µA	 V_{OUT} = 0.45V V_{OUT} = 0.45V 0.45V < V_{OUT} < V_{CC}
	Condition 2: When SUS_STAT# and/or ONCE# active Pins with internal 60k PU Pins with internal 300 PU Other Output Pins		 ±15 ±15 ±15	 µA µA µA	 0.45V < V_{OUT} < V_{CC} 0.45V < V_{OUT} < V_{CC} 0.45V < V_{OUT} < V_{CC}
I_{BHL}	Input Sustaining Current (Bus Hold Low)		38	µA	V_{IN} ≤ 0.8V(3,4)
I_{BHH}	Input Sustaining Current (Bus Hold High)		−60	µA	V_{IN} ≥ 3.0V(3,5)
I_{BHLO}	Bus Hold Low Overdrive		300	µA	(Notes 3, 6)
I_{BHHO}	Bus Hold High Overdrive		−550	µA	(Notes 3, 7)

ADVANCE INFORMATION
SEE INTEL FOR DESIGN-IN INFORMATION

Table 5-3. Capacitance D.C. Specifications

Symbol	Parameter	Min	Max	Unit	Notes
C_{IN}	Input Capacitance		10	pF	EFI = 1 MHz(8)
C_{OUT}	Output or I/O Capacitance		20	pF	EFI = 1 MHz(8)
C_{CLK}	EFI Capacitance		15	pF	EFI = 1 MHz(8)

NOTES:
1. List of pins which have 24 mA/4 mA I_{OL}/I_{OH} specification, (reference section 2).
2. Other output pins which do not belong to list in Note 1, (reference Section 2).
3. Tested with CPU Clock stopped.
4. This is the maximum current the bus hold circuit can sink without raising the node above 0.8V. I_{BHL} should be measured after lowering V_{IN} to Ground (0V) and then raising to 0.8V.
5. This is the maximum current the bus hold circuit can source without lowering the node voltage below 3.0V. I_{BHH} should be measured after raising V_{IN} to V_{CC} and then lowering to 3.0V.
6. An external driver must source at least I_{BHLO} to switch this node from low to high.
7. An external driver must sink at least I_{BHHO} to switch this node from high to low.
8. Not tested. Guaranteed by design characterization.

Table 5-4. 386™ SL CPU I_{CC} Specifications

Symbol	Parameter	Min	Max	Unit	Notes
I_{CC}	Supply Current				(Note 1)
	Minimum Configuration		400	mA	(Notes 2, 4)
	Maximum Configuration		750	mA	(Notes 3, 4)
I_{CC1}	Supply Current/Stop Clock		75	mA	(Notes 5, 6)
I_{CC2}	Supply Current/Suspend Mode/Oscillators Free Running/Suspend Refresh ON		10	mA	(Notes 5, 7)
I_{CC3}	Supply Current/Suspend Mode/Oscillators OFF Running/Suspend Refresh ON		6	mA	(Note 8)
I_{CC4}	Supply Current/Suspend Mode/Oscillators OFF Running/Suspend Refresh OFF		5	mA	(Note 9)

DRAFT INFORMATION NOT FOR DESIGN-IN INFORMATION

NOTES:
1. Tested at EFI and ISACLK2 at maximum frequency, with 50 pF load and no resistive loads on the outputs.
2. Minimum System Configuration consists of 1 bank of 1 Megabyte x 4 DRAMs (2 Megabyte total memory), cache disabled with no cache SRAM, 25 pF capacitive loading on the PI-bus control/status signals, 100 pF capacitive loading on the ISA-bus, 100 pF loading on the SYSCLK.
3. Maximum System Configuration consists of 4 banks of 4 Megabyte x 1 DRAMs (32 Megabytes total), cache enabled with 2 x (16k x 16) cache SRAMs, 65 pF capacitive loading on the PI-bus control/status signals, 300 pF capacitive loding (8 slots) on the ISA-bus and 300 pF capacitive loading on the SYSCLK signal.
4. Not tested, very conservative estimates provided from engineering analysis at worst case temperature and at 5.5V with the described system configuration for comparison only.
5. Characterized with V_{CC} = 5.5V, EFI = 40 MHz, ISACLK2 = 16 MHz
6. 412.5 mW with 386 SL CPU with Stop Clock, all external oscillators are free running, there are no active bus cycles on the Cache, Memory or ISA busses. Internal logic such as the Cache and Memory Controller are unaffected by stopped or slow clock and continue to consume the fixed power represented in I_{CC1}.
7. 55 mW with 386 SL CPU in suspend mode, all external oscillators are free running, there are no active bus cycles on the Cache, Memory or ISA busses except suspend refresh.
8. 33 mW with 386 SL CPU in suspend mode, all external oscillators are off (fixed Logic State), there are no active bus cycles on the Cache, Memory or ISA busses except suspend refresh.
9. 27.5 mW with 386 SL CPU in suspend mode, all external oscillators are off (fixed Logic State), there are no active bus cycles on the Cache, Memory or ISA busses including suspend refresh.

386™ SL CPU I_{CC} Specifications: Special Topics

DETERMINING I_{CC} WITH SLOW CLOCK CONTROL

The 386 SL CPU supports CPU clock division which reduces power consumption of the CPU core logic. The EFI clock input is similar to the CLK2 input found on the 386 CPU. However, the internal CPUCLK signal in the 386 SL CPU is not always one half of the frequency of the EFI (CLK2) input. An internal clock divider and synchronizer allows the CPU core clock to be slowed down and even stopped. However, additional internal logic such as the memory controller and cache controller continue to use half the EFI frequency. Therefore, when calculating the theoretical power consumption with CPU clock division it is important to recognize that a fixed constant (K) value of power is required by the 386 SL CPU.

The value K is constant only if the ISA bus loading is constant. Figure 5-1 shows the value of K for different values of ISA bus capacitance.

I_{CC}(divided clock) = [I_{CC}(normal clock) * n] + K

I_{CC}(normal clock) = The I_{CC} value calculated from the following section, excluding ISA bus power.

n = The fractional value that the clock is divided (e.g., divide by 2 = 0.5)

K = Is a constant in MilliAmps which determined by reading the value in Figure 5-1.

To determine the maximum current for the 386 SL CPU with CLK2 divider perform the following steps:

1. Multiply the I_{CC} of the normal minimum system configuration by the fractional value of the clock divider.
2. Sum the total capacitive load of all active ISA bus output signals from the 386 SL CPU to all devices.
3. From Figure 5-1 draw a line from the horizontal axis (capacitance) where it intersects the diagonal line.
4. From Figure 5-1 draw a perpendicular line to the vertical axis to determine K.
5. Solve the equation for I_{CC} (divided clock).

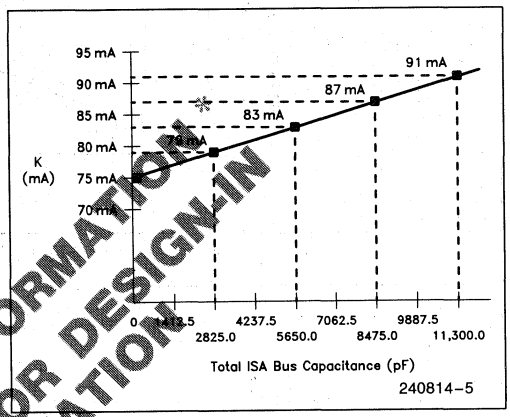


Figure 5-1. Variation of the constant current (K) with respect to the total ISA bus capacitance

ADVANCE INFORMATION FOR DESIGN-IN INFORMATION

I_{CC} WITH STOPPED CLOCK

Table 5-3. I_{CC} Static

Symbol	Parameter	Min	Max	Unit	Notes
I_{CCS}	Supply Current (static)	0	5	mA	(Note 1)

NOTE:
1. Tested while clock stopped in PH2 and inputs at V_{CC} or V_{SS} with the outputs unloaded. Clock stopped after IO Read at address 25H. EFI and ISACLK2 inputs should be at V_{CC} or V_{SS}.

5

POWER VARIATIONS WITH CAPACITIVE LOADS AT VARIOUS VOLTAGES

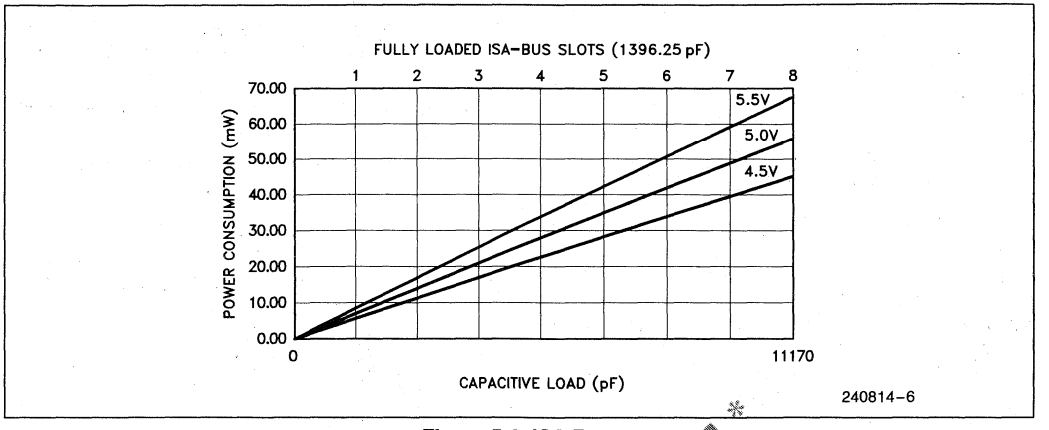
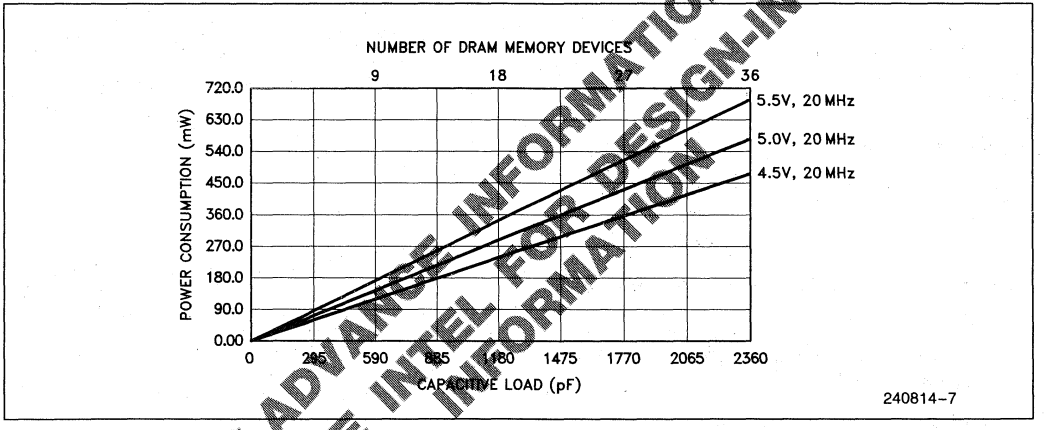


Figure 5-2. ISA Bus



ADVANCE INFORMATION
SEE INTEL FOR DESIGN-IN
INFORMATION

Figure 5-3a. Memory Bus without Cache

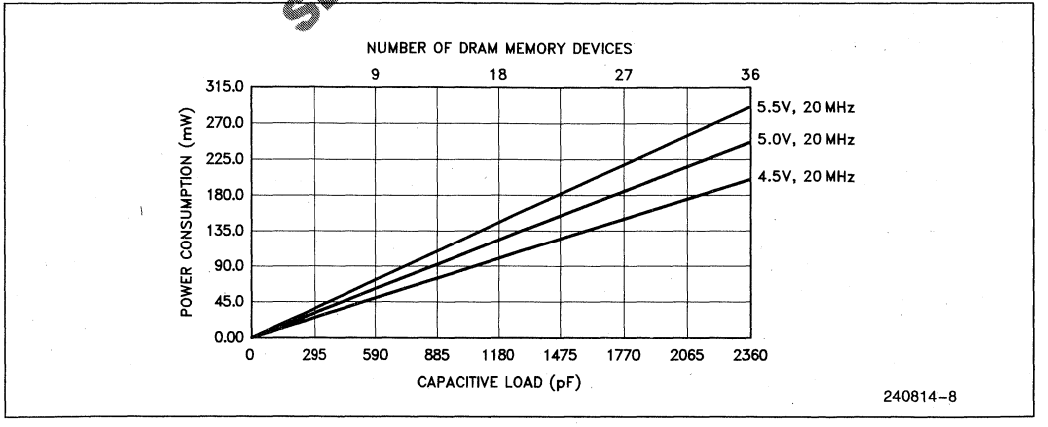


Figure 5-3b. Memory Bus with Cache

Calculation of I_{CC} for Various SL SuperSet System Configurations

A set of three curves with V_{CC} at 4.5V, 5V and 5.5V are plotted in Figure 5-2. Figure 5-2 illustrates the power consumption in milliWatts with respect to the capacitive loading on the ISA bus signals of the 386 SL CPU. The CPUCLK is assumed to be 20 MHz and EFI input is 40 MHz. A similar set of curves are provided for the memory bus without a cache subsystem in Figure 5-3a. The power consumption with respect to load capacitance for the memory bus with a cache subsystem is illustrated in Figure 5-3b. To find the Power (P in milliWatts) of the 386 SL CPU for the configuration of your system, use the following method.

1. Prepare a configuration list for your system including how many ISA-bus connectors, how many memory chips will be provided and whether a cache will be connected or not.
2. From the curves in Figure 5-2, use the voltage of your system and the total capactive load of all of the 386 SL CPU ISA signals to find the power consumed by the ISA-bus interface.
3. If a cache is connected to the 386 SL CPU in your system, use Figure 5-3b to find memory bus power. If cache is not connected, use Figure 5-3a.
4. Find the internal power consumption of the 386 SL CPU from Table 5-4 and the cache internal power and cache bus power from Tables 5-5 and 5-6.
5. For a system with no cache, add the ISA-bus interface power, the memory bus interface power without cache and the internal power. This gives the power consumption of the 386 SL CPU without cache.
6. For a system with cache, add the ISA bus interface power, the memory interface power with cache, the cache internal power, the cache bus interface power and the internal power. This gives the power consumption of the 386 SL CPU with cache.

Table 5-4. Internal Power

Frequency (MHz)	Power (mW)
20	1758.0

Table 5-5. Cache Bus Power (mW)

Freq. (MHz)	4.5V	5.0V	5.5V
20	24.71	30.5	36.91

Table 5-6. Cache Internal Power

Frequency (MHz)	Power (mW)
20	650

As an example, the power consumed by the 386 SL CPU when it is used in a 20 MHz system with 8 memory chips and 2 fully loaded ISA bus expansion slots will be calculated. The system voltage is assumed to be 5V.

From Figure 5-2, the power consumed by the ISA expansion bus interface is found to be 15 mW (the total capacitance of all the pins of a fully loaded AT-bus slot is 1396.25 pF). For a system with no cache, the power consumed by the memory bus for 8 chips is about 140 mW from Figure 5-3a. The internal power at 20 MHz is 1758.0 mW from Table 5-4. The power consumed by 386 SL CPU is the sum of the power for the internal power (ISA bus and CPU core) and memory bus. The total power consumed by the 386 SL CPU for this system is 1913 mW.

For a system with cache, the ISA bus interface power is 15 mW as previously determined. The memory bus interface power is determined from Figure 5-3b is found to be 60 mW. The internal power remains 1758.0 mW. The cache bus power is read off from Table 5-5 to be 30.5 mW and the cache internal power from table 5.6 is 650 mW. Hence, in this system, the 386 SL CPU consumes a total of 2513.5 mW.

82360SL D.C. Specifications

Functional operating range: V_{CC} = 5.0V ±10%, T_{CASE} = 0°C to 90°C.

Table 5-7. 82360SL D.C. Specifications

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	0.8	V	
V_{IH}	Input High Voltage	2.0 V_{CC} − 0.3	V_{CC} + 0.3	V V	 (Note 2)
I_{LI}	Input Leakage Current		±15	μA	(Note 1)
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance		15	pF	
C_{OUT}	Output or I/O Capacitance		15	pF	
I_{CCS1}	Suspend with Slow Refresh		500	μA	(Note 9)
I_{CCS2}	Suspend without Slow Refresh		450	μA	(Note 9)
I_{CC}	Power Supply Current		180	mA	(Note 10)
D.C. Specifications for Standard ISA Bus Signals					
V_{OL}	Output Low Voltage		0.5	V	I_{OL} = 24 mA(4)
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −3.3 mA(4)
D.C. Specifications for Parallel Port					
V_{OL}	Output Low Voltage		0.5	V	I_{OL} = 8 mA(3)
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −2 mA(3)
D.C. Specifications for Open Drain Outputs					
V_{OL}	Output Low Voltage		0.5	V	I_{OL} = 24 mA(5) I_{OL} = 12 mA(6) I_{OL} = 16 mA(11)
D.C. Specifications for All Other Outputs					
V_{OL}	Output Low Voltage		0.5	V	I_{OL} = 12 mA(7)
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −2 mA(7)
D.C. Specifications for Power-Down Mode					
V_{BATT}	Battery Supply Voltage	3.0		V	
I_{BATT}	Battery Supply Current		100 50	μA μA	V_{BATT} = 5V V_{BATT} = 3.0V(8)

ADVANCE INFORMATION
SEE INTEL FOR DESIGN-IN
INFORMATION

NOTES:
1. No pullup or pulldown.
2. For inputs—COMX1, CX1, RTCX1
3. For outputs—LPTD7:0
4. For outputs—OSC, AEN, SA16:0, LA23:17, MEMR#, MEMW#, IOR#, IOW#, SMEMW#, SMEMR#, SBHE#, TC, SD7:0, XD7, HD7, RESETDRV.
5. 0WS#, IOCHRDY, REFRESH#.
6. LPTSTROBE#, LPTAFD, LPTINIT#, LPTSLCTIN#, LPTDIR.
7. For all other outputs of the module.
8. Measured at V_{CC} = 0V, V_{BATT} = 3.0V, 32 kHz RTC clock with input rise time and fall time, $t_r = t_f < 50$ ns.
9. RTC clock at 32 kHz; Timer Clock, Serial clock and SYSCLK stopped; V_{CC} = 5.5V and RTCVCC = 5.5V, C_L = 50 pF with outputs unloaded.
10. I_{CC} tests at maximum frequency with no resisitive loads on the outputs.
11. REFRESH#

6.0 SL SuperSet TIMING SPECIFICATIONS

386 SL CPU A.C. Specifications

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
General: 20 MHz						
Ct 101	Qt1	EFI Period	25	500	ns	QNT1
Ct 102a	Qt2a	EFI High Time at 2V	8		ns	
Ct 102b	Qt2b	EFI High Time at 3.7V	5		ns	
Ct 103a	Qt3a	EFI Low Time at 2V	8		ns	
Ct 103b	Qt3b	EFI Low Time at 0.8V	6 *		ns	
Ct 104	Qt4	EFI Fall Time from (V_{CC} − 0.8V) to 0.8V		8	ns	
Ct 105	Qt5	EFI Rise Time 0.8V to (V_{CC} − 0.8V)		8	ns	
Ct 111		PWRGOOD Minimum Pulse Width	1		EFI	
Ct 111a	Qt21a	PWRGOOD Setup to EFI	12		ns	QNT3
Ct 111b	Qt21b	PWRGOOD Hold Time	4		ns	
Ct 112		CPURESET Minimum Pulse Width	1		EFI	
Ct 112a	Qt22a	CPURESET Setup to EFI	12		ns	QNT3
Ct 112b	Qt22b	CPURESET Hold Time	4		ns	
Ct 113		STPCLK# Minimum Pulse Width	2		EFI	
Ct 113a	Qt23a	STPCLK# Setup to EFI	15		ns	QNT3
Ct 113b	Qt23b	STPCLK# Hold Time	20		ns	
Ct 114a	Qt24a *	SUS_STAT# Setup to EFI	20		ns	QNT3
Ct 114b	Qt24b	SUS_STAT# Hold Time	15		ns	
Ct 115		ONCE# Minimumimum Pulse Width	35		ns	
Ct 115a	Qt25a	ONCE# Setup to EFI	20		ns	QNT3
Ct 115b	Qt25b	ONCE# Hold Time	15		ns	
Ct 116a	Nt2a	SMI# Setup to EFI	15		ns	QNT3
Ct 116b	Nt2b	SMI# Hold Time	21		ns	
Ct 117a	Xtla	INTR Setup to EFI	15		ns	QNT3
Ct 117b	Xtlb	INTR Hold Time	45		ns	
Ct 118a	Xt2a	NMI Setup to EFI	11		ns	QNT3
Ct 118b	Xt2b	NMI Hold Time	16		ns	

ADVANCE INFORMATION SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:
QNT1. EFI maximum period is specified only for the case where a MCP (Math co-processor) is present in the system. NPXCLK period, high and low time are tested at 2V. All other parameters are guaranteed by design characterization.
QNT3. A20GATE, CPURESET, INTR, NMI, ONCE#, PWRGOOD, SMI#, STPCLK# and SUS_STAT# are asynchronous inputs to the 386 SL CPU. Setup and hold times with respect to the EFI input are provided for test purposes only. The minimum setup and hold times are specified for valid recognition at a specific clock edge. The minimum valid pulse width can be extrapolated from the setup and hold times with respect to EFI.

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

386 SL CPU A.C. Specifications (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
ISA-Bus Clock Timings						
Ct 201	Qt31	ISACLK2 Period	62.5		ns	QNT4
Ct 202	Qt32	ISACLK2 High Time at 2V	28	32.5	ns	QNT4
Ct 203	Qt33	ISACLK2 Low Time at 2V	28	32.5	ns	QNT4
Ct 204	Qt34	ISACLK2 Fall Time from (V_{CC} − 0.8V) to 0.8V	*	8	ns	QNT4
Ct 205	Qt35	ISACLK2 Rise Time from 0.8V to (V_{CC} − 0.8V)		8	ns	QNT4
Ct 206	Qt36	ISACLK2 to SYSCLK Delay, Falling to Rising Edge	7	32	ns	
Ct 211	Qt41	SYSCLK Period	125		ns	QNT5
Ct 212	Qt42	SYSCLK High Time at 2V	53		ns	QNT5
Ct 213	Qt43	SYSCLK Low Time at 2V	57		ns	QNT5
Ct 214	Qt44	SYSCLK Fall Time from (V_{CC} − 0.8V) to 0.8V		10	ns	QNT5
Ct 215	Qt45	SYSCLK Rise Time from 0.8V to (V_{CC} − 0.8V)		10	ns	QNT5
Ct 272a	Nt1a	A20GATE Setup to EFI (PH1)	11		ns	QNT3
Ct 272b	Nt1b	A20GATE Hold Time	21		ns	
ISA-Bus Timings *						
Ct 221	G7	BALE Active Delay from T_S phi 2 Low		52	ns	Nt1
Ct 222	G8	BALE Inactive Delay from T_C phi 1 Low	8	47	ns	
Ct 223	G9	LA17–23 Valid Delay from T_C or T_C phi 2 Low		34	ns	
Ct 224	G10	LA17–23 Invalid Delay from T_C phi 2 Low	0		ns	

ADVANCE INFORMATION SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:
QNT4. ISACLK2 minimum period, high and low times are specified with ISACLK2 input = 16 MHz and SYSCLK output = 8 MHz. The ISACLK2 input specifications are provided to ensure that the SYSCLK output, period, minimum high and low time, rise and fall time and ISACLK2 to SYSCLK skew are met.
QNT5. SYSCLK capacitive loading is 20 pF minimum and 120 pF maximum. SYSCLK period, low and high time are tested at 1.5V thresholds. All other parameters are guaranteed by design characterization.

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

386 SL CPU A.C. Specifications (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
ISA-Bus Timings (Continued)						
Ct 225	G13	SA1–19 Valid Delay from T_S phi 2 Low		56	ns	
Ct 226	G13a	SA0–19, SBHE#, LA17–23 Valid Setup to phi 1 Low (External Master)	18		ns	
Ct 227	G14	SA1–19 Invalid Delay from T_S phi 2 Low	0	45	ns	
Ct 228	G15	SA0, SBHE# Valid Delay from T_S phi 2 Low		52	ns	*
Ct 229	G16	SA0, SHBE# Float Delay from T_S phi 1	0	45	ns	
Ct 230	G17	MEMR#, MEMW# Active from T_C phi 1 Low (16-bit Memory Cycles)	7	45	ns	
Ct 231	G17a	Command Active Setup to phi 1 Low (External Master)	17.5		ns	
Ct 232	G17b	HALT# Valid Delay from phi 1 Low		34	ns	NT8
Ct 233	G18	Command Inactive to Float Delay from TI phi 1 Low (External Master)		45	ns	
Ct 234	G19	Command Active Delay from T_C phi 2 Low (IOR#/IOW# 8- or 16-bit, MEM#/MEMW# 8-bit)	7	45	ns	
Ct 235	G20	Command Inactive Delay from Teoc phi 1 Low (MEMR#/MEMW#, IOR#/IOW# and HALT#)		45	ns	NT2
Ct 238	G23	MEMCS16# Setup to T_C phi 1 Low	0		ns	NT6, NT12
Ct 239	G24	MEMCS16# Hold from T_C phi 1 Low	10		ns	NT6, NT12
Ct 240	G25	IOCS16# Setup to T_C phi 2 Low	2		ns	NT7
Ct 241	G26	IOCS16# Hold from Teoc phi 1 Low	0		ns	NT7
Ct 242	G27	ZEROWS# Setup to T_C phi 2 Low	0		ns	NT7, NT9
Ct 244	G29	ZEROWS# Hold from T_C phi 2 Low	10		ns	NT7, NT9
Ct 245	G29a	MEMCS16# Active Delay from Valid Address (External Master Cycles)		64	ns	
Ct 246	G30	SD0–15 Valid Setup to Teoc phi 1 Low	18		ns	Ext. Master
Ct 247	G31	SD0–15 Hold from Teoc phi 1 Low	16		ns	Read Cycle, NT10
Ct 248	G32	SD0–7 Valid Delay from T_S phi 2 Low	30	65	ns	Write Cycle
Ct 249	G33	SD8–15 Valid Delay from T_S phi 2 Low	37	65	ns	Write Cycle
Ct 250	G34	SD0–15 Invalid Delay from Teoc phi 1 Low	4		ns	Write Cycle

ADVANCE INFORMATION
SEE INTEL FOR DESIGN-IN INFORMATION

5

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

386 SL CPU A.C. Specifications (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
ISA-Bus Timings (Continued)						
Ct 251	G35	IOCHRDY Setup to T_C phi 2 Low	0		ns	
Ct 252	G36	IOCHRDY Hold from T_C phi 2 Low	8		ns	NT11
Ct 255	G39	NMI/SMI# Setup to Tx phi 2 Low	16			Asynch
Ct 256	G40	NMI/SMI# Hold from Tx phi 2 Low			bs	Asynch
Ct 257	G41	INTR Setup to Tx phi 2 Low	45		ns	Asynch, NT4
Ct 258	G42	INTR Hold from Tx phi 2 Low			ns	Asynch
Ct 259	G43	INTA Active Delay from T_C phi 2 Low		45	ns	NT16
Ct 260	G44	INTA Inactive Delay from Teoc phi 1 Low		60	ns	NT17
Ct 261	G45	HRQ Setup to T_C or Ti phi 2 Low	15		ns	
Ct 262		HRQ Hold from Th phi 2 Low	5		ns	
Ct 263	G48a	HLDA Active Delay from Th phi 1 Low	7	38	ns	NT3 C_L = 65 pF
Ct 264	G48b	HLDA Inactive Delay from Th phi 1 Low	7	38	ns	C_L = 65 pF
Ct 265	G49	DMA8/16# Setup to Th phi 2 Low	15		ns	NT13, NT14, NT15
Ct 266	G50	MASTER# Setup to Th phi 2 Low	15		ns	NT15
Ct 267	G51	REFREQ Setup to Ti or T_C phi 2 Low	15		ns	
Ct 268	G53c	VGACS# Active Delay from LA[23:17]		35	ns	
Ct 269	G53d	VGACS# Inactive Delay from LA[23:17]		35	ns	
Ct 270	G54a	ROMCSO#/CMUX14# Active Delay from T_S phi 2 Low		48	ns	NT18
Ct 271	G54b	ROMCSO#/CMUX14# Inactive Delay from T_S phi 2 Low		48	ns	NT18
Ct 272	G54c	ROMCSO#/CMUX14# Active Delay from LA[23:17]		41	ns	NT19
Ct 273	G54d	ROMCSO#/CMUX14# Inactive Delay from LA[23:17]		41	ns	NT19
Ct 274	G55a	SMRAMCS# Active Delay from T_S phi 2 Low	10	49	ns	NT18
Ct 275	G55b	SMRAMCS# Inactive Delay from T_S or Ti phi 2 Low	10	49	ns	NT18
Ct 271	G56	TURBO Setup	16		ns	Asynch

ADVANCE INFORMATION SUBJECT TO CHANGE FOR DESIGN-IN

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

386 SL CPU A.C. Specifications (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
ISA-Bus Timings (Continued)						
Ct 276		SD15–0 Valid Delay from IOCHRDY Asserted (External Master)		48	ns	
Ct 277		SD15–0 Data Invalid Delay from MEMR# Inactive (External Master)	7		ns	
Ct 278		SD15–0 Data Invalid Delay from IOR# Inactive (External Master)	7		ns	
Ct 279		SD15–0 Data Setup to MEMW# Active (External Master)	0		ns	
Ct 280		SD15–0 Data Hold from MEMW# Inactive (External Master)	0		ns	
Ct 281		SD15–0 Setup to IOW# Active (External Master)	0		ns	
Ct 282		BALE Active Delay from Th phi 1 Low (External Master)		45	ns	
Ct 283		BALE Inactive from Th phi 1 Low (External Master)		45	ns	
Ct 284		LA23–17, SA19–0, SBHE# Float to Invalid Delay from Th phi 2 (External Master)		54	ns	
Ct 285		LA23–17, SA19–0, SBHE# Invalid to Float Delay from Th phi 1 (External Master)		54	ns	
Ct 286		SA19–17 Delay from LA19–17 (External Master)	10	45	ns	
Ct 287		Command Float to Inactive from Th phi 2 Low (External Master)		45	ns	
Ct 288		Address Setup to Command Active (External Master)	40		ns	
Ct 289		SA15–0 Hold after IOR# or IOW# Inactive (External Master)	15		ns	
Ct 290		IOCS16# Active Delay from Valid Address (External Master)		52	ns	
Ct 291		SD15–0 Delay from IOR# Active (External Master) Read from CPU I/O Ports)		65	ns	
Ct 292		SD15–0 Valid Delay from phi 2 Low (External Master) Read from On Board Memory)		95	ns	Test Only
Ct 293		SD15–0 Hold from IOW# Inactive (External Master)	15		ns	
Ct 294		Byte Swap Delay (External Master)	10	72	ns	NT5

ADVANCE INFORMATION FOR DESIGN-IN INFORMATION

5

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

386 SL CPU A.C. Specifications (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
ISA-Bus Timings (Continued)						
Ct 295		IOCHRDY Invalid from Command Active (External Master)		105	ns	
Ct 296		IOCHRDY Active Delay from phi 2 Low (External Master)		85	ns	Test Only
Ct 297		IOCHRDY Inactive from MEMR# Active (External Master Accessing ROM)		44	ns	

NOTES:

NT1. The ISA bus timings are specified in a synchronous manner with respect to the ISACLK2 input. ISACLK2 input is 16 MHz, which is twice the frequency of the SYSCLK output. Each SYSCLK period represents one T-state and each T-state corresponds to either the beginning of a bus cycle (T_S—Send Status), middle of a bus cycle (T_C—execute command), end of cycle (T_{eoc}), hold (T_h) or idle (T_i). T-States, (T_S, T_C, T_{eoc}, and T_i) are comprised of two ISACLK2 periods (Phi 1 and Phi 2). The ISACLK2 Periods or Phases, (Phi 1 and Phi 2), falling or rising edge are used to reference the synchronous ISA parameters. ISACLK2 Phi 1 falling edge leads SYSCLK rising edge, ISACLK2 Phi 2 falling edge leads SYSCLK falling edge.

NT2. T_{eoc} represents the End of Cycle. The falling edge of ISACLK2 Phi 2 during T_C indicates T_{eoc}.

NT3. After HLDA (Hold Acknowledge) is de-asserted, the 386 SL CPU drives the address bus with the previous address that was latched prior to the beginning of the HLDA cycle. The term "invalid" refers to this latched address. The latched address may or may not be valid for the next CPU bus cycle. At the start of the next CPU bus cycle on an external bus a valid address will be placed on the address bus.

NT4. INTR, NMI, SMI#, and TURBO are asynchronous inputs with respect to ISACLK2 and SYSCLK. These are input signals to the 386 SL CPU. Setup and hold times with respect to the ISACLK2 input are provided for reference. The minimum setup and hold times are specified for valid recognition at a specific clock edge in other timing diagrams with the EFI clock input.

NT5. The setup time is required to ensure that byte swapping is not delayed when an external master reads from an 8-bit device on an odd byte address boundary.

NT6. MEMCS16# is sampled on the falling edge of ISACLK2 Phi 2.

NT7. IOCS16# and ZEROWS# are sampled on the falling edge of ISACLK2 Phi 1.

NT8. HALT timing is identical to a 16-bit ISA bus default memory bus cycle except that no BALE or Status Signal is asserted.

NT9. ZEROWS# and IOCHRDY should not both be driven LOW during the same bus cycle.

NT10. SD0–15 read data is sampled on the falling edge of ISACLK2 Phi 2 at T_{eoc} (End of Cycle).

NT11. IOCHRDY de-asserted (LOW) is sampled on the falling edge of ISACLK2 Phi 2 when Command is active (LOW). De-asserting IOCHRDY# adds incremental wait states (1 SYSCLK long). IOCHRDY should not be held LOW longer than 17 SYSCLKs (2.1 μs.)

NT12. ROM read bus cycles are similar to 8/16 bit ISA bus memory read bus cycles except that MEMCS# is ignored. The strapping pin ROM16/8# is sampled to determine if the ROM read is an 8-bit or 16-bit memory read. Additionally ROMCS0# and/or ROMCS1# are asserted during a ROM read.

NT13. DMA bus cycles are not supported to On-board I/O ports. AEN is HIGH during MASTER, DMA and access to the configuration registers.

NT14. Byte swap timing for 8-bit DMA bus cycles is identical to that of an external master.

NT15. During DMA cycles the 386 SL CPU drives SA17–19 with the value of LA17–19 while HLDA is active. During other Slave cycles (i.e., Refresh and External Master) the 386 SL CPU does not drive SA17–19.

NT16. During the INTA# cycle, SD8–15 should not change state. During the first INTA# pulse SD0–15 are ignored. The second INTA# pulse in an INTA# bus cycle indicates a bus cycle that is similar to an 8-bit I/O read in which the interrupt vector is read from SD0–7.

NT17. The 8259 INTA# minimum pulse width is 160 ns.

NT18. ROMCS0#, ROMCS1# and SMRAMCS# are specified with respect to ISACLK2 when the CPU is the bus master.

NT19. ROMCS0#, ROMCS1# and SMRAMCS# are specified with respect to valid address when an external master controls the bus.

ADVANCE INFORMATION FOR PRE-FIRST SILICON

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

386 SL CPU A.C. Specifications (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
PI-Bus Timings: 20 MHz						
Ct 301		Min. Chip Select and Command Setup to PSTART# Active	30		ns	
Ct 302		Min. Chip Select and Command Hold from PSTART# Active	50		ns	
Ct 303		Max. PRDY# Hold Time after PCMD# Inactive		72	ns	
Ct 304		Min. Read Data Setup Time to PCMD# Inactive	30		ns	
Ct 305		Min. Read Data Hold Time from PCMD# Inactive	9		ns	
Ct 306		Min. PRDY# Active Delay from PSTART# Active	94		ns	
Ct 307		Maximum Write Data Valid Delay from PSTART# Active	*	54	ns	
Ct 308		Min. Write Data Invalid Delay from PCMD# Inactive	27		ns	
Ct 309		Min Address Setup Time to PSTART# Active	20		ns	
Ct 310		Min Address Hold Time from PSTART# Active	58		ns	
Ct 311		PSTART# Pulse Width	50		ns	
Ct 312		Min Delay from PSTART# Active to PCMD# Active	50		ns	
Ct 313		Min Delay from PRDY# Active to PCMD# Inactive	32		ns	
Ct 314		Min Delay from PCMD# Inactive to PSTART# Active	50		ns	
External Master Timings: SYSCLK at 8 MHz (Slave CPU)						
Ct 321	t1s	PW/R# Valid Delay		35	ns	ATCLK2 Sync.
Ct 321	t1s	M/IO# Valid Delay		35	ns	ATCLK2 Sync.
Ct 321	t1s	VGACS# Valid Delay		35	ns	ATCLK2 Sync.
Ct 325	t3s	PSTART# Valid Delay		24	ns	ATCLK2 Sync.
Ct 326	t4s	PCMD# Valid Delay		24	ns	ATCLK2 Sync.
Ct 327a	t5as	PRDY# Set-up	5		ns	ATCLK2 Sync.
Ct 327b	t5bs	PRDY# Hold	25		ns	ATCLK2 Sync.
CPU Master						
Ct 328	t2s	SA[1:16] Valid Delay				Tri-Stated
Ct 328	t2s	SA[17:19] Valid Delay				Tri-Stated
Ct 328	t2s	LA[17:23] Valid Delay				Tri-Stated
Ct 328	t2s	SBHE#, SAO Valid Delay				Tri-Stated
Ct 332	t6s	SD[0:15] Valid Delay				Tri-Stated

ADVANCE INFORMATION
SEE INTEL FOR DESIGN-IN INFORMATION

5

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

386 SL CPU A.C. Specifications (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
CPU Master (Continued)						
Ct 341		PW/R#, PM/IO#, VGACS# Valid Delay from EFI T1 phi 1 High		53	ns	(Note 3)
Ct 342		SA[19:0], LA[23:17], SBHE# Valid Delay from EFI T1 phi 1 High	*	63	ns	
Ct 343a		PSTART# Active (LOW) Delay from EFI T2 phi High		33	ns	
Ct 343b		PSTART# Inactive (HIGH) Delay from EFI T2 phi 2 Low		33	ns	
Ct 344a		PCMD# Active (LOW) Delay from EFI T2 phi 1 High		33	ns	
Ct 344b		PCMD# Inactive (HIGH) Delay from EFI T2 phi 2 Low		33	ns	
Ct 345a		PRDY# Setup to EFI T2 phi 1 Low (CPU is Bus Master)	0		ns	
Ct 345b		PRDY# Hold from EFI T2 phi 2 Low (CPU is Bus Master)	25		ns	
Ct 347a		SD[15:0] Setup to EFI T2 phi 2 Low (CPU Read from PI Bus Slave Device)	21		ns	(Note 7)
Ct 347b	*	SD[15:0] Hold from EFI T2 phi 2 Low (CPU Read from PI Bus Slave Device)	15		ns	
Ct 348		SD[15:0] Valid Delay from EPI T2 phi 2 High (CPU Write to PI Bus Slave Device)		62	ns	

ADVANCE INFORMATION FOR DESIGN-IN

NOTES:
1. VGACS#, FLSHDCS#, PW/R#, PM/IO# and Addresses change for each subsequent read or write.
2. PSTART# indicates a new cycle in which address, status and chip selects are valid before PSTART# is asserted LOW. PRDY# terminates each bus cycle and a new PSTART# is driven if a new address and status signals are available.
3. EFI = 50 MHz, Internal CPU Phase CLK = 25 MHz.
4. ISACLK2 = 16 MHz.
5. Maximum parameters are based on worst case condition of V_{CC} = 4.2V, 120°C.
6. Minimum parameters are based on best case condition of V_{CC} = 5.6V, 10°C.
7. PRDY# setup worst case condition is −4 ns, 0 ns specified for test purposes.

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

386 SL CPU A.C. Specifications (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
Cache Bus Timings: 20 MHz						
Ct 401	t1	CABUS Setup to COE# Active Low	−1		ns	HNT1
Ct 402	t2	COE# Puslse Width	72		ns	HNT2
Ct 403	t3	CCSH#, CCSL# Active to COE# Active	1		ns	HNT3
Ct 404a	t4	CDBUS Setup to COE# Active	36		ns	HNT4
Ct 404b		CDBUS Hold from COE# Active	72		ns	
Ct 405	t5	CABUS Valid to CWE# Inactive High	36		ns	
Ct 406	t6	CWE# Active Width	35		ns	
Ct 407	t7	CDBUS Setup to CWE# Inactive	25		ns	
Ct 408	t8	CDBUS Hold to CWE# Inactive	0		ns	
Ct 409	t9	CABUS Hold to CWE# Inactive	6		ns	
Math Coprocessor Timings: 20 MHz						
Ct 421	Ht11	CA2 Valid Delay (NPX Cyc)	3	25	ns	HNT1
Ct 422	Ht12	NPXADS# Valid Delay	5	27	ns	HNT5
Ct 423	Ht13	NPXW/R# Valid Delay	5	27	ns	HNT5
Ct 424	Ht14	CD Valid Delay (NPX Cycle)	2	35	ns	HNT4
Ct 425a	Ht15a	NPXRDY# Setup	16		ns	
Ct 425b	Ht15b	NPXRDY# Hold	3		ns	
Ct 426a	Ht16a	BUSY#, PEREQ, ERROR# Setup	14		ns	
Ct 426b	Ht16b	BUSY#, PEREQ, ERROR# Hold	5		ns	
Ct 427a	Ht17a	CD Setup (NPX Cycle)	12		ns	
Ct 427b	Ht17b	CD Hold (NPX Cycle)	6		ns	

ADVANCE INFORMATION SEE INTEL FOR DESIGN-IN INFORMATION

NOTES:
QNT1. EFI maximum period is specified only for the case where a MCP (Math Co-processor) is present in the system. NPXCLK, period, high and low time at 2V are tested. All other parameters are guaranteed by design characterization.
QNT2. NPXCLK, NPXRESET Loading: 30 pF. (Timing specified here is for in-system loading, Timing Spec with Tester Loading is TBD.)
HNT1. CA Loading: min 10 pF, max 50 pF.
HNT4. CD Loading: min 10 pF, max 35 pF. (Timing specified here is for in-system loading, Timing Spec with Tester Loading is TBD.)
HNT5. NPXADS#, NPXW/R# Loading: 25 pF. (Timing specified here is for in-system loading, Timing Spec with Tester Loading is TBD.)

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

386 SL CPU A.C. Specifications (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
Math Coprocessor Timings: 20 MHz (Continued)						
Ct 441	Qt11	NPXCLK Period	25	500	ns	QNT1
Ct 442a	Qt12a	NPXCLK High Time 2V	6		ns	QNT2
Ct 442b	Qt12b	NPXCLK High Time 3.7V	3		ns	
Ct 443a	Qt13a	NPXCLK Low Time 2V	6		ns	
Ct 443b	Qt13b	NPXCLK Low Time 0.8V	4		ns	
C t 444	Qt14	NPXCLK Fall Time (V_{CC} − 0.8V) to 0.8V		8	ns	
Ct 445	Qt15	NPXCLK Rise Time 0.8V to (V_{CC} − 0.8V)		* 8	ns	
Ct 446	Qt26	NPXCLK - Delay from RESET - NPXCLK	8	14	ns	
SRAM Mode: 20 MHz Timings						
Ct 501	tOAC	Access Time from OE#	40		ns	2 Wait State
Ct 502	tOAC	Access Time from OE#	50		ns	3 Wait State
Ct 503	tCSD	CE# Setup to OE# Active	40		ns	2 Wait State
Ct 504	tCSD	CE# Setup to OE# Active	50		ns	3 Wait State
Ct 505	tASD	Addr Setup to OE# Active	40		ns	2 Wait State
Ct 506	tASO	Addr Setup to OE# Active	50		ns	3 Wait State
Ct 507	tCSW	CE# Setup to WE# Active	0		ns	2 Wait State
Ct 508	tCSW	CE# Setup to WE# Active	0		ns	3 Wait State
Ct 509	tASW	Addr Setup to WE# Active	0		ns	2 Wait State
Ct 510	tASW	Addr Setup to WE# Active	0		ns	3 Wait State
Ct 511	tWP *	WE# Active Pulse Width	70		ns	2 Wait State
Ct 512	tWP	WE# Active Pulse Width	90		ns	3 Wait State
Ct 513	tWR	WE# Recovery Time	10		ns	2 Wait State
Ct 514	tWR	WE# Recovery Time	10		ns	3 Wait State
Ct 515	tDS	Write Data Setup to WE# Inactive	35		ns	2 Wait State
Ct 516	tDS	Write Data Setup to WE# Inactive	40		ns	3 Wait State
Ct 517	tDH	Write Data Hold from WE# Inactive	0		ns	2 Wait State
Ct 518	tDH	Write Data Hold from WE# Inactive	0		ns	3 Wait State

ADVANCE INFORMATION FOR DESIGN-IN SEE INTEL FOR LATEST INFORMATION

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

386 SL CPU A.C. Specifications (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
SRAM Mode: 20 MHz Timings (Continued)						
Ct 519	tDSO	DIR Setup to OE# Active	0		ns	2 Wait State
Ct 520	tDSO	DIR Setup to OE# Active	0		ns	3 Wait State
Ct 521	tDHO	DEN# Hold from OE# Inactive	0		ns	2 Wait State
Ct 522	tDHO	DEN# Hold from OE# Inactive	0		ns	3 Wait State
Ct 523	tOSD	OE# Inactive Setup to DEN# Active	30	*	ns	2 Wait State
Ct 524	tOSD	OE# Inactive Setup to DEN# Active	40		ns	3 Wait State
Ct 525	tDSW	DIR Inactive Setup to WE# Active	0		ns	2 Wait State
Ct 526	tDSW	DIR Inactive Setup to WE# Active	0		ns	3 Wait State
Ct 527	tDHW	DEN# Hold from WE# Inactive	0		ns	2 Wait State
Ct 528	tDHW	DEN# Hold from WE# Inactive	0		ns	3 Wait State
Ct 529	tDSD	DIR Inactive Setup to DEN# Active	0		ns	2 Wait State
Ct 530	tDSD	DIR Inactive Setup to DEN# Active	0		ns	3 Wait State
Ct 531	tDRH	DIR Hold from DEN# Inactive	0		ns	2 Wait State
Ct 532	tDRH	DIR Hold from DEN# Inactive	0		ns	3 Wait State
Ct 533	tDRS	DIR Setup to DEN# Active	0		ns	2 Wait State
Ct 534	tDRS	DIR Setup to DEN# Active	0		ns	3 Wait State
Ct 535	tASL	Upper Addr Setup to LE Inactive	8		ns	2 Wait State
Ct 536	tASL	Upper Addr Setup to LE Inactive	8		ns	3 Wait State
Ct 537	tAHL	Upper Addr Hold from LE Inactive	0		ns	2 Wait State
Ct 538	tAHL	Upper Addr Hold from LE Inactive	0		ns	3 Wait State
Ct 539	tLP	LE Active Pulse Width	8		ns	2 Wait State
Ct 540	tLP	LE Active Pulse Width	8		ns	3 Wait State
Ct 541	tAVL	Addr Valid Delay from LE Inactive		50	ns	2 Wait State
Ct 542	tAVL	Addr Valid Delay from LE Inactive		70	ns	3 Wait State

ADVANCE INFORMATION
SEE ITE FOR DESIGN-IN
INFORMATION

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

386 SL CPU A.C. Specifications (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
DRAM Mode: 20 MHz Timings						
Ct 601	tASR	Row Addr Setup to RAS# Active	0		ns	F1 Mode
Ct 602	tASR	Row Addr Setup to RAS# Active	0		ns	F2 Mode
Ct 603	tASR	Row Addr Setup to RAS# Active	0		ns	P1 Mode
Ct 605	tRAH	Row Addr Hold from RAS# Active	12		ns	F1 Mode
Ct 606	tRAH	Row Addr Hold from RAS# Active	12		ns	F2 Mode
Ct 607	tRAH	Row Addr Hold from RAS# Active	12		ns	P1 Mode
Ct 609	tASC	Col Addr Setup to CAS# Active	0		ns	F1 Mode
Ct 610	tASC	Col Addr Setup to CAS# Active	0		ns	F2 Mode
Ct 611	tASC	Col Addr Setup to CAS# Active	0		ns	P1 Mode
Ct 613	tCAH	Col Addr Hold from CAS# Active	15		ns	F1 Mode
Ct 614	tCAH	Col Addr Hold from CAS# Active	15		ns	F2 Mode
Ct 615	tCAH	Col Addr Hold from CAS# Active	20		ns	P1 Mode
Ct 617	tRCD	RAS# to CAS# Delay	20		ns	F1 Mode
Ct 618	tRCD	RAS# to CAS# Delay	20		ns	F2 Mode
Ct 619	tRCD	RAS# to CAS# Delay	20		ns	P1 Mode
Ct 621	tCSH	CAS# Hold Time from RAS# Active	80		ns	F1 Mode
Ct 622	tCSH	CAS# Hold Time from RAS# Active	100		ns	F2 Mode
Ct 623	tCSH	CAS# Hold Time from RAS# Active	100		ns	P1 Mode
Ct 625	tRSH	RAS# Hold Time from CAS# Active	20		ns	F1 Mode
Ct 626	tRSH	RAS# Hold Time from CAS# Active	20		ns	F2 Mode
Ct 627	tRSH	RAS# Hold Time from CAS# Active	30		ns	P1 Mode
Ct 629	tWCS	WE# Setup to CAS# Active (Write)	0		ns	F1 Mode
Ct 630	tWCS	WE# Setup to CAS# Active (Write)	0		ns	F2 Mode
Ct 631	tWCS	WE# Setup to CAS# Active (Write)	0		ns	P1 Mode
Ct 633	tWCH	WE# Hold from, CAS# Active (Write)	20		ns	F1 Mode
Ct 634	tWCH	WE# Hold from, CAS# Active (Write)	20		ns	F2 Mode
Ct 635	tWCH	WE# Hold from, CAS# Active (Write)	20		ns	P1 Mode
Ct 637	tRCS	WE# Setup to CAS# Active (Read)	0		ns	F1 Mode
Ct 638	tRCS	WE# Setup to CAS# Active (Read)	0		ns	F2 Mode
Ct 639	tRCS	WE# Setup to CAS# Active (Read)	0		ns	P1 Mode
Ct 641	tRCH	WE# Hold from CAS# Inactive (Read)	0		ns	F1 Mode
Ct 642	tRCH	WE# Hold from CAS# Inactive (Read)	0		ns	F2 Mode
Ct 643	tRCH	WE# Hold from CAS# Inactive (Read)	0		ns	P1 Mode
Ct 645	tWDS	Write Data Setup to CAS# Active	0		ns	F1 Mode

ADVANCE INFORMATION SEE INTEL FOR DESIGN-IN INFORMATION

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

386 SL CPU A.C. Specifications (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
DRAM Mode: 20 MHz Timings (Continued)						
Ct 646	tWDS	Write Data Setup to CAS# Active	0		ns	F2 Mode
Ct 647	tWDS	Write Data Setup to CAS# Active	0		ns	P1 Mode
Ct 649	tWDH	Write Data Hold from CAS# Active	20		ns	F1 Mode
Ct 650	tWDH	Write Data Hold from CAS# Active	20		ns	F2 Mode
Ct 651	tWDH	Write Data Hold from CAS# Active	20		ns	P1 Mode
Ct 653	tRAC	Access Time from RAS# Active	80		ns	F1 Mode
Ct 654	tRAC	Access Time from RAS# Active	100		ns	F2 Mode
Ct 655	tRAC	Access Time from RAS# Active	100*		ns	P1 Mode
Ct 657	tCAC	Access Time from CAS# Active	20		ns	F1 Mode
Ct 658	tCAC	Access Time from CAS# Active	20		ns	F2 Mode
Ct 659	tCAC	Access Time from CAS# Active	20		ns	P1 Mode
Ct 661	tRDH	Read Data Hold from CAS# Inactive	0		ns	F1 Mode
Ct 662	tRDH	Read Data Hold from CAS# Inactive	0		ns	F2 Mode
Ct 663	tRDH	Read Data Hold from CAS# Inactive	0		ns	P1 Mode
Ct 665	tRAS	RAS# Active Pulse Width	80	*	ns	F1 Mode
Ct 666	tRAS	RAS# Active Pulse Width	100	*	ns	F2 Mode
Ct 667	tRAS	RAS# Active Pulse Width	100	*	ns	P1 Mode
Ct 669	tCAS	CAS# Active Pulse Width	25		ns	F1 Mode
Ct 670	tCAS	CAS# Active Pulse Width	25		ns	F2 Mode
Ct 671	tCAS	CAS# Active Pulse Width	35		ns	P1 Mode
Ct 673	tRP	RAS# Precharge Pulse Width	70		ns	F1 Mode
Ct 674	tRP	RAS# Precharge Pulse Width	90		ns	F2 Mode
Ct 675	tRP	RAS# Precharge Pulse Width	110		ns	P1 Mode
Ct 677	tCP	CAS# Precharge Pulse Width	15		ns	F1 Mode
Ct 678	tCP	CAS# Precharge Pulse Width	15		ns	F2 Mode
Ct 679	tCP	CAS# Precharge Pulse Width	15		ns	P1 Mode
Ct 681	tPSW	PARx# Setup to CAS# Active (Write)	1		ns	F1 Mode
Ct 682	tPSW	PARx# Setup to CAS# Active (Write)	1		ns	F2 Mode
Ct 683	tPSW	PARx# Setup to CAS# Active (Write)	1		ns	P1 Mode
Ct 685	tPHW	PARx# Hold from CAS# Active (Write)	20		ns	F1 Mode
Ct 686	tPHW	PARx# Hold from CAS# Active (Write)	20		ns	F2 Mode
Ct 687	tPHW	PARx# Hold from CAS# Active (Write)	20		ns	P1 Mode
Ct 689	tPVR	PARx# Valid from CAS# Active (Read)	27		ns	F1 Mode
Ct 690	tPVR	PARx# Valid from CAS# Active (Read)	27		ns	F2 Mode

ADVANCE INFORMATION FOR INTEL DESIGN-IN SEE INTEL

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

386 SL CPU A.C. Specifications (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
DRAM Mode: 20 MHz Timings (Continued)						
Ct 691	tPVR	PARx# Valid from CAS# Active (Read)	42		ns	P1 Mode
Ct 693	tPHR	PARx# Hold from CAS# Inactive (Read)	1		ns	F1 Mode
Ct 694	tPHR	PARx# Hold from CAS# Inactive (Read)			ns	F2 Mode
Ct 695	tPHR	PARx# Hold from CAS# Inactive (Read)	1		ns	P1 Mode
Parity Error						
Ct 701	tPED	PERR# Delay from SYSCLK		38	ns	
Ct 702	tCSR	CAS# Setup to RAS# Active (DRAM Refresh)	10		ns	
Ct 703	tCHR	CAS# Hold from RAS# Active (DRAM Refresh)	30		ns	
Ct 704	tWRP	WE# Setup to RAS# Active (DRAM Refresh)	15		ns	
Ct 705	tWRP	WE# Hold from RAS# Active (DRAM Refresh)	15		ns	
Ct 706	tRDS	RAS# Active Delay from SYSCLK (DRAM DMA/Master)		55	ns	
Ct 707	tADS	Address Valid Delay from SYSCLK (DRAM DMA/Master)		65	ns	

ADVANCE INFORMATION SUBJECT TO CHANGE FOR DESIGN-IN INFORMATION

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

82360SL I/O Timing Specifications Summary

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
It1		SYSCLK Period	125		ns	
It2		SYSCLK Low Time @V_{IL} = 1.5V	50		ns	
It3		SYSCLK High Time @V_{IL} = 1.5V	50		ns	
It4		SYSCLK Rise Time and Fall time		10	ns	
It5		RESETDRV Inactive (LOW) from PWRGOOD Active (HIGH)		40	ns	
It5a		RESETDRV Active (HIGH) from SYSCLK (During Resume after Suspend)		125		
It6a		A20GATE Active (HIGH) Delay from KBDA20 Active (HIGH)		30	ns	
It6b		A20GATE Active (HIGH) Delay from SYSCLK *		45	ns	
It7		SYSCLK to KBDCLK Delay		30	ns	
It8a		RC#/PERR#/IOCHCK# Pulse Width	250		ns	
It8b		RC#/PERR#/IOCHCK# Setup to SYSCLK Falling Edge	12		ns	
It9a		CPURESET Active (HIGH) from SYSCLK	5	50	ns	
It10a		NMI Active (HIGH) from SYSCLK		125	ns	
It10b		NMI Inactive from IOW# Active (LOW)	0			
It11		RTCRESET# Pulse Width	5		ns	
It14		BALE hold from SYSCLK	2	45	ns	
It15		IOR#/IOW#/INTA# Input Active (LOW) Delay from SYSCLK Low		20	ns	
It15a		IOR#/IOW#/MEMW# Output Active (LOW) Delay from SYSCLK		90	ns	
It16		IOR#/IOW#/INTA#/MEMW#/MEMR# Input Inactive from SYSCLK		35	ns	
It16a		IOR#/IOW# Output Inactive from SYSCLK		120	ns	
It17		ZEROWS# Output Active from SYSCLK		65	ns	
It18		ZEROWS# Output Inactive from SYSCLK	0		ns	
It19		BALE Setup to SYSCLK (DMA Cycle)	18		ns	
It20		IOCHRDY Input Active Setup to SYSCLK	15		ns	
It20a		IOCHRDY Input Inactive Setup to SYSCLK	15		ns	
It21		DMA8/16# Active Delay from SYSCLK		65	ns	
It22		DMA8/16# Inactive Delay from SYSCLK (4 MHz DMACLK)		65	ns	
It22a		DMA8/16# Inactive Delay from SYSCLK Low (8 MHz DMACLK)		65	ns	
It23		AEN Active from HLDA Active		35	ns	

ADVANCE INFORMATION SUBJECT TO CHANGE FOR DESIGN-IN INFORMATION

5

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

82360SL I/O Timing Specifications Summary (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
It24		AEN Inactive Delay from HLDA Inactive		35	ns	
It24f		AEN Inactive from SYSCLK		65	ns	
It25		SA15:0, SBHE# Valid Delay from SYSCLK	10	100	ns	
It26		SA16 (Only if DMA8/16# = 0) SA15:0, SHBE# Valid Output Hold from SYSCLK	6		ns	
It26a		SA16 (Only if DMA 8/16# = 1), LA17:23 Valid Output Hold from IOR#/IOW#/MEMR#/MEMW# Output	10		ns	
It26f		SA16:0, LA17:23, SBHE# Float Delay from SYSCLK		90	ns	
It27		DACKx# Active Delay from SYSCLK (4 MHz DMACLK)		75	ns	
It27a		DACKx# Active Delay from SYSCLK Low (8 MHz DMACLK)		75	ns	
It28		DACKx# Inactive Delay from SYSCLK (4 MHz DMACLK)		75	ns	
It28a		DACKx# Inactive Delay from SYSCLK Low (8 MHz DMACLK)		75	ns	
It29		IOR#/IOW#/MEMW# Float-to-Drive-Inactive from SYSCLK		75	ns	
It30		IOR#/IOW#/MEMW# Float Delay from SYSCLK#		75	ns	
It30a		SMRAMCS# Setup to MEMR#/MEMW# Active		75	ns	
It31		MEMR#/MEMW# Input Active Delay from SYSCLK		70	ns	
It31a		MEMR#/MEMW# Output Active Delay from SYSCLK		70	ns	
It32a		MEMR#/MEMW# Output Inactive Delay from SYSCLK		75	ns	
It33		T/C Active Delay from SYSCLK		85	ns	
It34		T/C Inactive Delay from SYSCLK		85	ns	
It35		TIM2CLK2 Period	125		ns	
It36		TIM2CLK2 Low Time	55		ns	
It37		TIM2CLK2 High Time	55		ns	
It38		TIM2CLK2 Rise Time		25	ns	
It39		TIM2CLK2 Fall Time		25	ns	
It40		TIM2GAT2 High Pulse Width	45		ns	

ADVANCE INFORMATION SEE INTEL FOR DESIGN-IN INFORMATION

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

82360SL I/O Timing Specifications Summary (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
It41		TIM2GAT2 Low Pulse Width	45		ns	
It42		TIM2GAT2 Setup to TIM2CLK2	45		ns	
It43		TIM2GAT2 Hold from TIM2CLK2	45		ns	
It44		TIM2OUT2 from TIM2CLK2 High to Low		110	ns	
It45		TIM2OUT2 from TIM2GAT2 High to Low		110	ns	
It46		SPKR Active Delay from TIM2GAT2 (When EXTAUD is Set)		120	ns	
It50		REFRESH# Active to MEMR# Output Active	150		ns	
It52		Address Valid to MEMR# Active	40		ns	
It53		MEMR# Output Inactive from IOCHRDY Input Low to High (During a Master Refresh)	125		ns	
It55		IOCHRDY Pulse Width		750	ns	
It56		MEMR# Output Pulse Width for Refresh	2		SYSCLK	
It59		FLPCS#/C8042CS#/HDCS0#/HDCS1# Active Setup to Command Active	25		ns	
It60		FLPCS#/C8042CS#/HDCS0#/HDCS1# Output Hold from Command Inactive	10		ns	
It60a		PMRAMCS# Hold from MEMR#/MEMW#	10		ns	
It69		DRQx Setup to SYSCLK High to Low	0		ns	
It79		EXTRTCAS Pulse Width	3	4	SYSCLK	
It80		IOCS16# Setup to Command	10		ns	
It81		IOCS16# Hold from Command	10		ns	
It82		STPCLK# Delay from SYSCLK		100	ns	
It83		PMI# from SYSCLK		100	ns	
It84		SMOUTx from SYSCLK		100	ns	
It85		SUS_STAT# from SYSCLK		100	ns	
It86		IOCHRDY Output from SYSCLK		60	ns	
It94		Delay from IOW# to Modem Output (RTS#, DTR#)		200	ns	
It109		KBDCLK Period (8 MHz)	125		ns	
		KBDCLK Period (4 MHz)	250		ns	
		KBDCLK Period (2 MHz)	500		ns	
It110		KBDCLK High Time (8 MHz)	40		ns	
		KBDCLK High Time (4 MHz)	95		ns	
		KBDCLK High Time (2 MHz)	200		ns	

ADVANCE INFORMATION SUBJECT TO FURTHER DESIGN-IN INFORMATION

5

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

82360SL I/O Timing Specifications Summary (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
It111		KBDCLK Low Time (8 MHz)	40		ns	
		KBDCLK Low Time (4 MHz)	95		ns	
		KBDCLK Low Time (2 MHz)	200		ns	
It117		HRQ Inactive to HLDA Inactive	185		ns	
It118		HLDA Inactive to HRQ Active (Back to Back Hold Acknowledge Cycles)	0		ns	
It120		IRQ1, 6, 10, 11: 12, 14, 15, ERROR# Pulse Width	50		ns	
It121		INTR Output Delay from IRQ1, 6, 10: 11, 12, 14, 15, ERROR#		100	ns	
It122		Data Output Valid from INTA# Active		120	ns	
It123		Data Output Hold from INTA# Inactive	10		ns	
It123f		Data Float from INTA# Inactive		35	ns	
It124a		SD7 Read Data Output Hold from MEMR# Inactive	5		ns	
It124f		SD7 Float from MEMR# Inactive		35	ns	
It125		Write Data Input Setup to MEMW#	40		ns	
It125a		XD7 Output Valid from MEMW# Active		45	ns	
It126		Write Data Input Hold from MEMW#	15		ns	
It126a		XD7 Output Hold from MEMW# Inactive	5		ns	
It126f		XD7 Float from MEMW# Inactive		45	ns	
It129		SMEMR#/SMEMW# Active from MEMR#/MEMW#		30	ns	
It129a		SMEMR#/SMEMW# Inactive from MEMR#/MEMW# Inactive	3	30	ns	
It200		BALE Active from SYSCLK Low	2	45	ns	
It201		Write Data Input Setup to IOW# Active	40		ns	
It202		Write Data Input Hold from IOW# Inactive	15		ns	
It203		Read Data Output Setup to IOR# Inactive	62		ns	
It204		Read Data Output Hold from IOR# Inactive	0		ns	
It204f		Data Bus Float from IOR#/MEMR#		35	ns	
It205		BALE Active Pulse Width	50		ns	
It206		Address Input Valid Setup to BALE Inactive	30		ns	
It207		AEN Active from SYSCLK during Indexed I/O Writes		80	ns	
It209		IOW# to EXTRTCAS		100	ns	
It210		XD7 Output Valid from IOW# Active		45	ns	

ADVANCE INFORMATION SUBJECT TO CHANGE FOR DESIGN-IN INFORMATION

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

82360SL I/O Timing Specifications Summary (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
It211		XD7 Output Hold from IOW# Inactive	5		ns	
It211f		XD7 Output Float from IOW# Inactive		45	ns	
It212		EXTRTCRW#/EXTRTCDS Active from Command Active		35	ns	
It213		EXTRTCRW#/ EXTRTCDS Hold from Command Inactive		35	ns	
It214		XDEN# Output Delay from IOR#/IOW#, MEMR# Inputs	10	50	ns	
It214a		XDEN# Output Delay from DACK2 Output	10	50	ns	
It215a		XDEN# Output Active from XDIR Output Active	0		ns	
It215b		XDIR Output Inactive from XDEN# Output Inactive	10		ns	
It216		SD7 Read Data Output Delay from XD7/HD7 Input		35	ns	
It217		SD7 Read Data Output Hold from IOR# Inactive	5		ns	
It217f		SD7 Float from IOR# Inactive		35	ns	
It218		Address-in Hold from Command Inactive	11		ns	
It219		HDENL#/HDENH# Output Active Delay from Command		35	ns	
It219a		HDENL#/HDENH# Output Inactive Delay from Command Inactive	5		ns	
It220	*	HD7 Output Valid from IOW# Active		45	ns	
It221		HD7 Output Hold from IOW# Inactive	10		ns	
It221f		HD7 Output Float from IOW# Inactive		35	ns	
It222		HALT# Input Setup to SYSCLK Low	20		ns	
It223		HALT# Input Hold from SYSCLK	20			
It224		XD7/HD7 Input Setup to IOR#/MEMR#	60		ns	
It225		XD7/HD7 Input Hold from IOR#/MEMR#	0		ns	
It230		XD7 Output Valid from EXTRTCRW# Active		35	ns	
It231		XD7 Output Hold from EXTRTCRW# Inactive	0		ns	
It231f		XD7 Output Float from EXTRTCRW#		35	ns	

ADVANCE INFORMATION
FOR INTEL DESIGN-IN
INFORMATION

6.0 SL SuperSet TIMING SPECIFICATIONS (Continued)

82360SL I/O Timing Specifications Summary (Continued)

Symbol	Alt Symbol	Parameter	Min	Max	Unit	Notes
It305		SA16, LA23:17 Valid Delay from SYSCLK	10	150	ns	
It311		HRQ Output Active from SYSCLK		45	ns	
It312		HLDA Setup to SYSCLK	18		ns	
It314		HRQ Inactive from SYSCLK	5		ns	
It317		REFREQ Active from SYSCLK		45	ns	
It319		REFREQ Inactive from SYSCLK		45	ns	
It322		MASTER# Active to REFRSH# Input Active Delay	25		ns	
It324		REFRESH# Output Active from HLDA		35	ns	
It325		REFRESH# Output Inactive from SYSCLK	5		ns	
It326		REFRESH# Input to REFREQ Active		30	ns	
It327		REFRESH# Input to REFREQ Inactive	5		ns	
It328		REFRESH# Pulse Width	4	5	SYSC	
It329		REFREQ Pulse Width during Master# Cycle	4	5	SYSC	
It330		DACKx# to MASTER# Delay	0		ns	
It331		AEN Delay from MASTER#	0	49	ns	
It332		Alternate Master Drives Address and Data		125	ns	
It333		MASTER# Delay from DRQx Inactive		100	ns	
It334		Alternate Master Tri-States Bus Signal	0		ns	

PRELIMINARY INFORMATION SUBJECT TO CHANGE WITHOUT NOTICE

7.0 SL SuperSet TIMING DIAGRAMS

7.1 386™ SL CPU Timing Diagrams

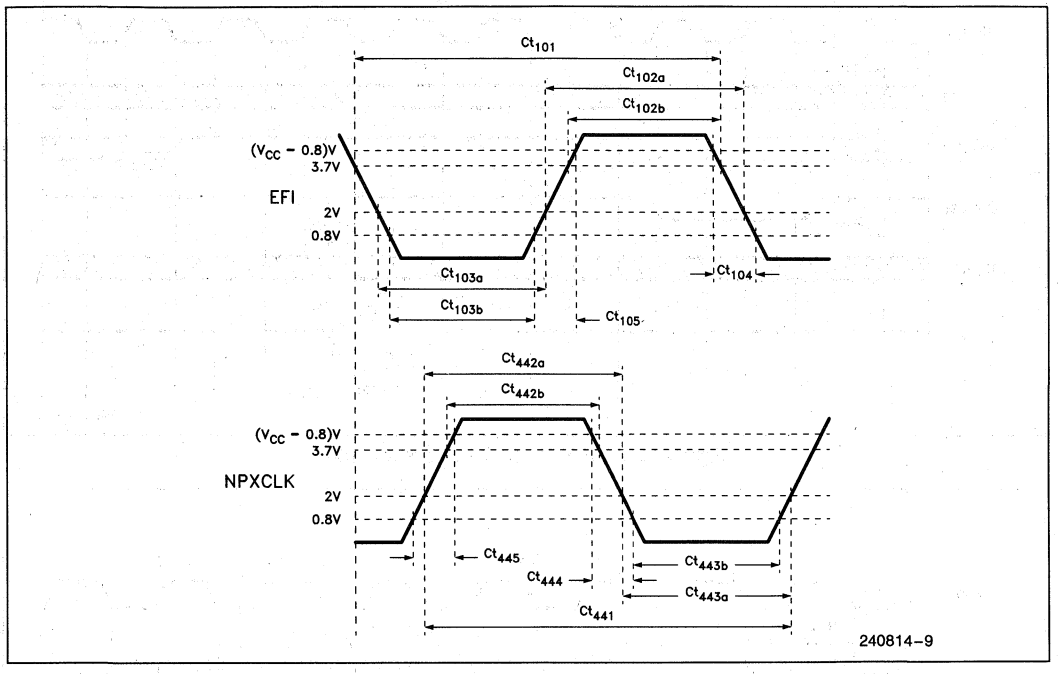


Figure 7.1.1. Clocks

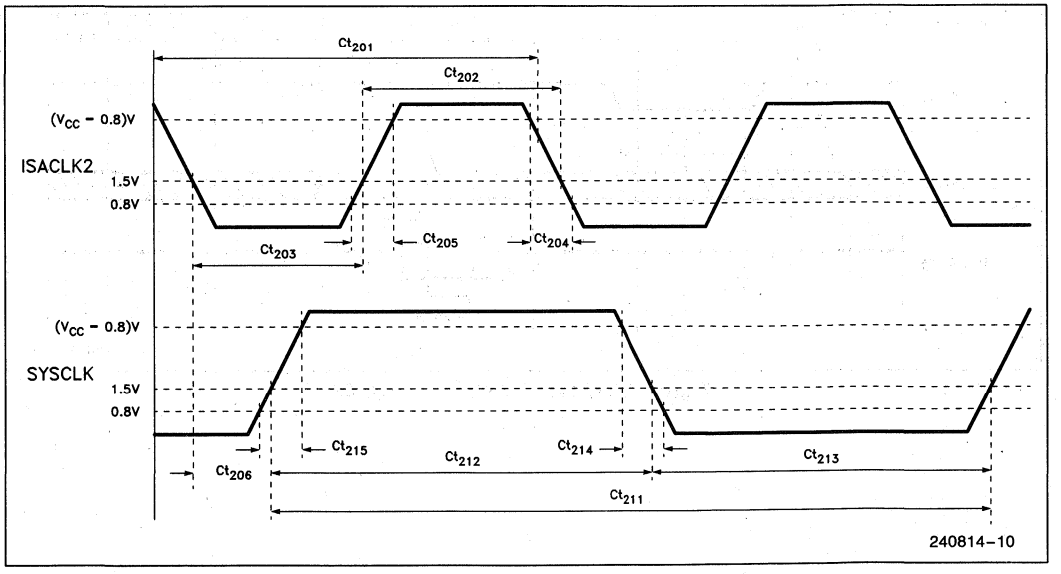


Figure 7.1.2. Clocks

7.1 386™ SL CPU Timing Diagrams (Continued)

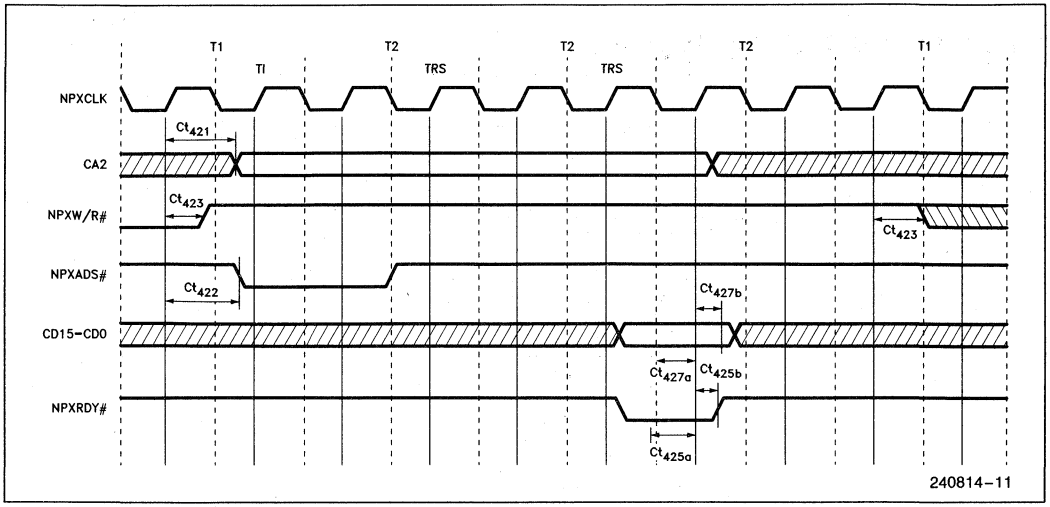


Figure 7.1.3. 386 SL CPU Read from MCP

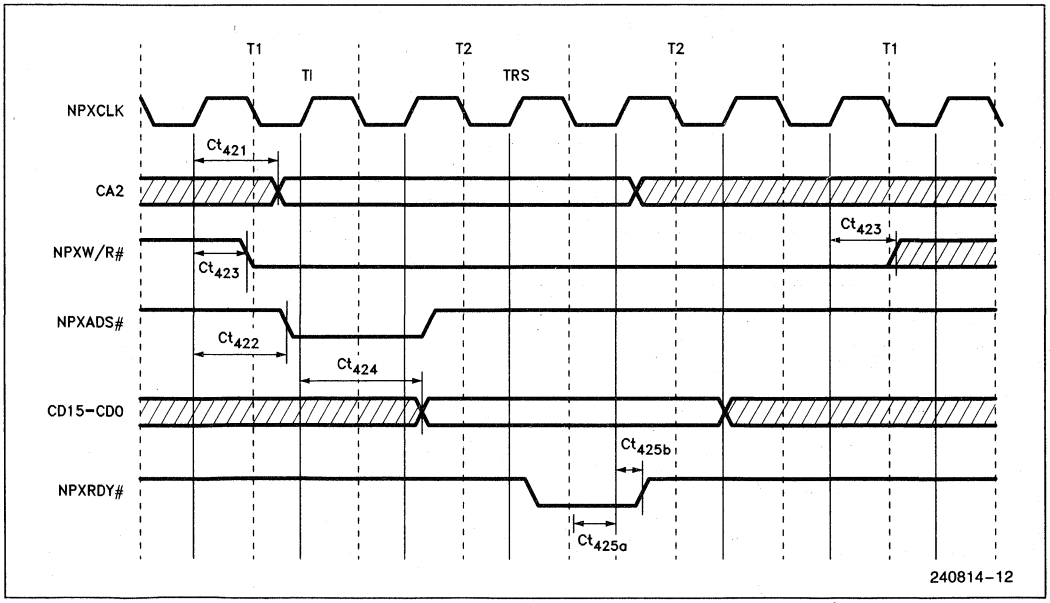
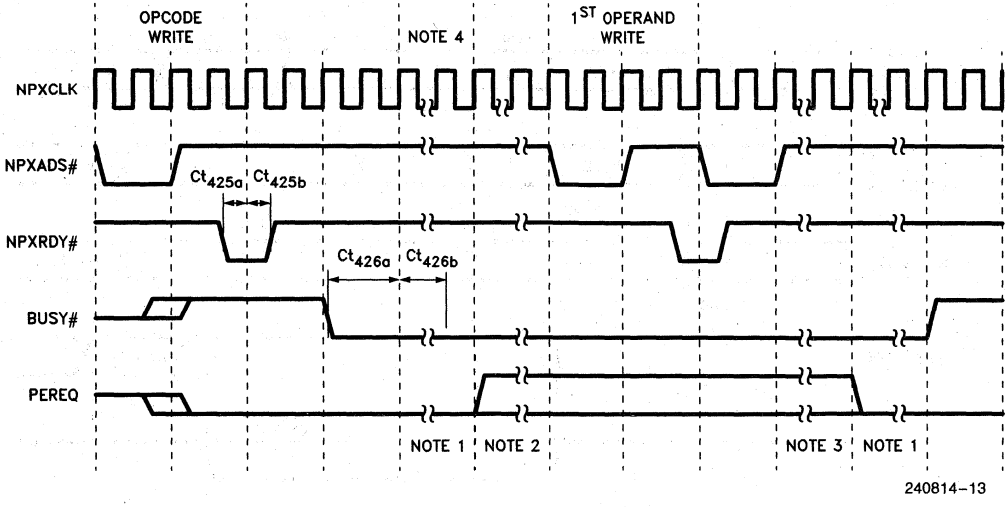


Figure 7.1.4. 386 SL CPU Write to MCP

7.1 386™ SL CPU Timing Diagrams (Continued)



NOTES:
1. Instruction dependent.
2. PEREQ is an asynchronous input to the 386 SL CPU. Instruction dependent as to when it is asserted.
3. Additional operand transfers.
4. Memory read (operand) cycles not shown.

Figure 7.1.5. MCP BUSY# and PEREQ Timings

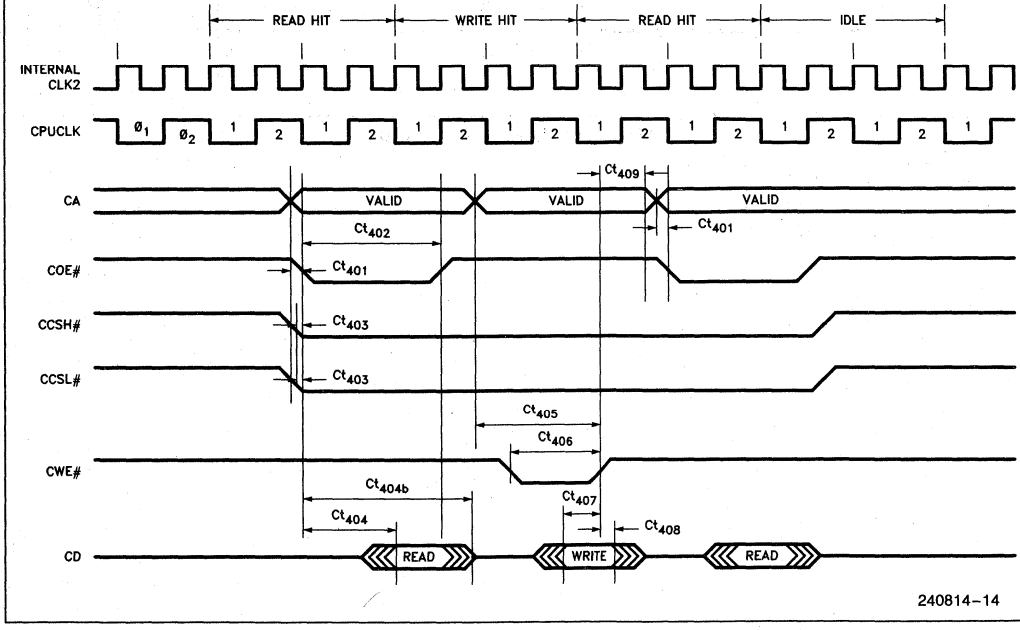


Figure 7.1.6. Cache Read/Write Hit Cycles

5

7.1 386™ SL CPU Timing Diagrams (Continued)

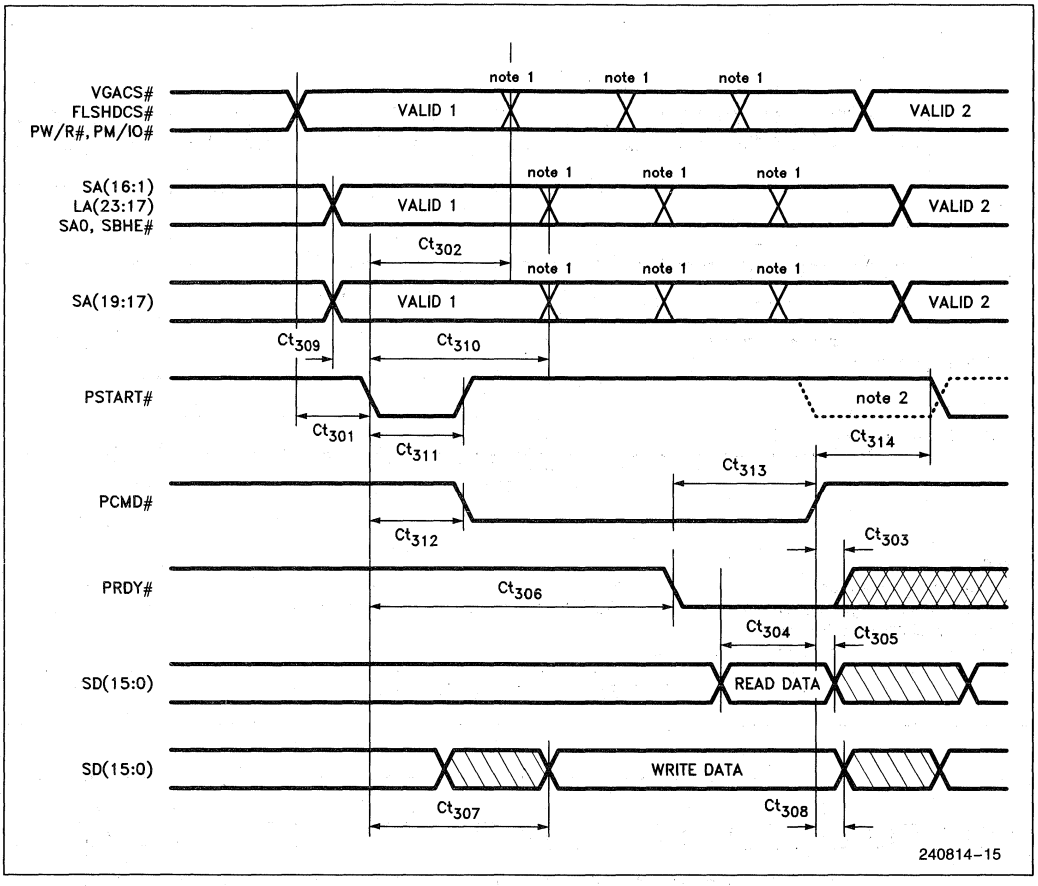


Figure 7.1.7. PI-Bus Timings

7.1 386™ SL CPU Timing Diagrams (Continued)

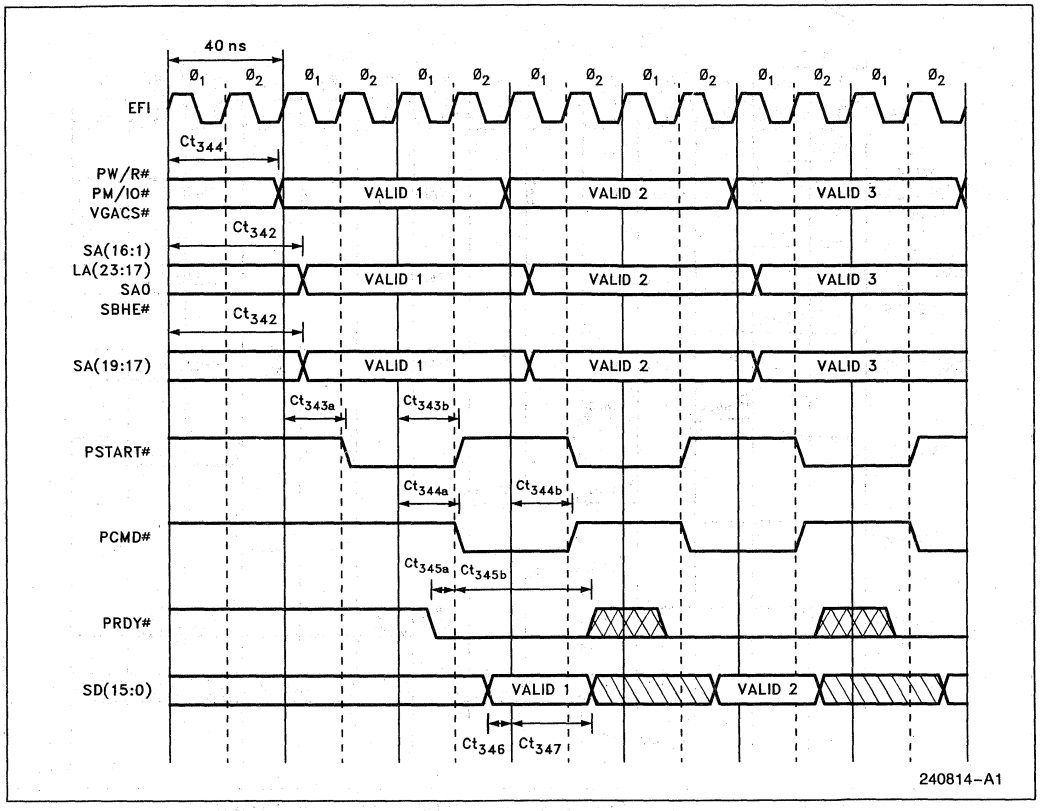


Figure 7.1.8a. PI Bus Synchronous CPU Generated Cycles (Read)

5

7.1 386™ SL CPU Timing Diagrams (Continued)

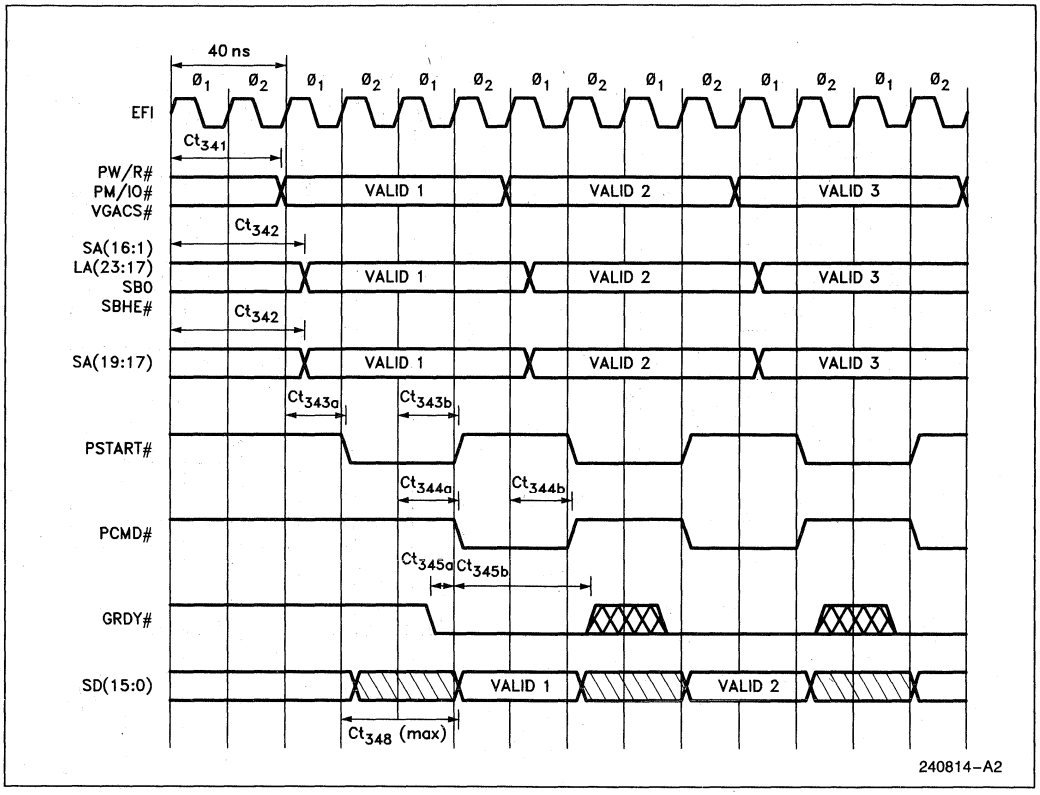


Figure 7.1.8b. PI-Bus Synchronous CPU Generated Cycles (Write)

7.1 386™ SL CPU Timing Diagrams (Continued)

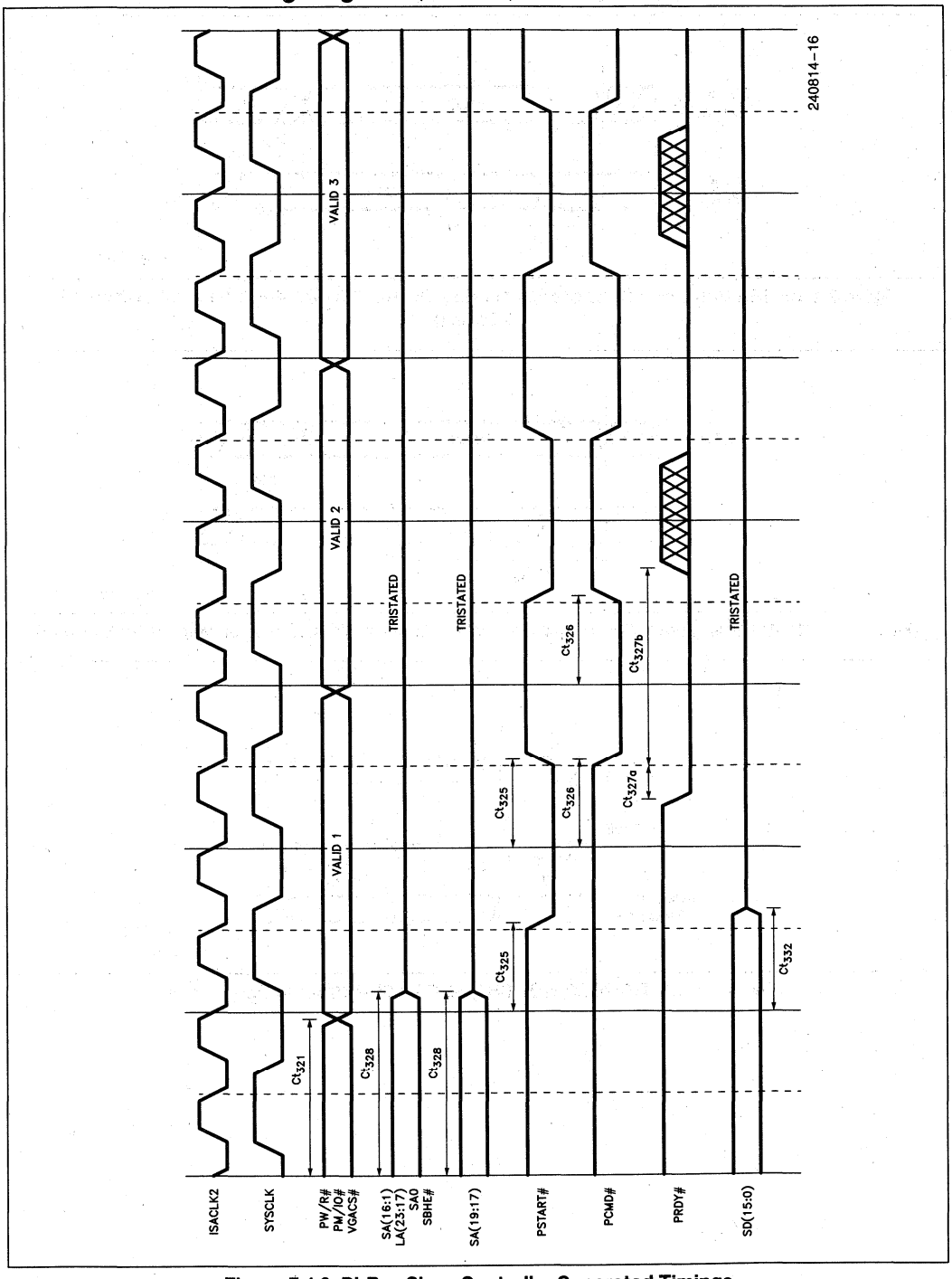


Figure 7.1.9. PI-Bus Slave Controller Generated Timings

7.1 386™ SL CPU Timing Diagrams (Continued)

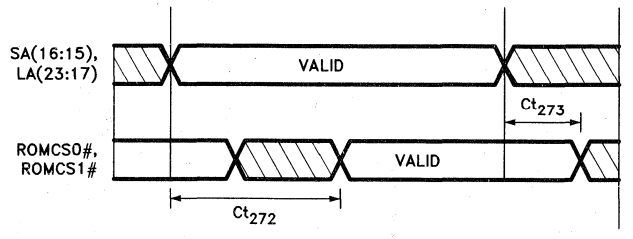


Figure 7.1.10. ISA Bus Slave Controller Generated Timings (ROMCS0#/CS1# with respect to Address)

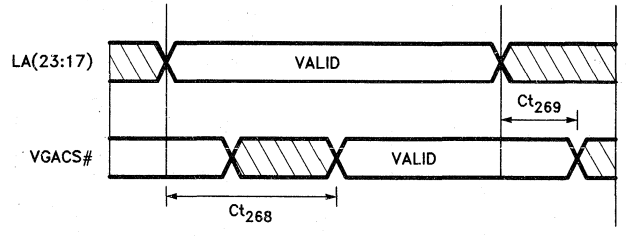


Figure 7.1.11. ISA Bus Master Controller Generated Timings (VGACS# with respect to Address)

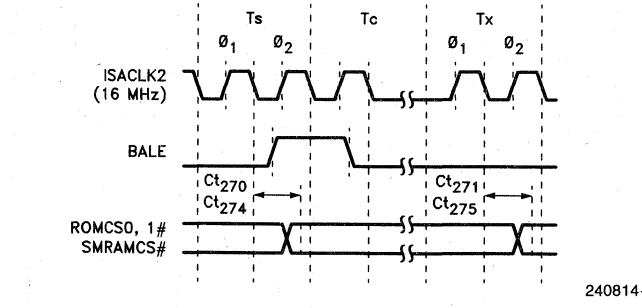


Figure 7.1.12. ROMCS0, ROMCS1, SMRAMCS# Propagation Delays

7.1 386™ SL CPU Timing Diagrams (Continued)

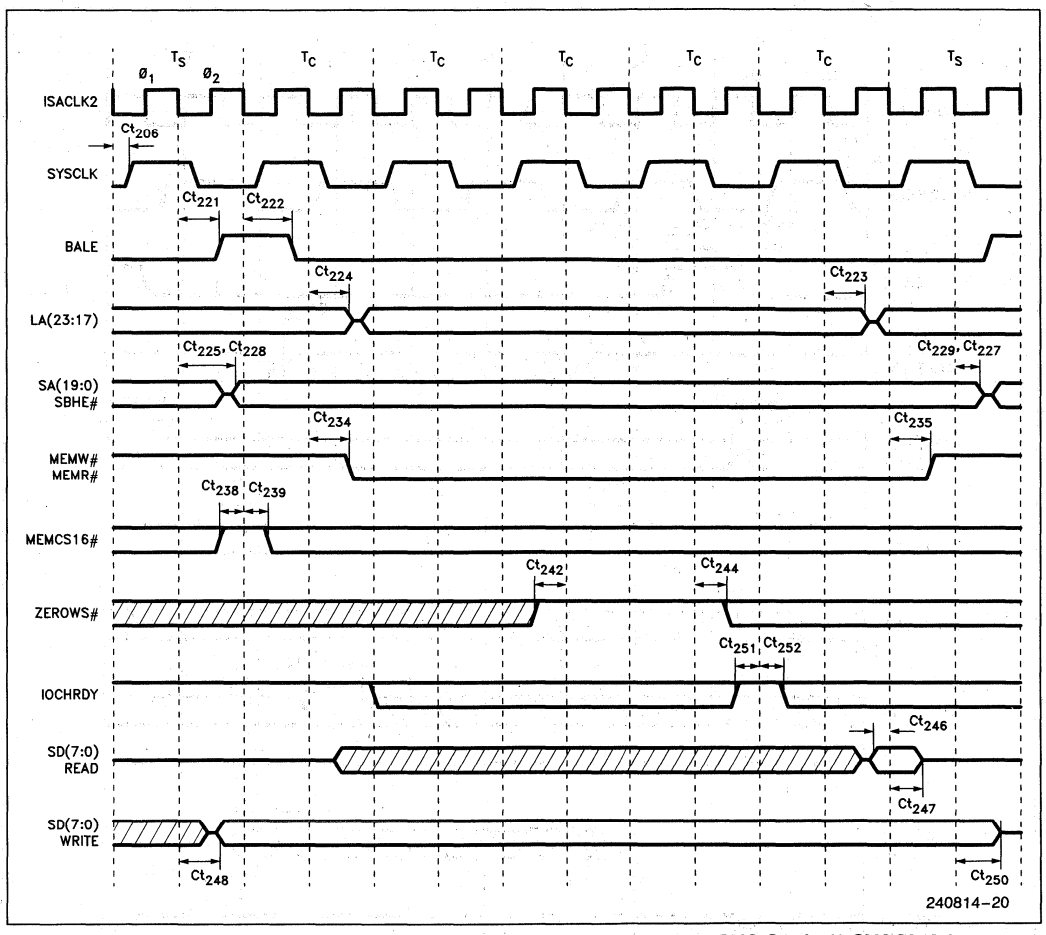


Figure 7.1.13. ISA Bus 8-Bit Memory Read/Write Standard ISA BUS Cycle (6 SYSCLKs)

5

7.1 386™ SL CPU Timing Diagrams (Continued)

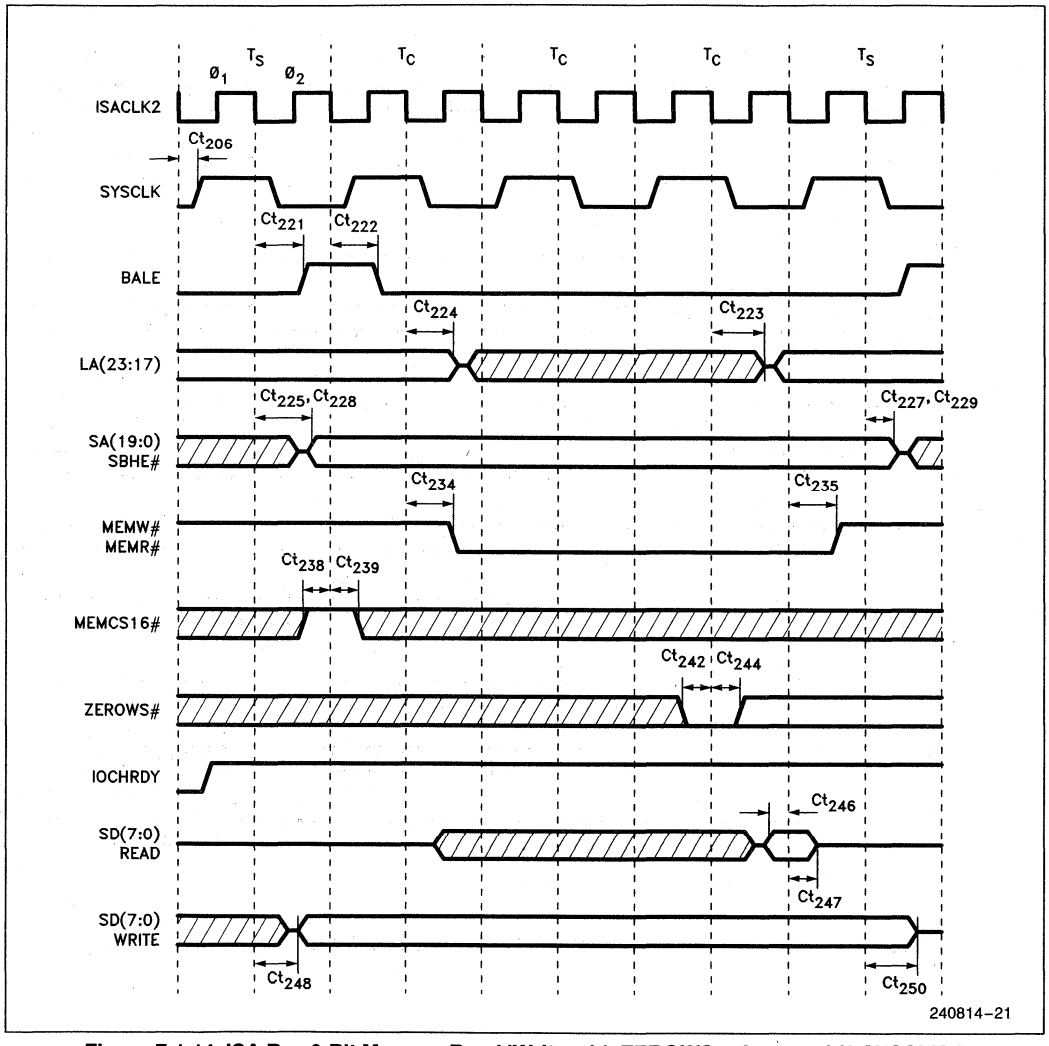


Figure 7.1.14. ISA Bus 8-Bit Memory Read/Write with ZEROWS# Asserted (3 SYSCLKs)

7.1 386™ SL CPU Timing Diagrams (Continued)

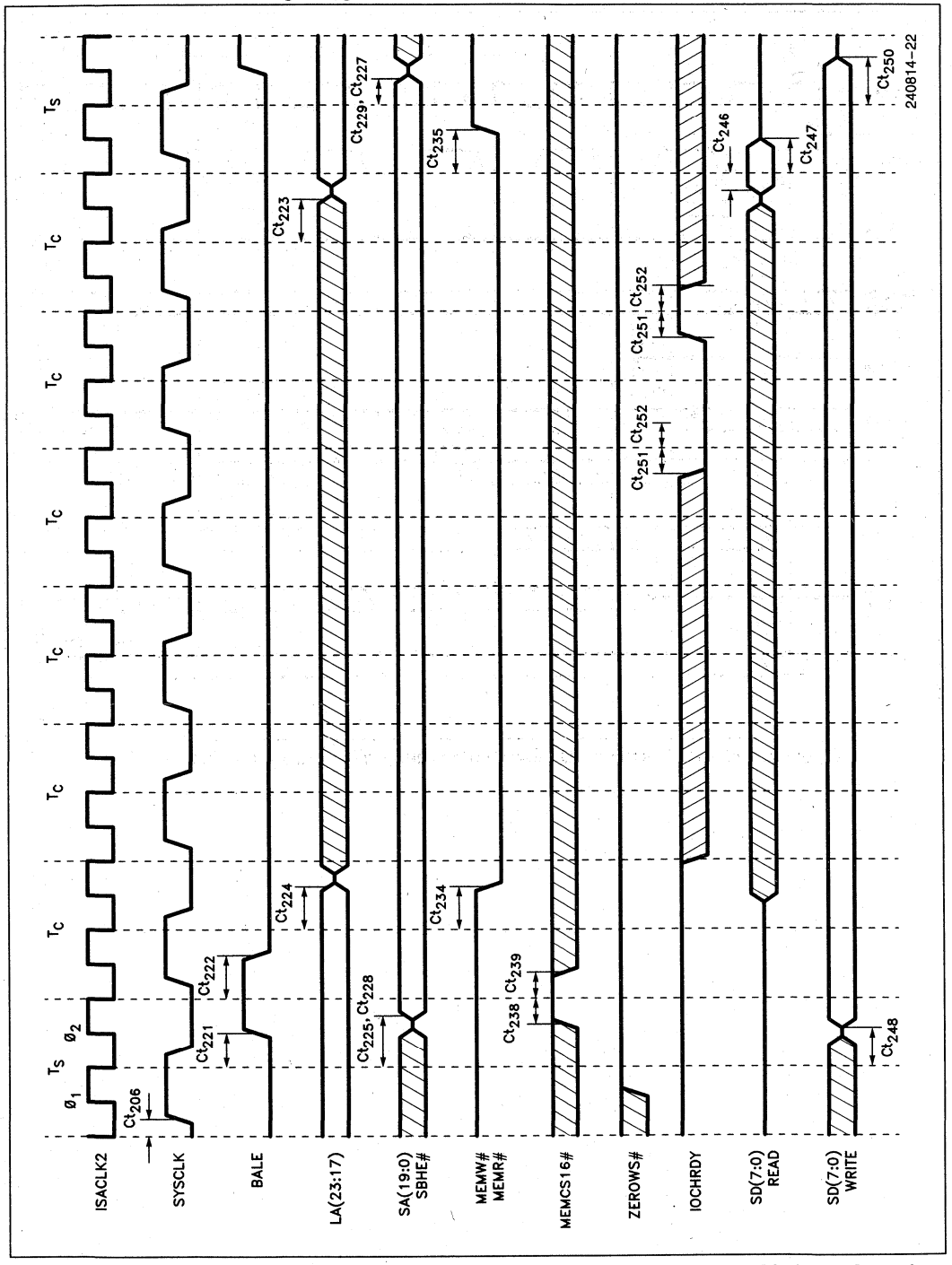


Figure 7.1.15. ISA Bus 8-Bit Memory Read/Write with IOCHRDY De-Asserted (Added Wait States)

7.1 386™ SL CPU Timing Diagrams (Continued)

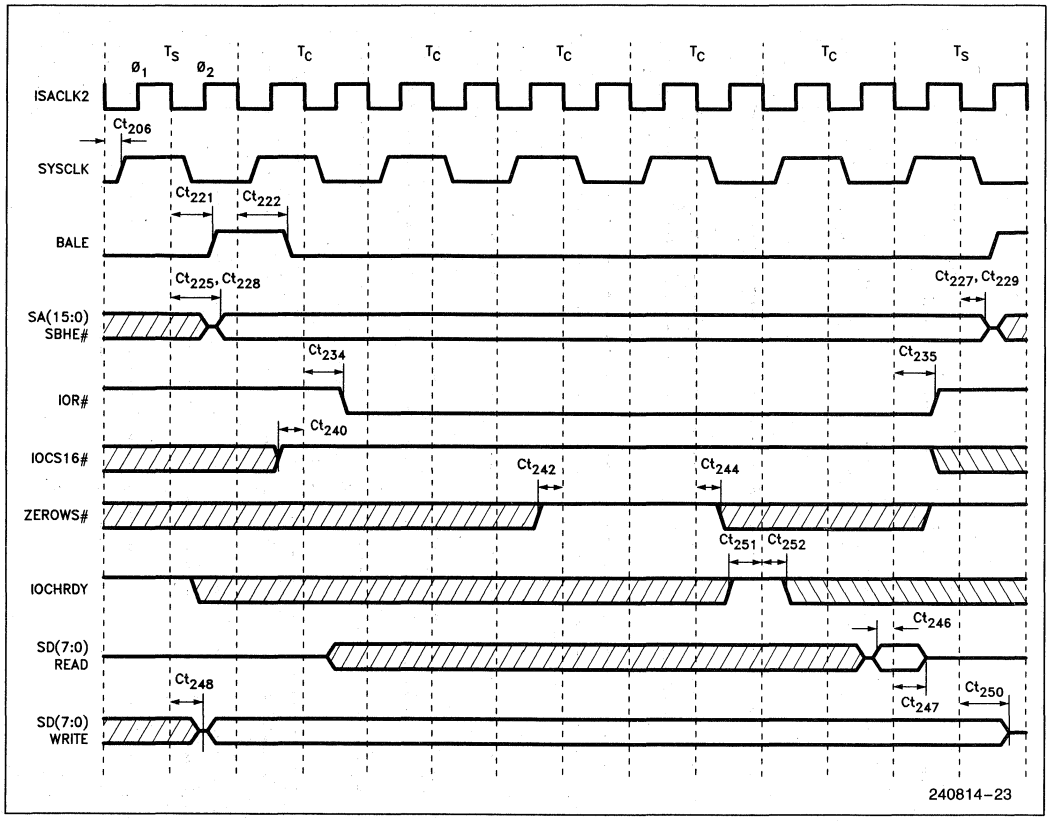


Figure 7.1.16. ISA Bus 8-Bit I/O Read/Write Standard ISA BUS Cycle (6 SYSCLKs)

7.1 386™ SL CPU Timing Diagrams (Continued)

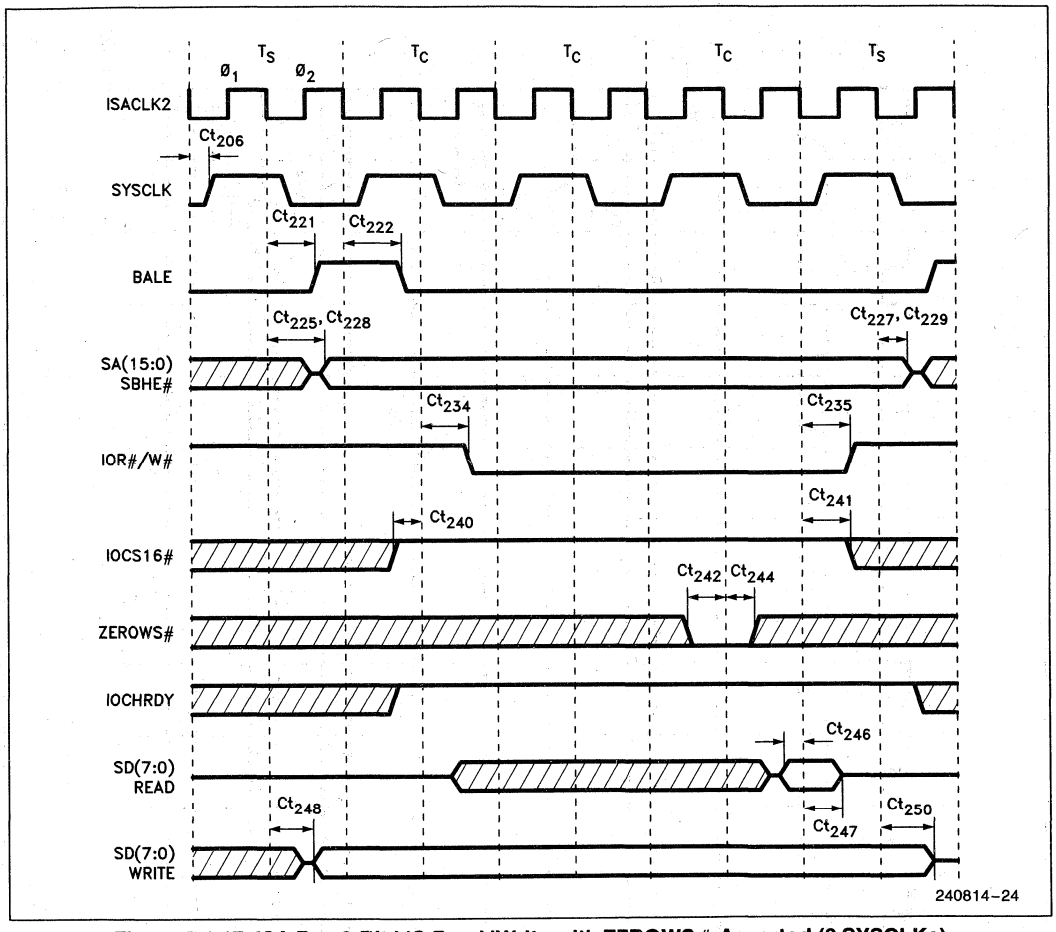


Figure 7.1.17. ISA Bus 8-Bit I/O Read/Write with ZEROWS# Asserted (3 SYSCLKs)

5

7.1 386™ SL CPU Timing Diagrams (Continued)

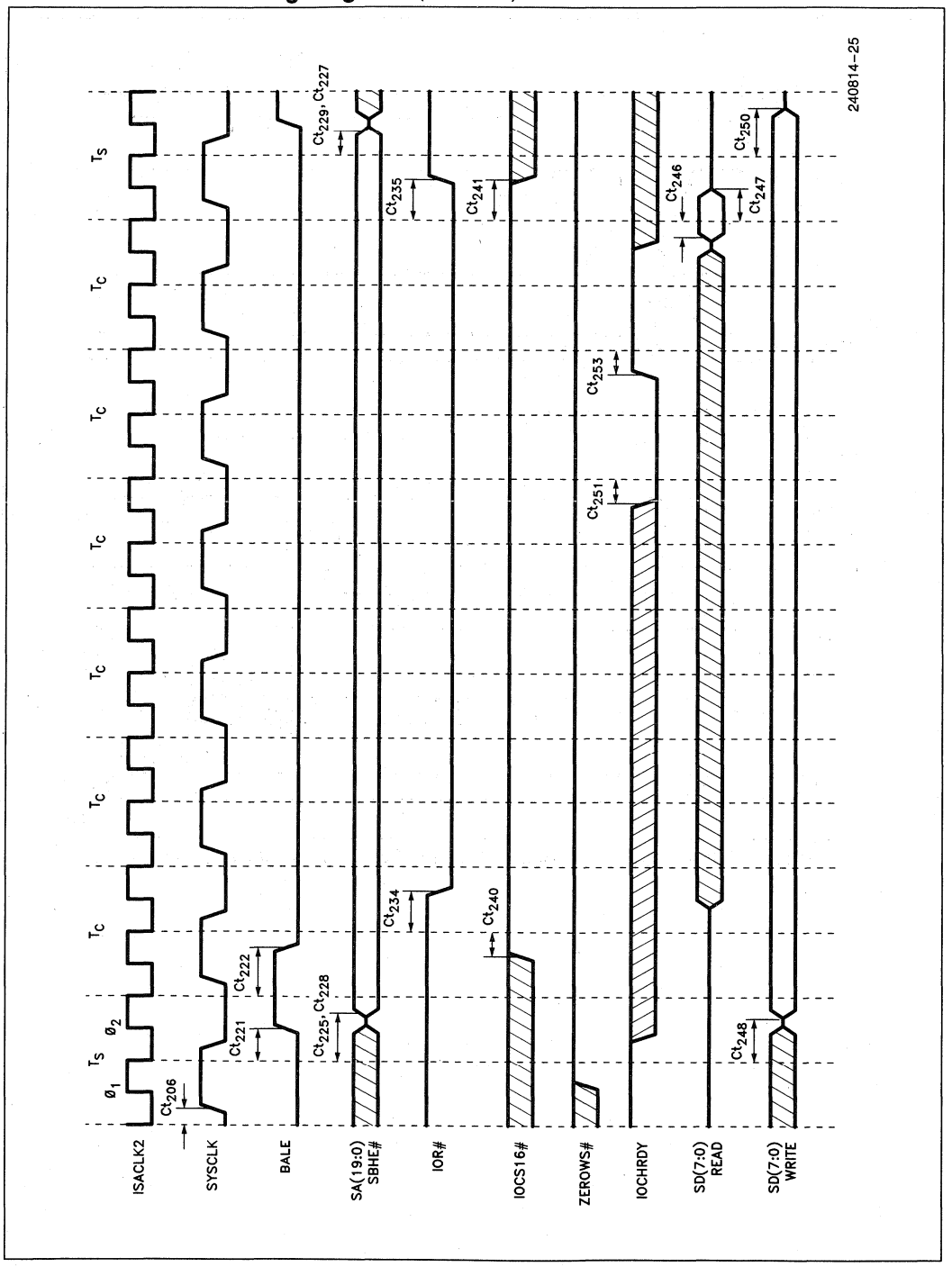


Figure 7.1.18. ISA Bus 8-Bit I/O Read/Write with IOCHRDY De-Asserted (Added Wait States)

7.1 386™ SL CPU Timing Diagrams (Continued)

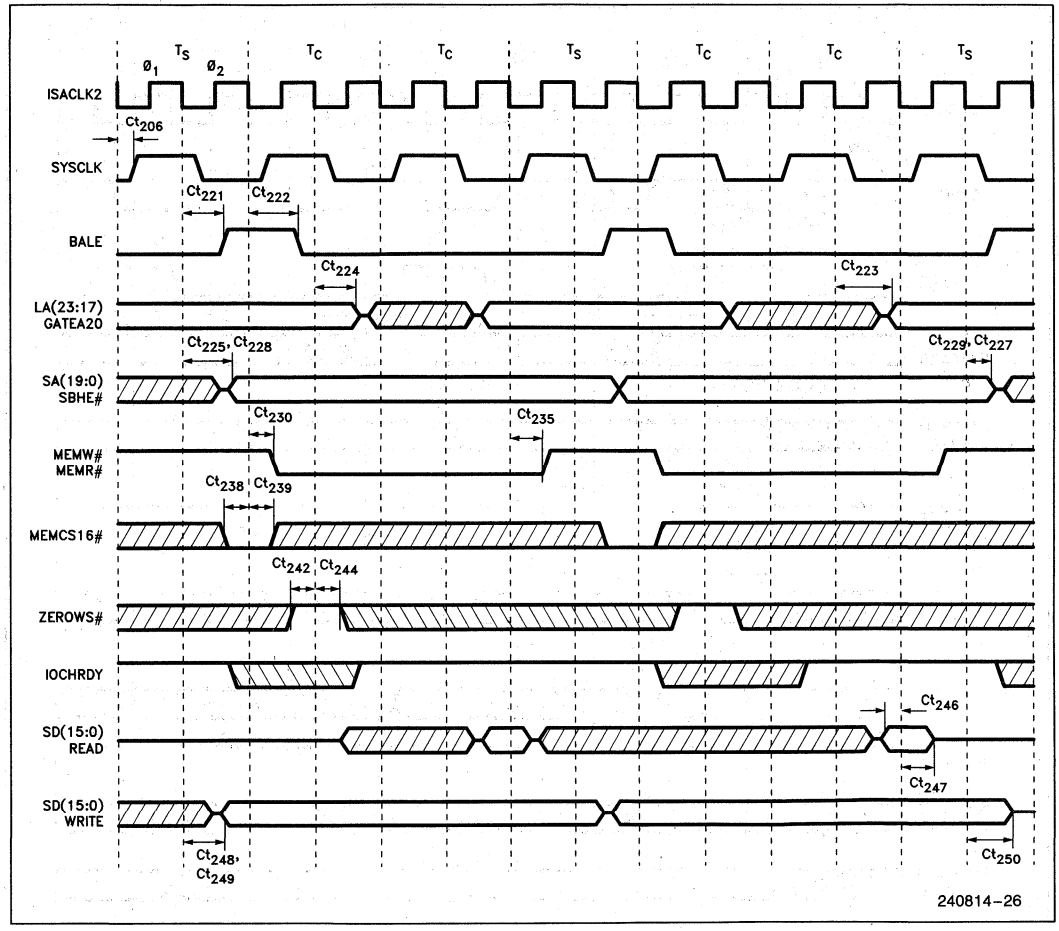


Figure 7.1.19. ISA Bus 16-Bit Memory Read/Write Standard ISA BUS Cycle (3 SYSCLKs)

5

7.1 386™ SL CPU Timing Diagrams (Continued)

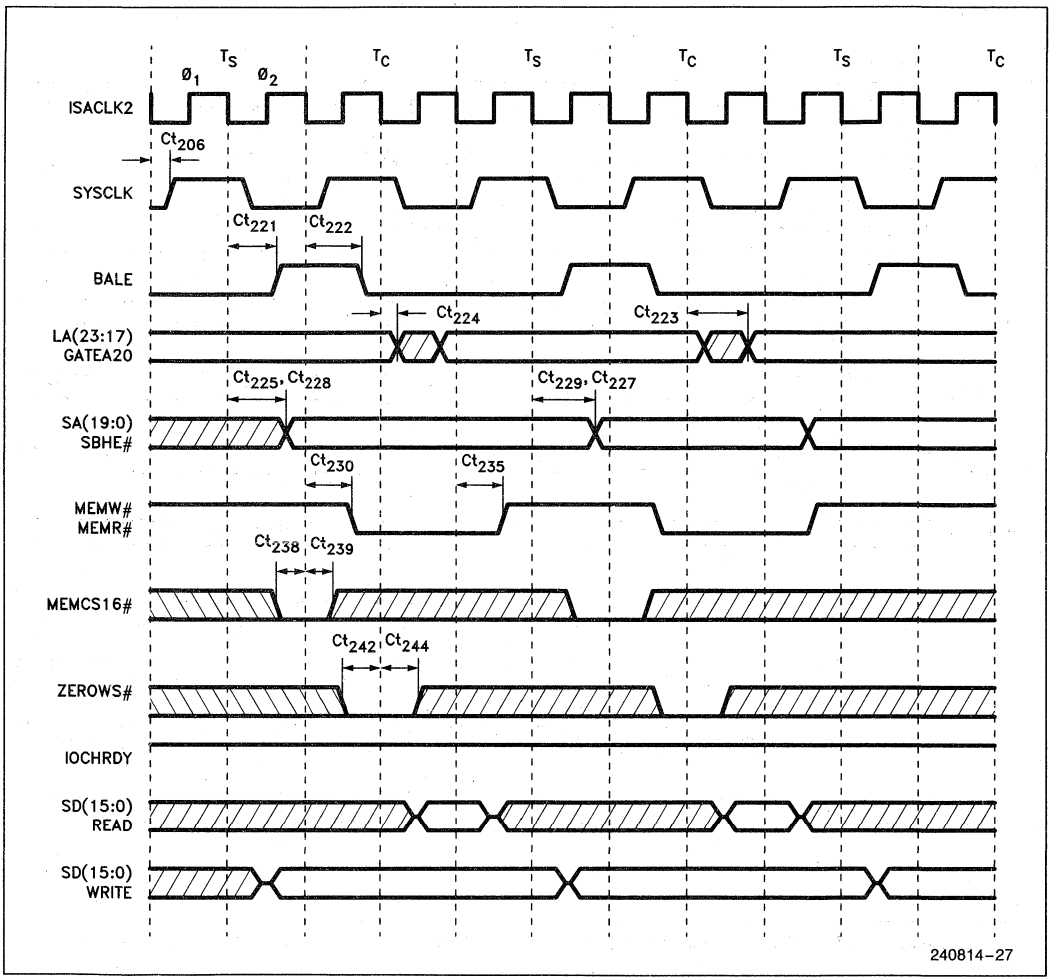


Figure 7.1.20. ISA Bus 16-Bit Memory Read/Write with ZEROWS# Asserted (2 SYSCLKs)

7.1 386™ SL CPU Timing Diagrams (Continued)

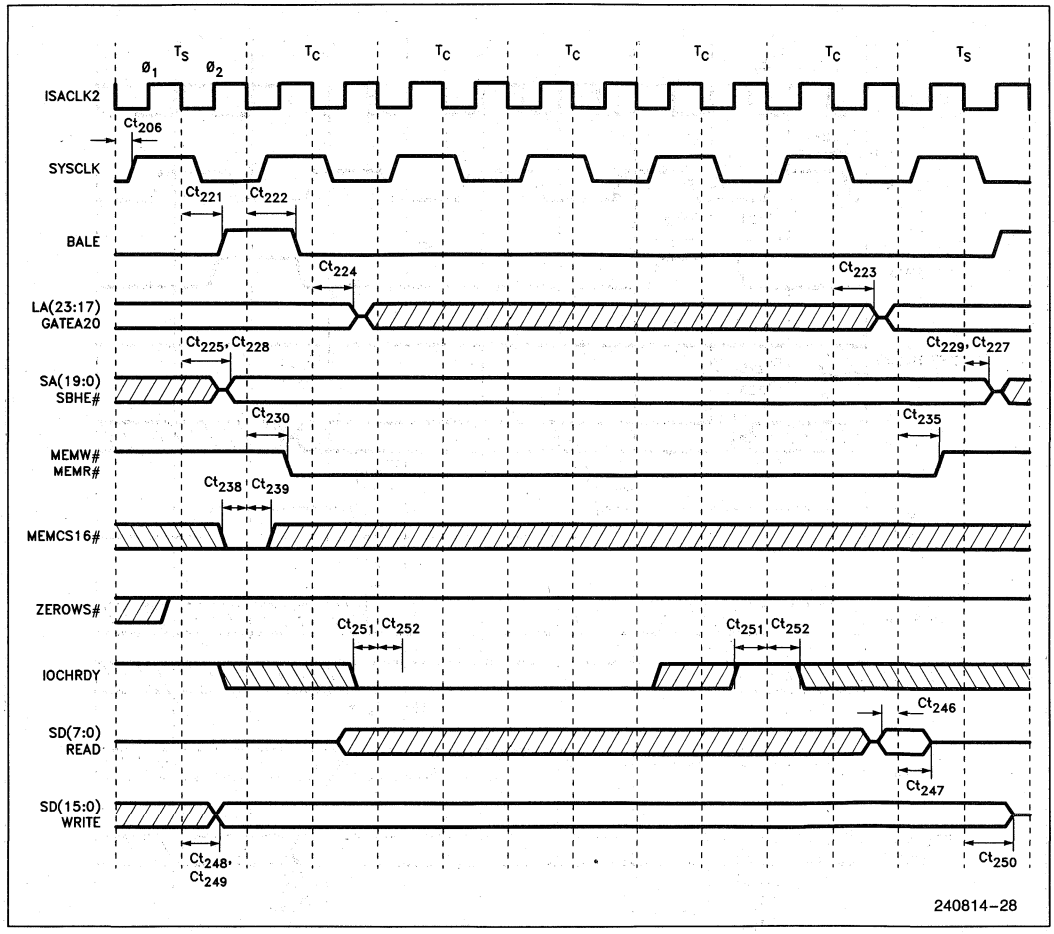


Figure 7.1.21. ISA Bus 16-Bit Memory Read/Write with IOCHRDY De-Asserted (Added Wait States)

7.1 386™ SL CPU Timing Diagrams (Continued)

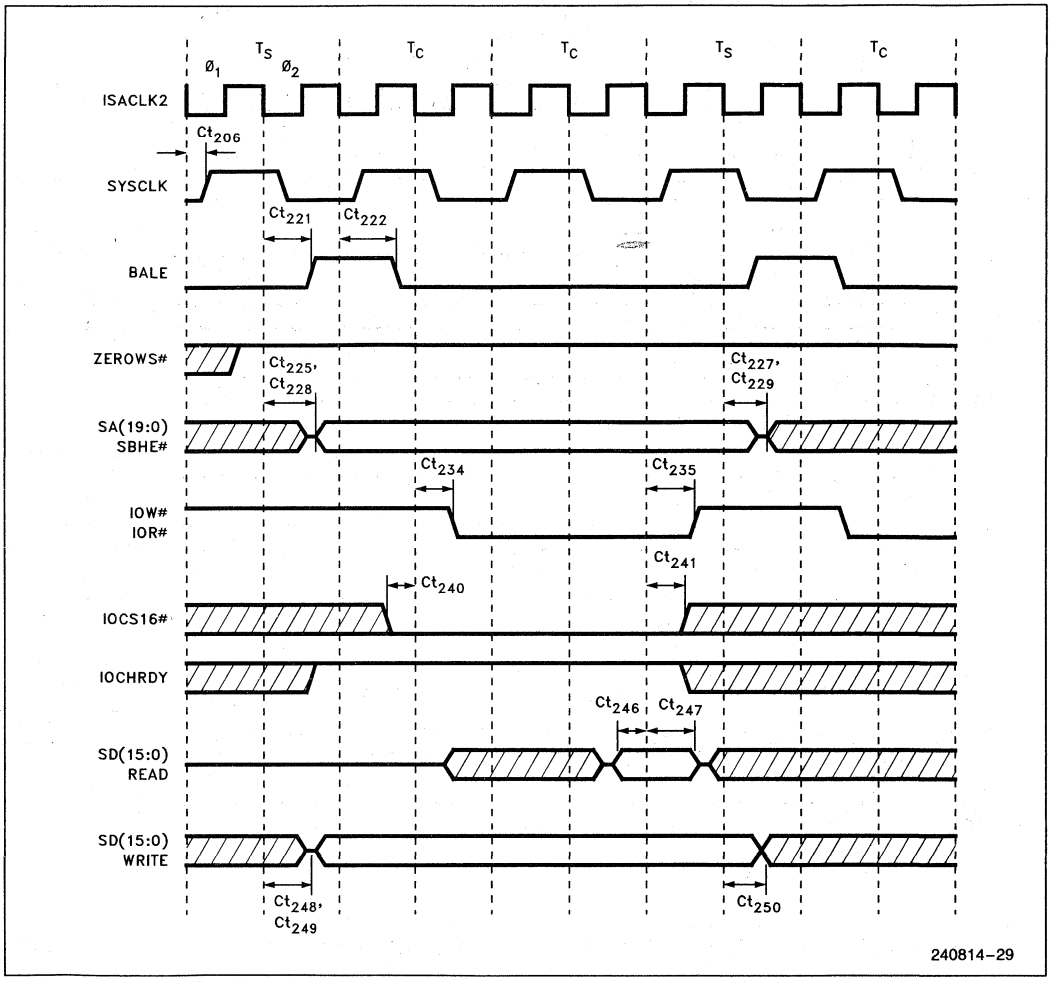


Figure 7.1.22. ISA Bus 16-Bit I/O Read/Write Standard ISA BUS Cycle (3 SYSCLKs)

7.1 386™ SL CPU Timing Diagrams (Continued)

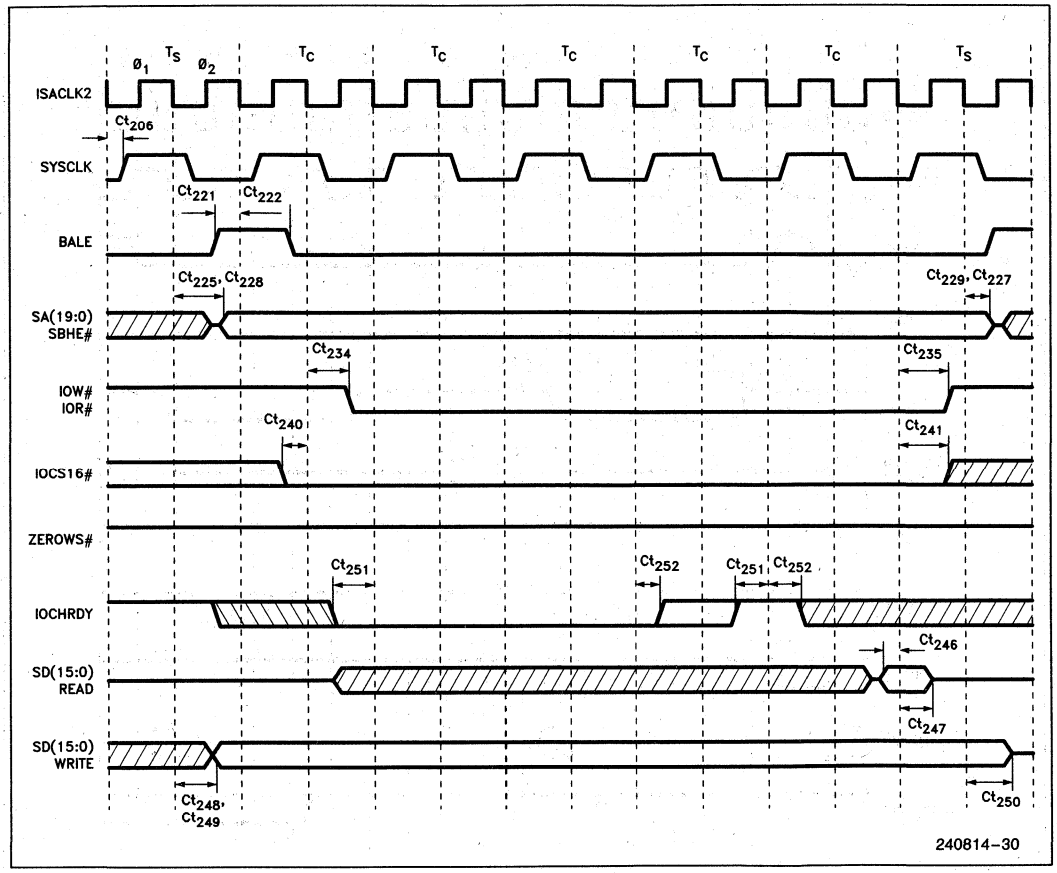


Figure 7.1.23. ISA Bus 16-Bit I/O Read/Write with IOCHRDY De-Asserted (Added Wait States)

5

7.1 386™ SL CPU Timing Diagrams (Continued)

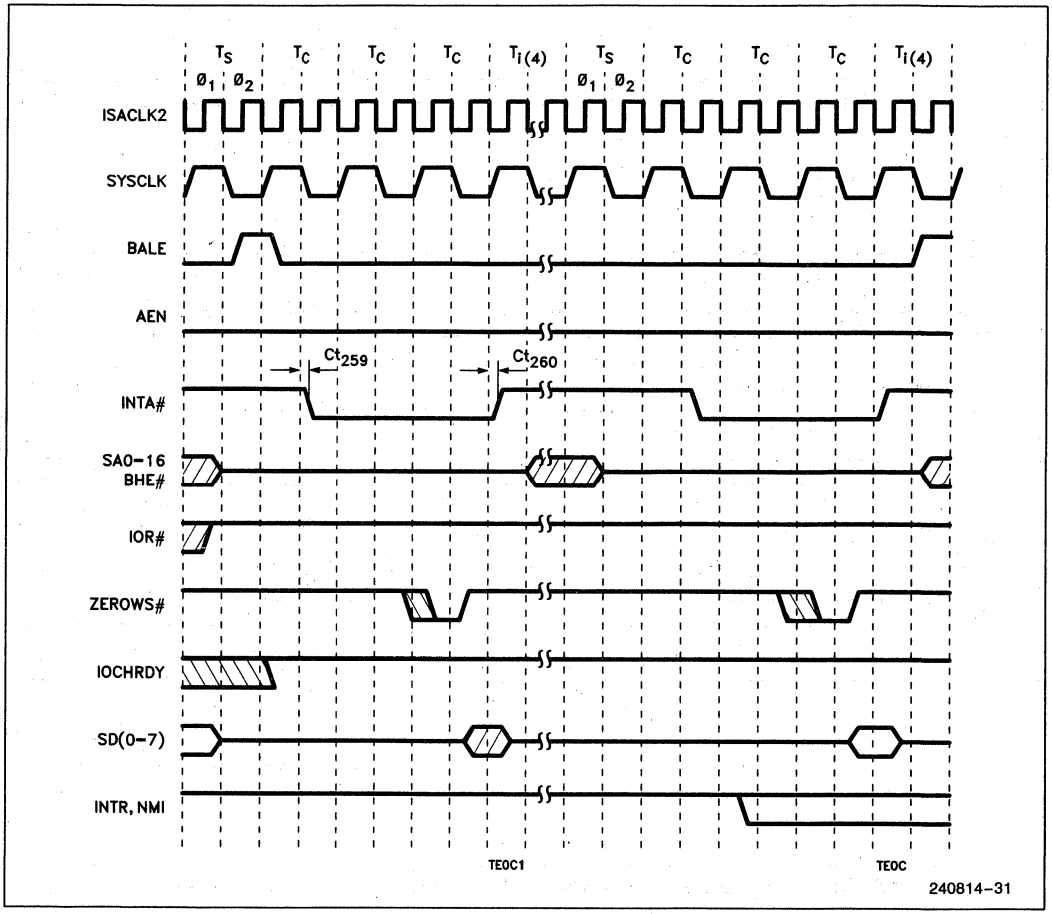


Figure 7.1.24. ISA Bus Interrupt Acknowledge Bus Cycle

7.1 386™ SL CPU Timing Diagrams (Continued)

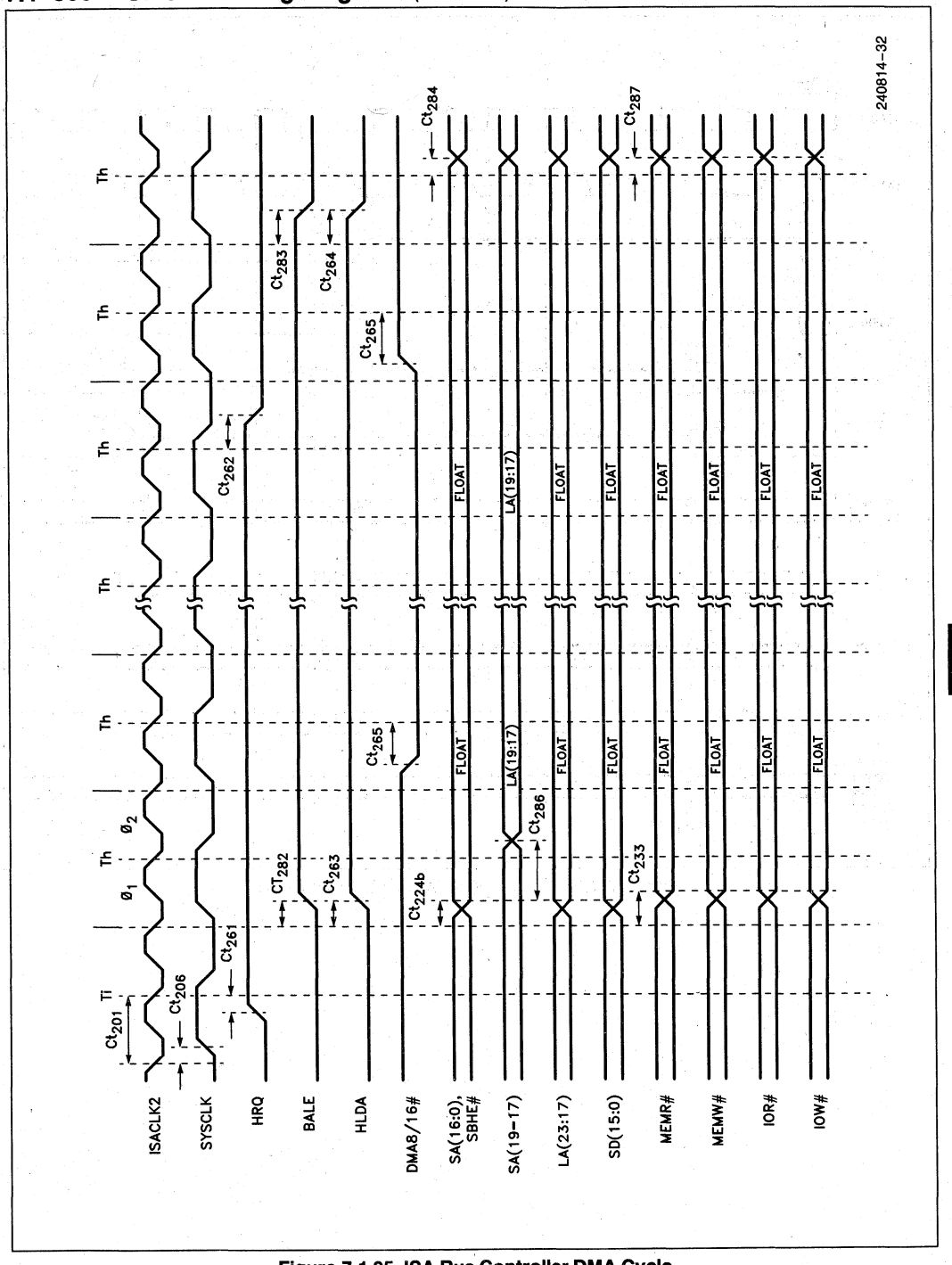


Figure 7.1.25. ISA Bus Controller DMA Cycle

7.1 386™ SL CPU Timing Diagrams (Continued)

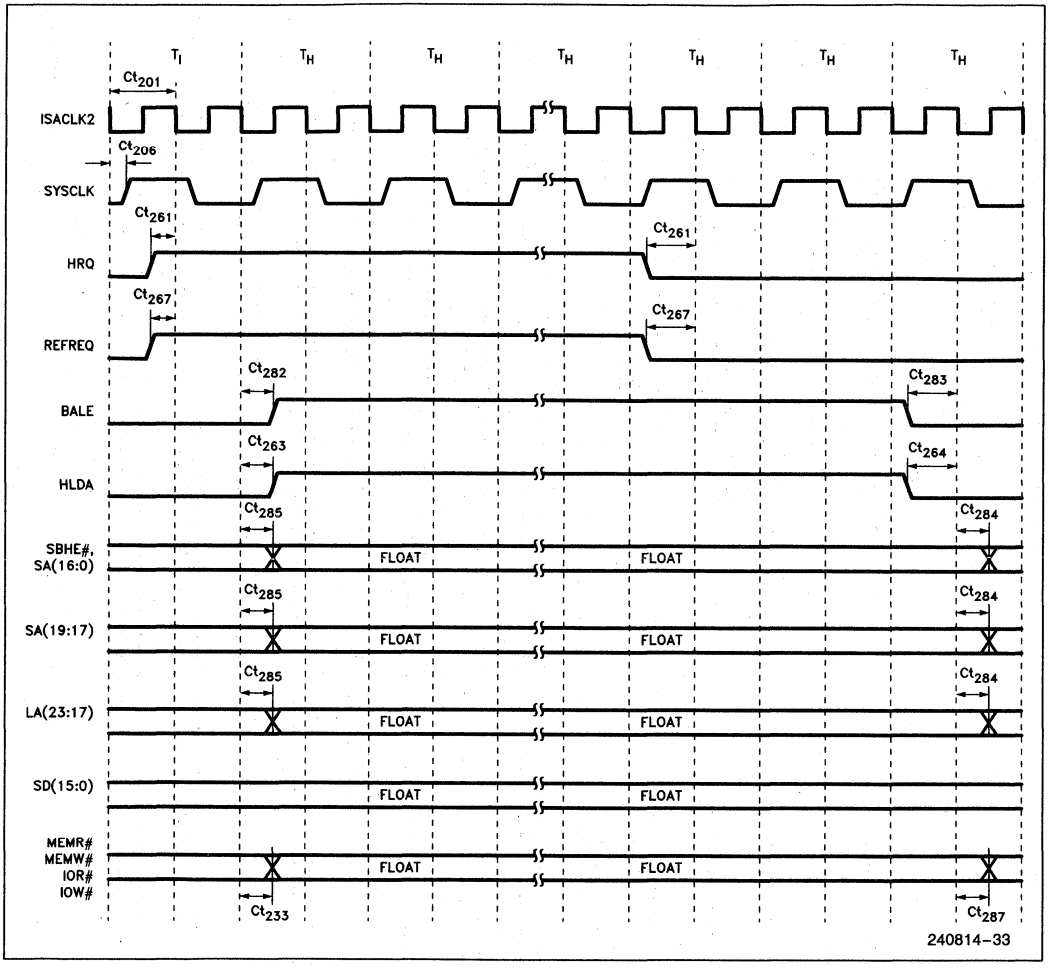
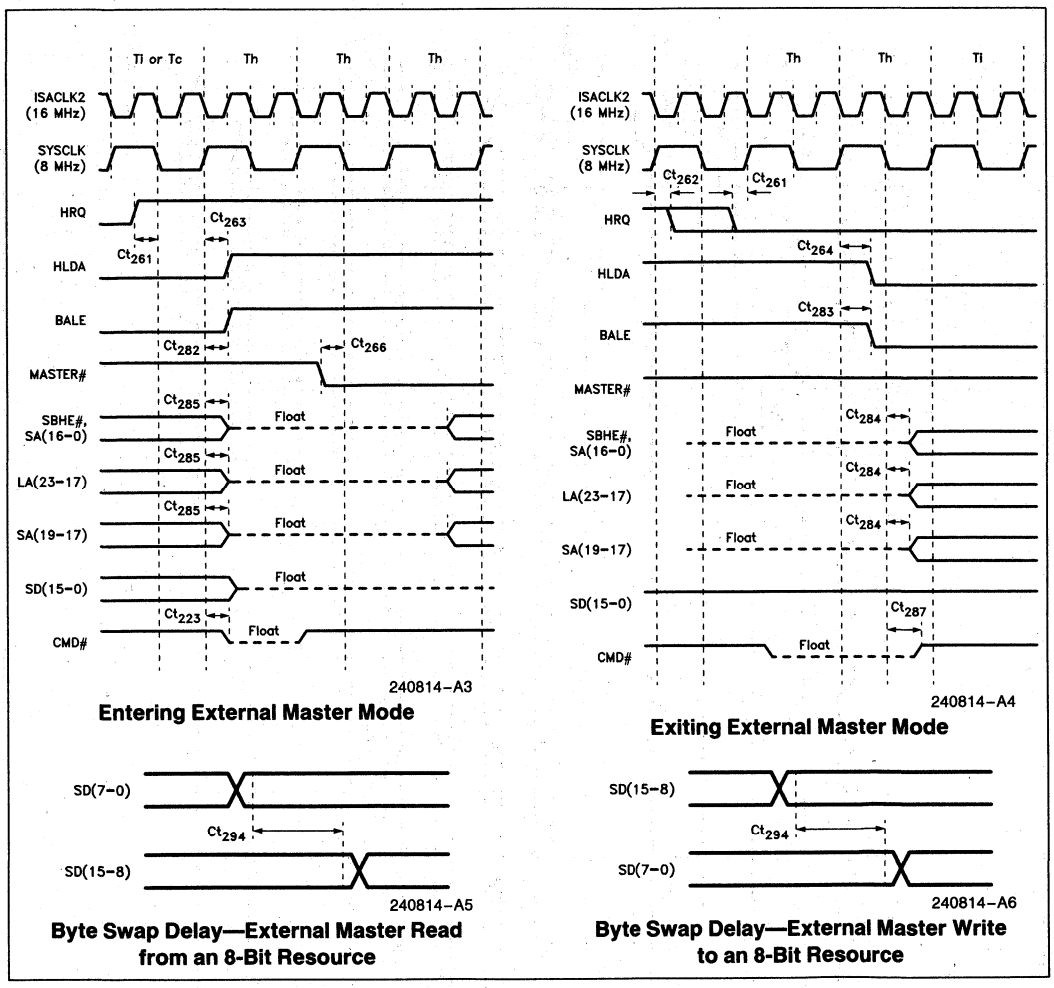


Figure 7.1.26. ISA Bus Controller Refresh Cycle

7.1 386™ SL CPU Timing Diagrams (Continued)



Entering External Master Mode

Exiting External Master Mode

Byte Swap Delay—External Master Read from an 8-Bit Resource

Byte Swap Delay—External Master Write to an 8-Bit Resource

Figure 7.1.27. ISA Bus External Bus Master

5

7.1 386™ SL CPU Timing Diagrams (Continued)

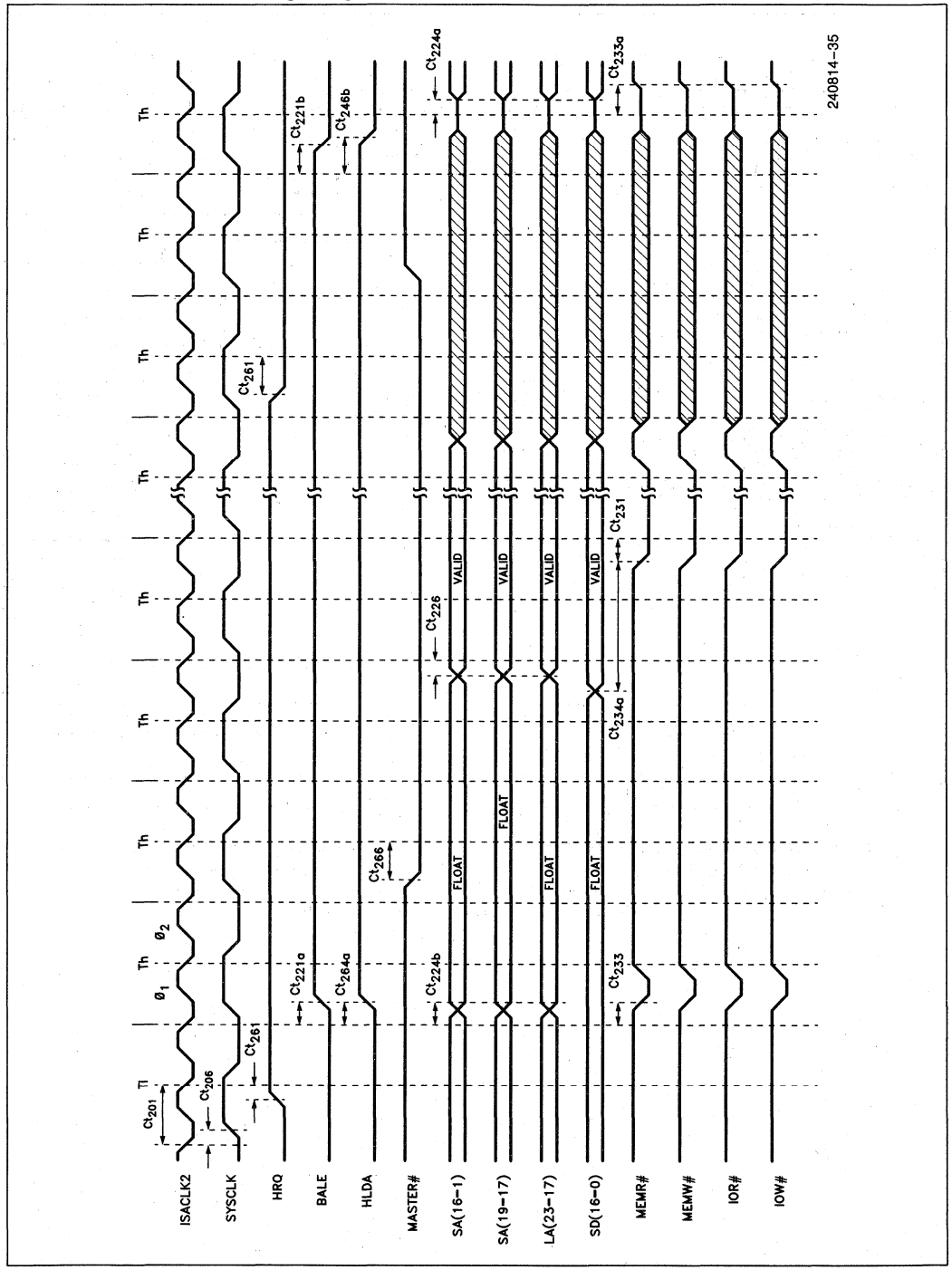
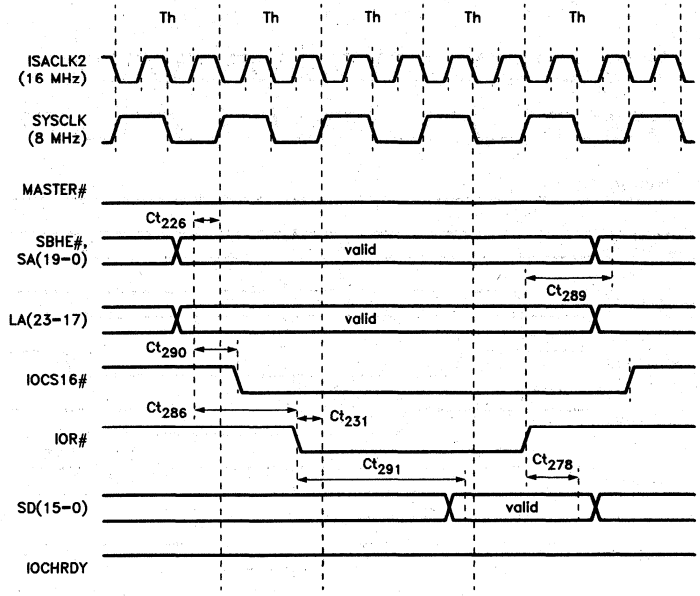
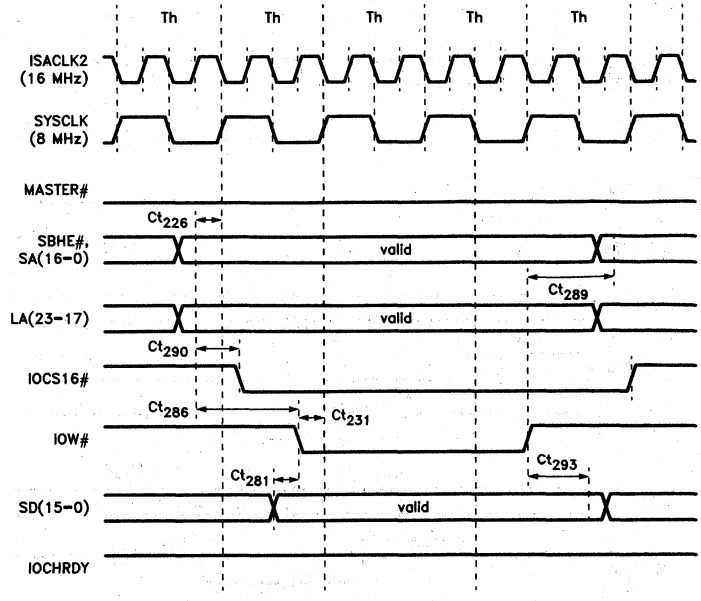


Figure 7.1.28. ISA Bus External Bus Master to Off-Board I/O Ports (No Byte-Swapping)

7.1 386™ SL CPU Timing Diagrams (Continued)



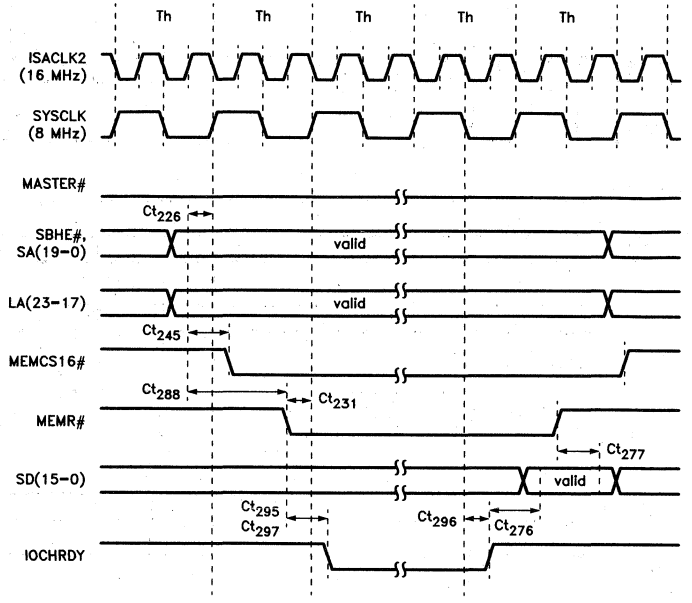
External Master Read from On-Board I/O Ports



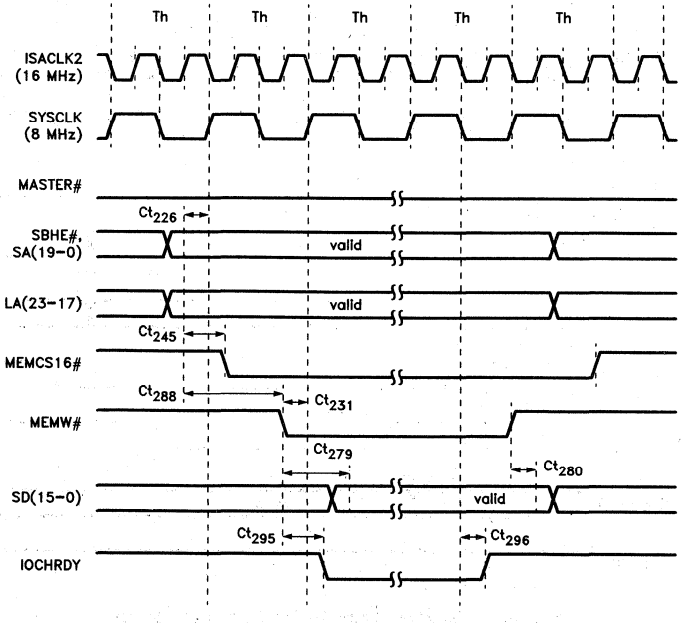
External Master Write to On-Board I/O Ports

Figure 7.1.29a. ISA Bus External Bus Master to On-Board I/O Ports (Read/Write)

7.1 386™ SL CPU Timing Diagrams (Continued)



External Master Read from On-Board Memory



External Master Write to On-Board Memory

Figure 7.1.29b. ISA Bus External Bus Master Accesses to On-Board Memory

7.1 386™ SL CPU Timing Diagrams (Continued)

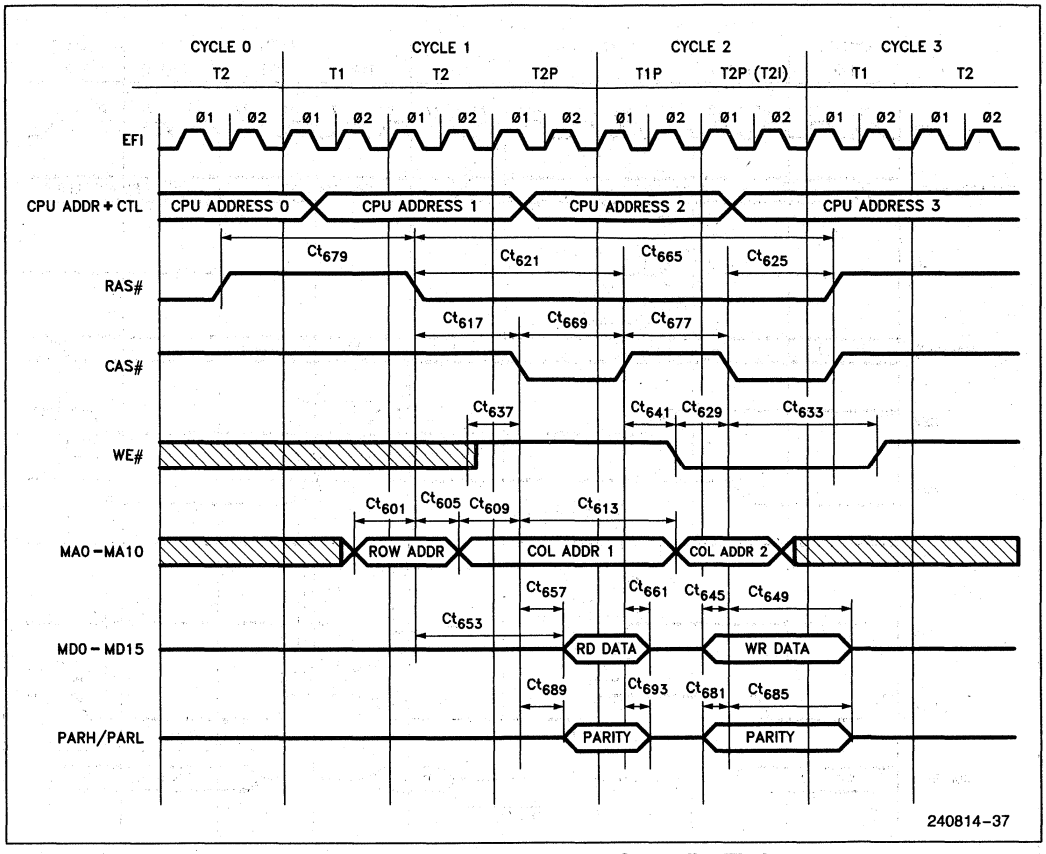


Figure 7.1.30. 386™ SL CPU Memory Controller Timings (DRAM F1 Mode Timing Parameters)

5

7.1 386™ SL CPU Timing Diagrams (Continued)

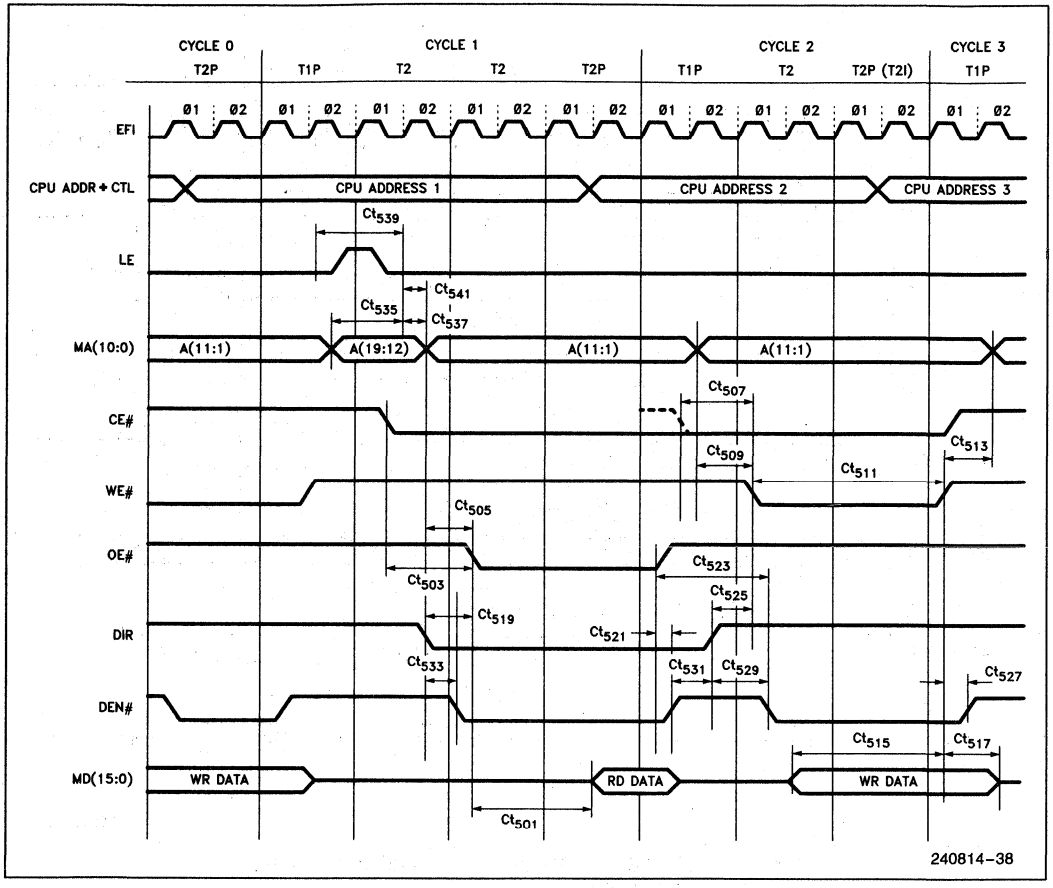


Figure 7.1.31. 386™ SL CPU Memory Controller Timings (SRAM Mode Timing Parameters; 2 Wait States)

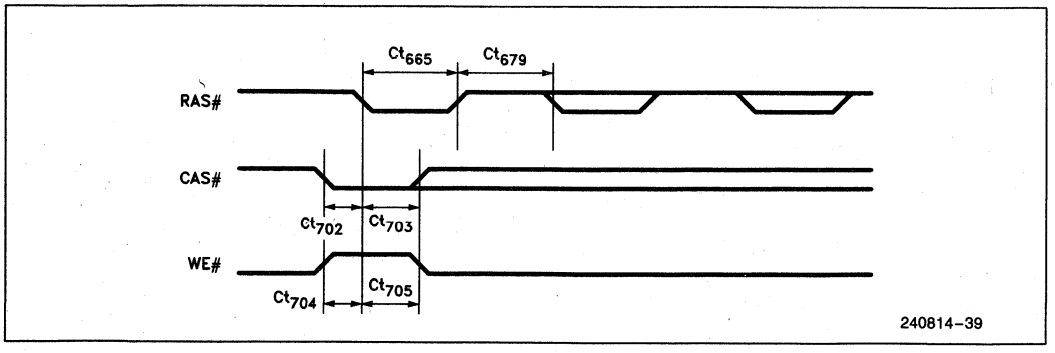


Figure 7.1.32. 386™ SL CPU Memory Controller Timings (CAS# before RAS# Refresh Timings)

7.1 386™ SL CPU Timing Diagrams (Continued)

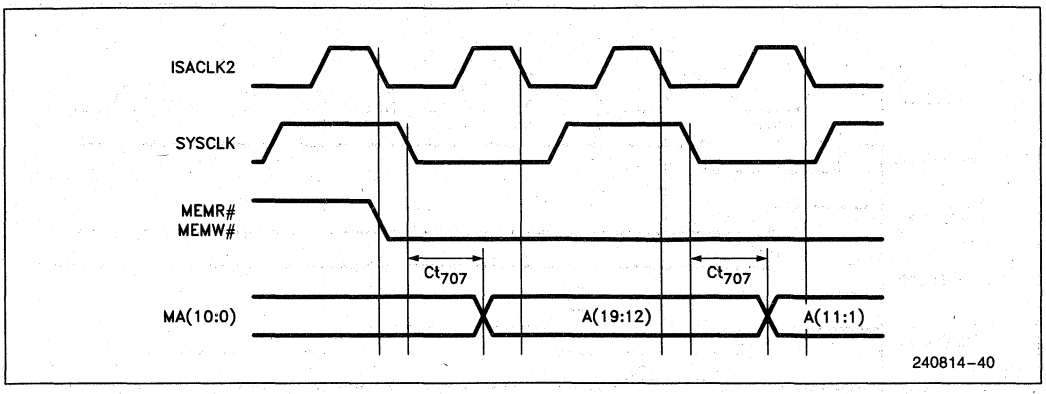


Figure 7.1.33. REFRESH, DMA/MASTER Timing Diagrams (Address Active Delay from SYSCLK)

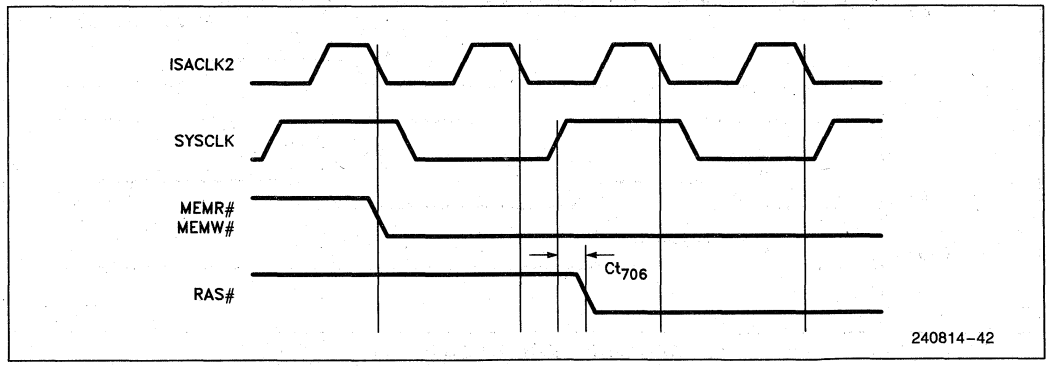


Figure 7.1.34. REFRESH, DMA/MASTER Timing Diagrams (RAS# Active Delay from SYSCLK)

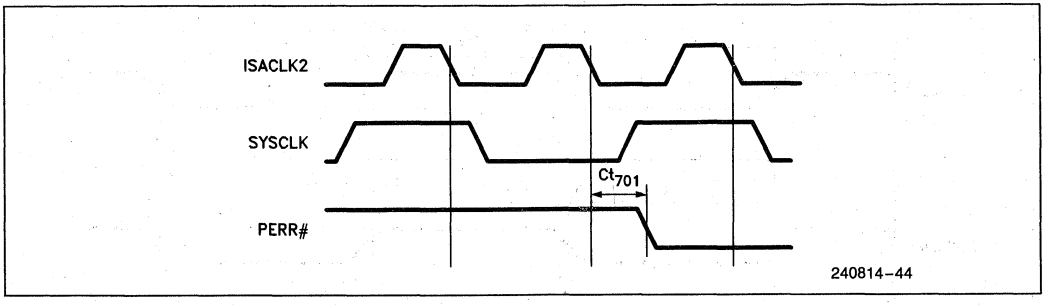


Figure 7.1.35. PERROR Timing Diagram (PERR# Active Delay from SYSCLK)

7.2 82360SL Timing Diagrams

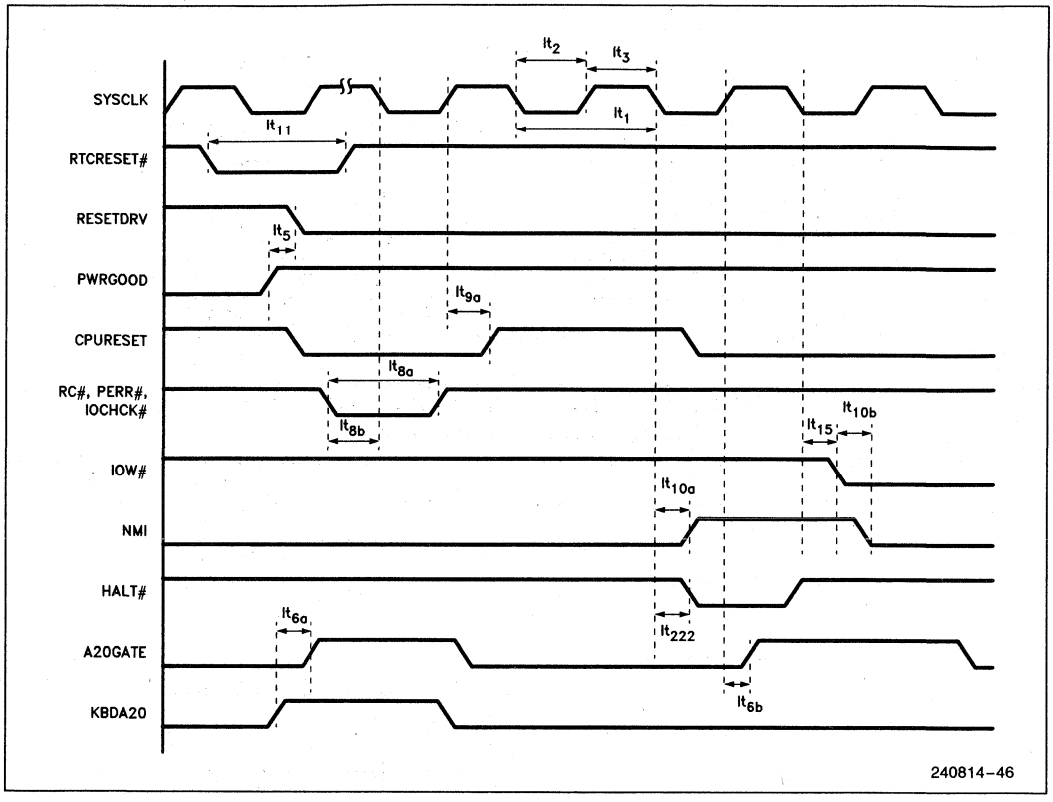


Figure 7.2.1. CPURESET, NMI, A20GATE and RC# Timings

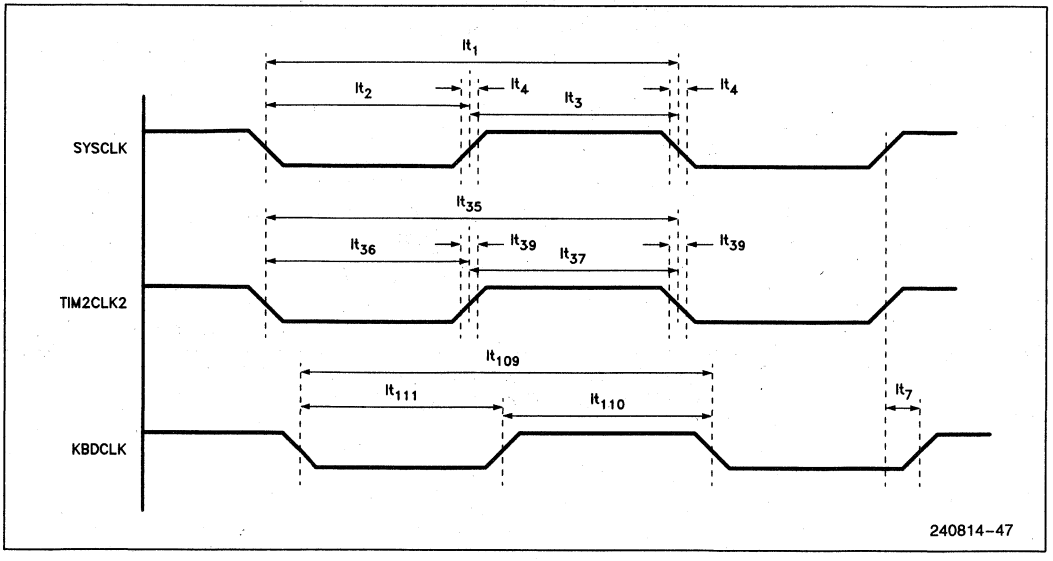


Figure 7.2.2. Clock Timings

7.2 82360SL Timing Diagrams (Continued)

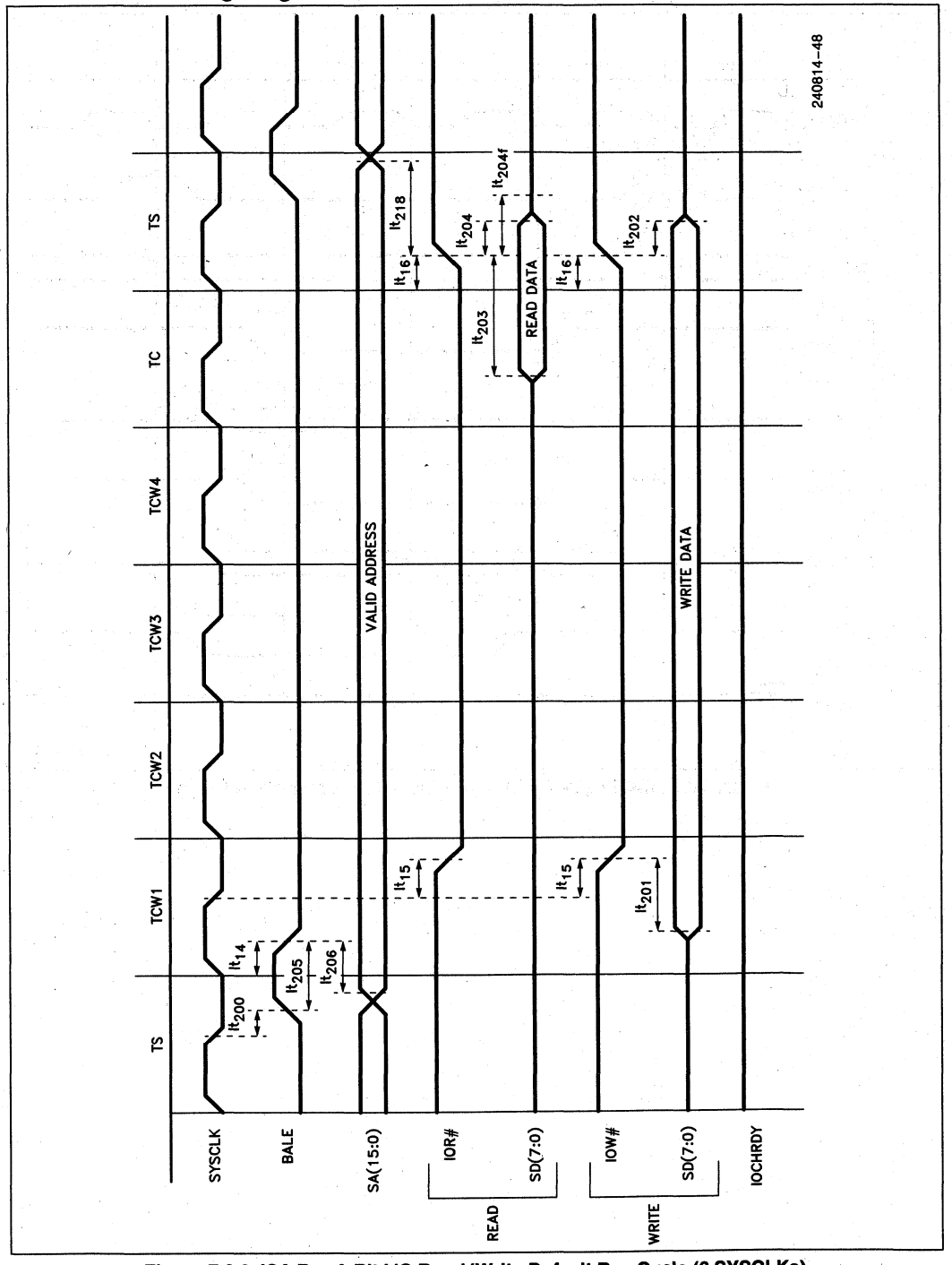


Figure 7.2.3. ISA Bus 8-Bit I/O Read/Write Default Bus Cycle (6 SYSCLKs)

7.2 82360SL Timing Diagrams (Continued)

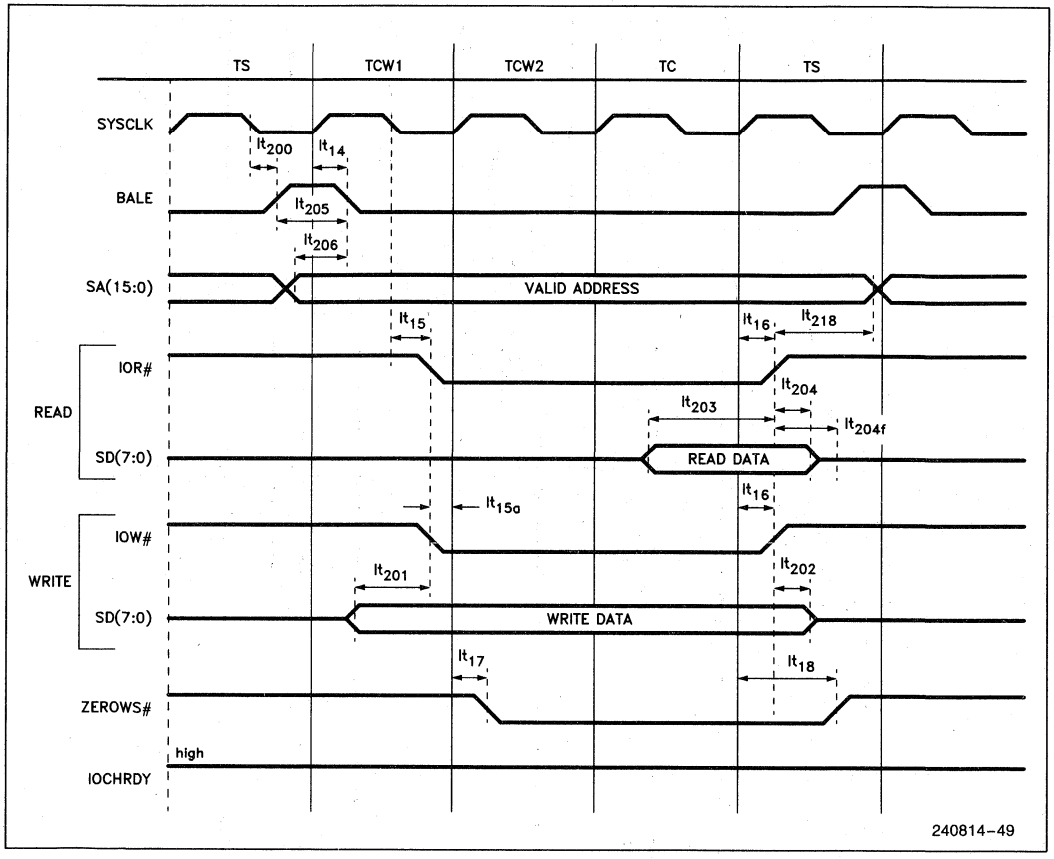


Figure 7.2.4. ISA Bus 8-Bit I/O Read/Write Compressed Bus Cycle

7.2 82360SL Timing Diagrams (Continued)

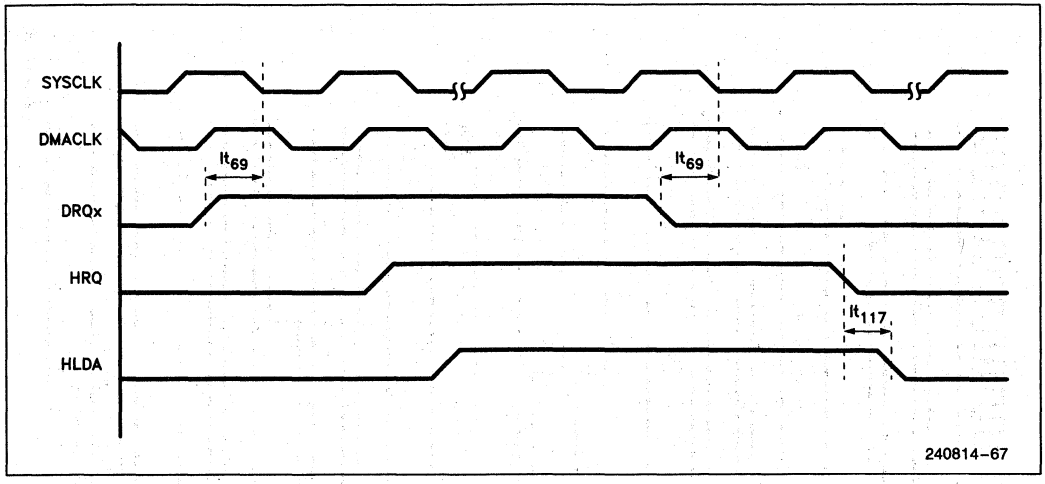


Figure 7.2.5. DMA Controller Timings

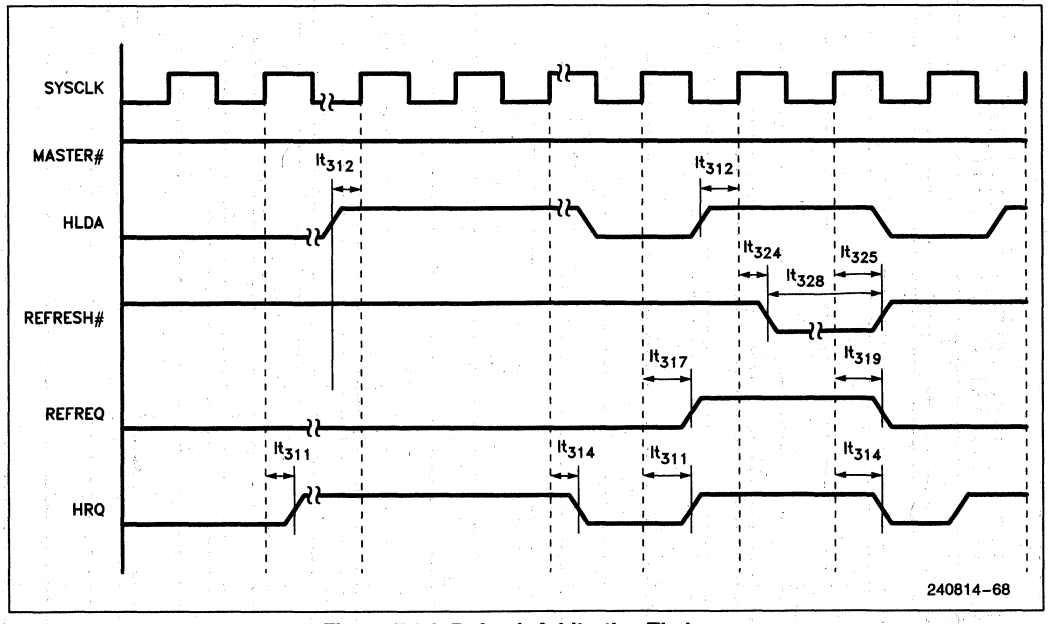


Figure 7.2.6. Refresh Arbitration Timings

7.2 82360SL Timing Diagrams (Continued)

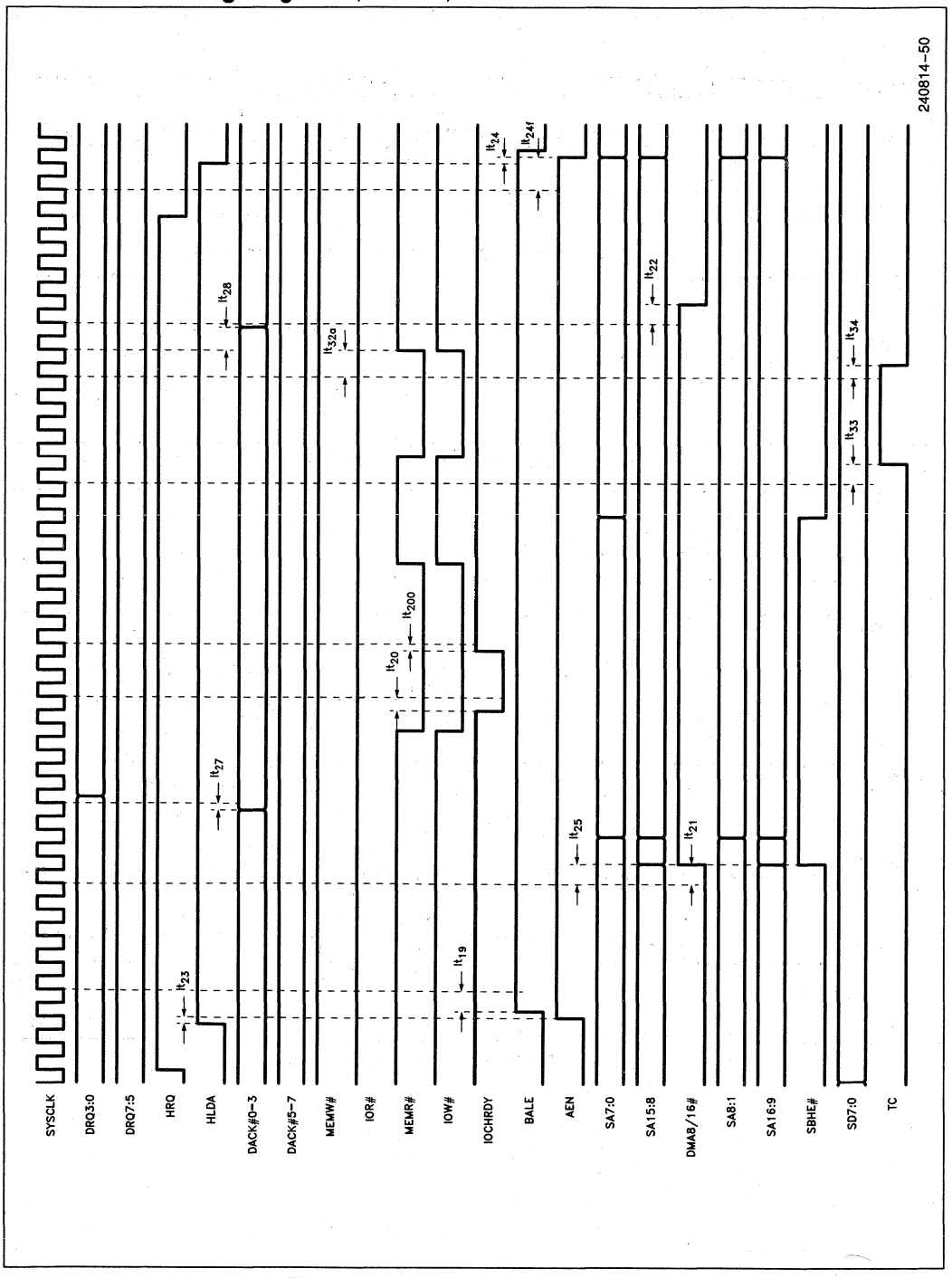


Figure 7.2.7. DMA Memory Write Timings

7.2 82360SL Timing Diagrams (Continued)

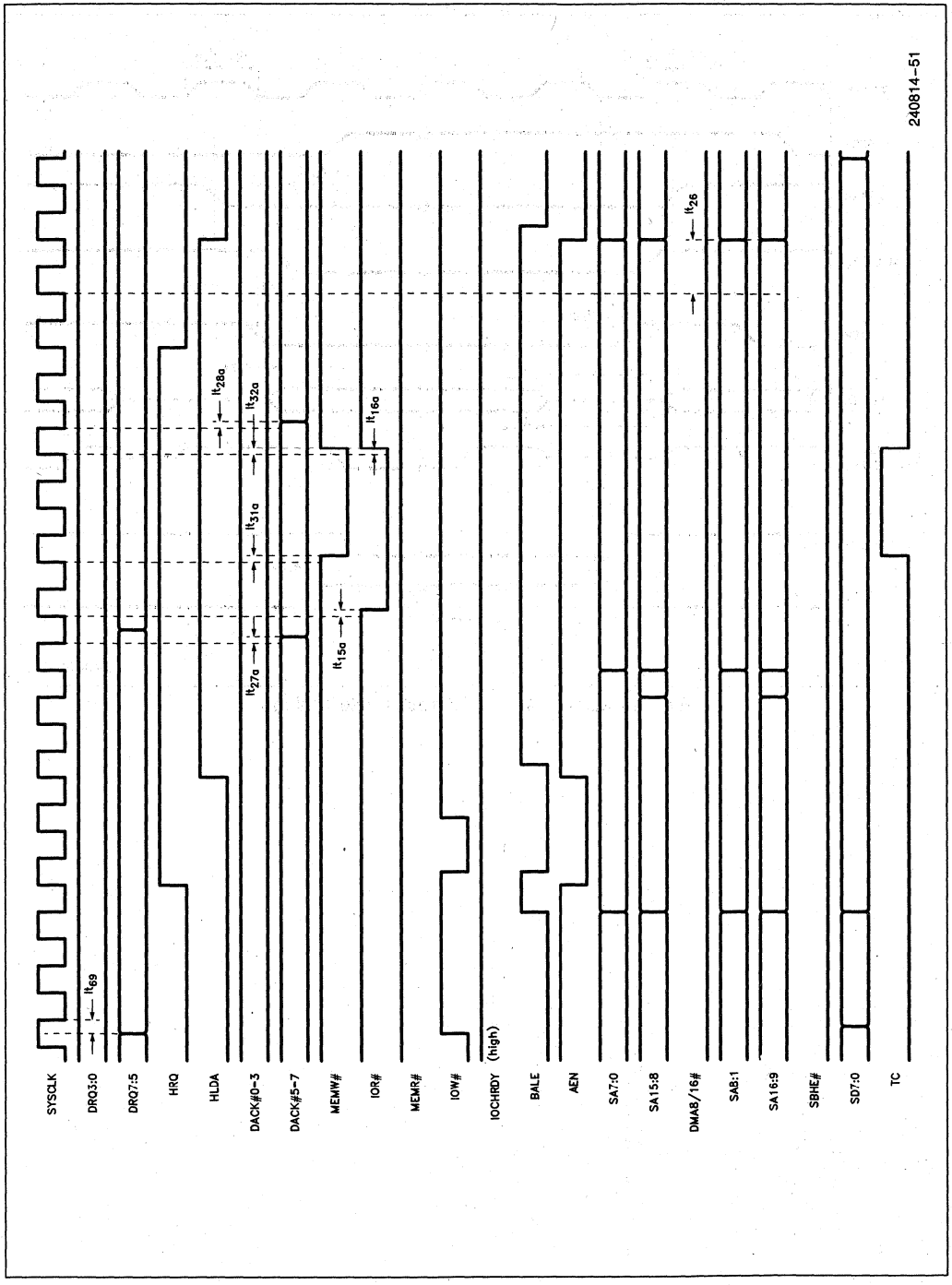


Figure 7.2.8. DMA Memory Read Timings

7.2 82360SL Timing Diagrams (Continued)

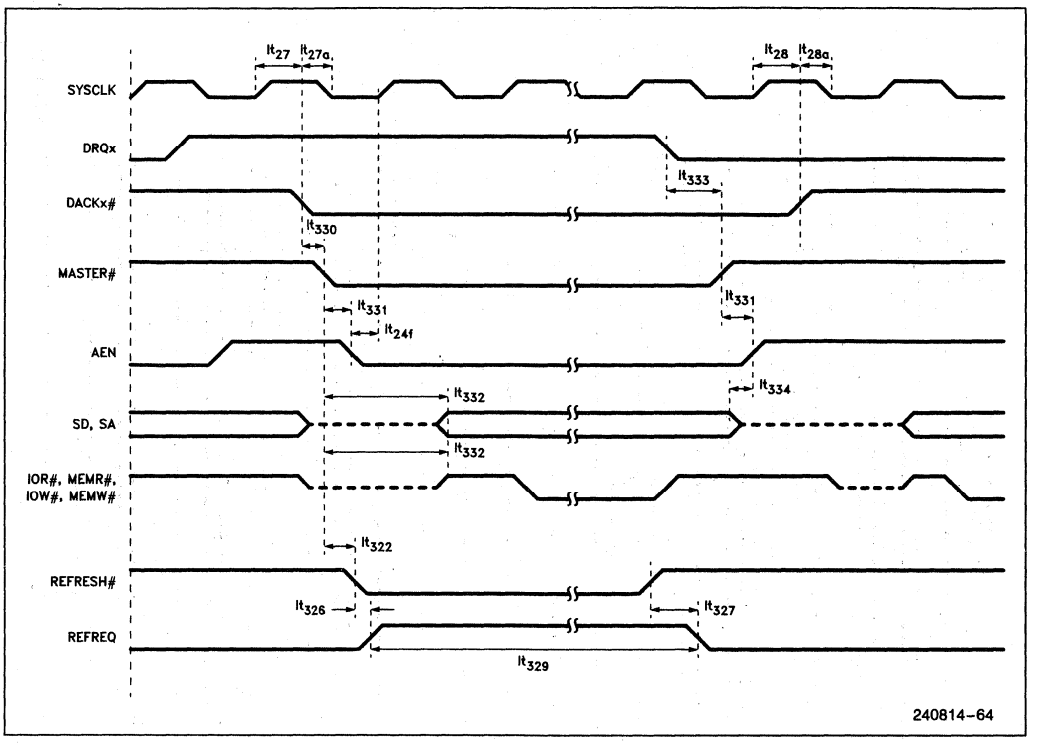


Figure 7.2.9. Bus Master Refresh Cycle Timings

7.2 82360SL Timing Diagrams (Continued)

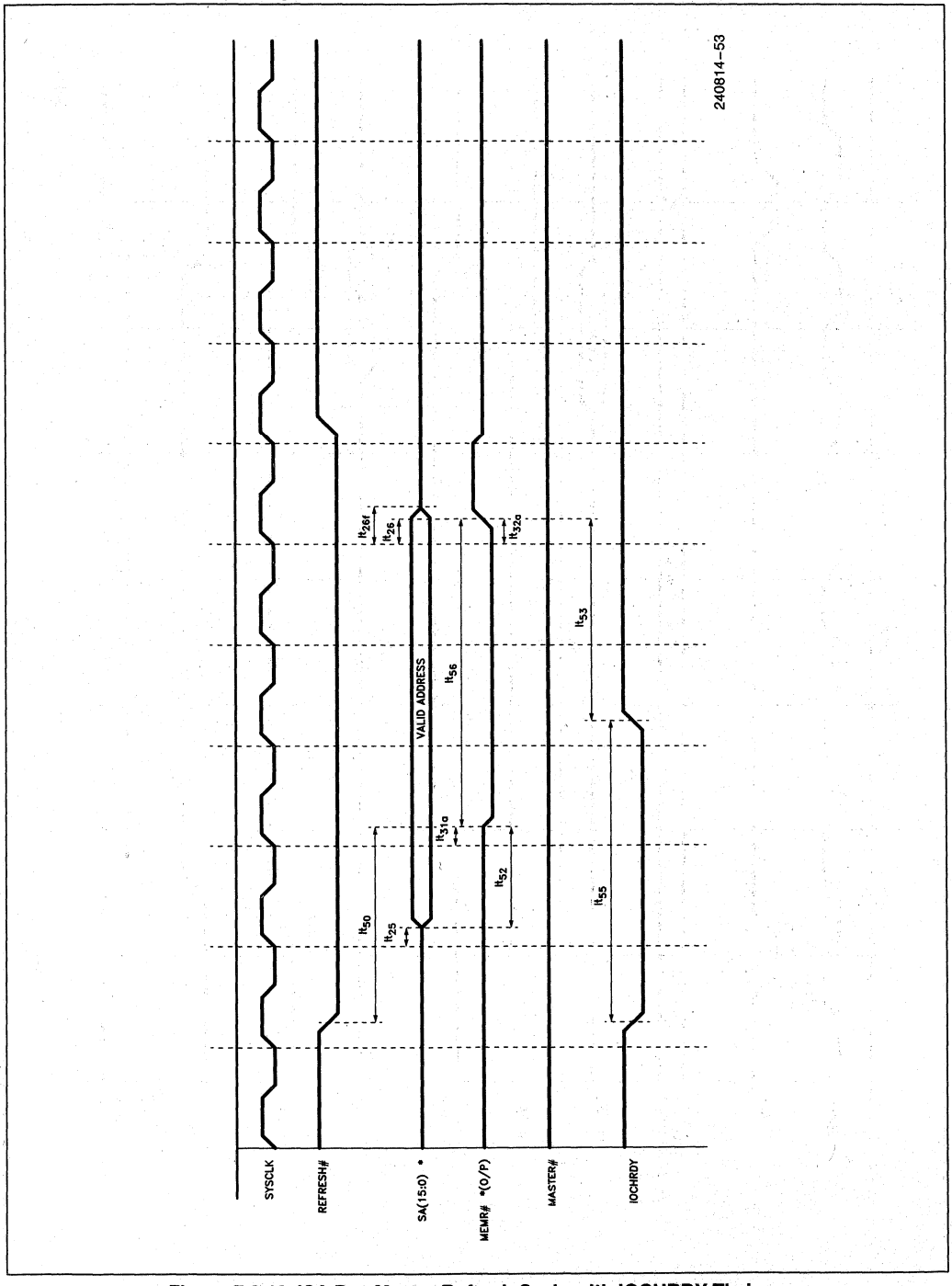


Figure 7.2.10. ISA Bus Master Refresh Cycle with IOCHRDY Timings

7.2 82360SL Timing Diagrams (Continued)

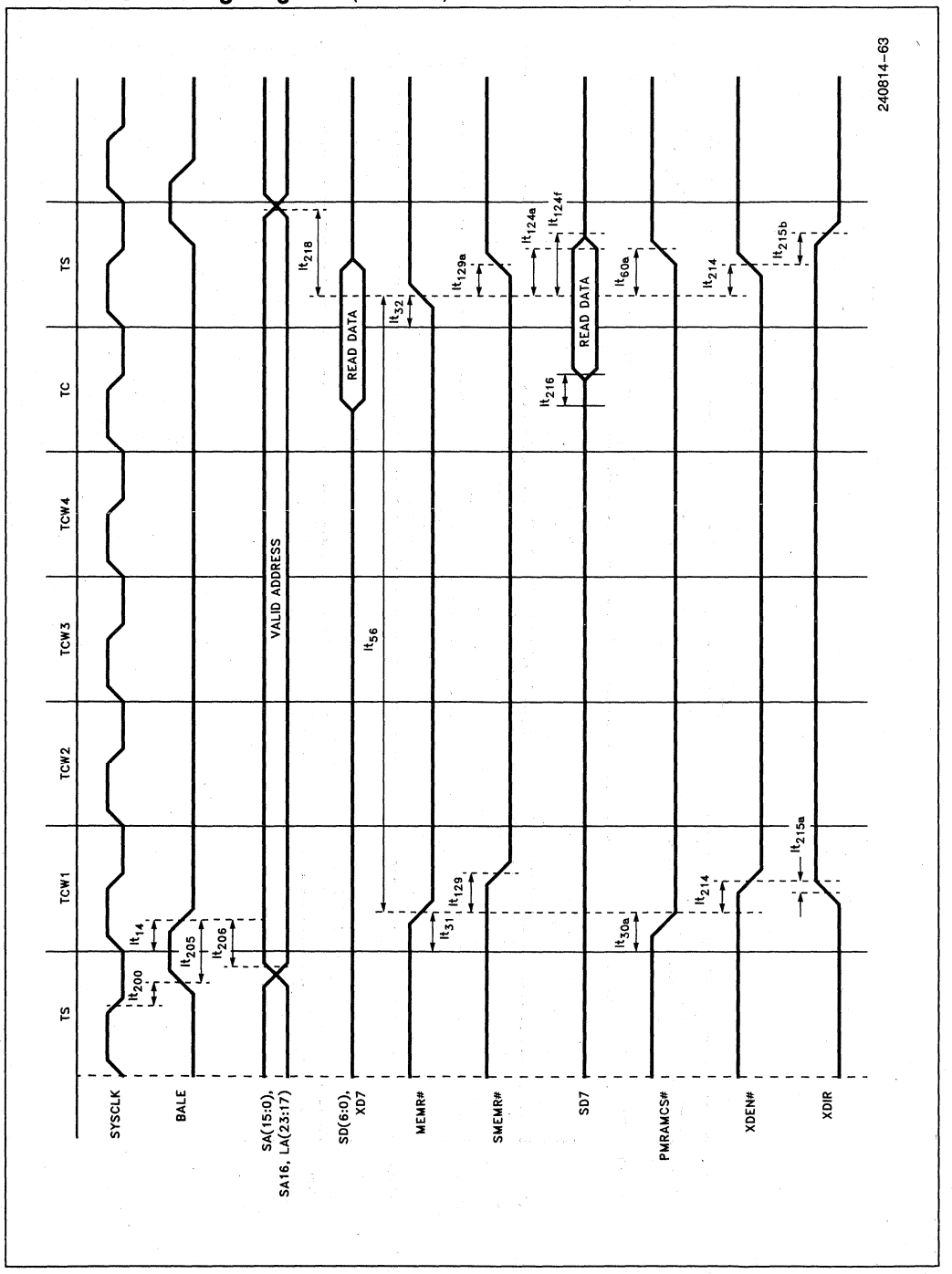


Figure 7.2.11. X-Bus Control Signals—Memory Read Timings

7.2 82360SL Timing Diagrams (Continued)

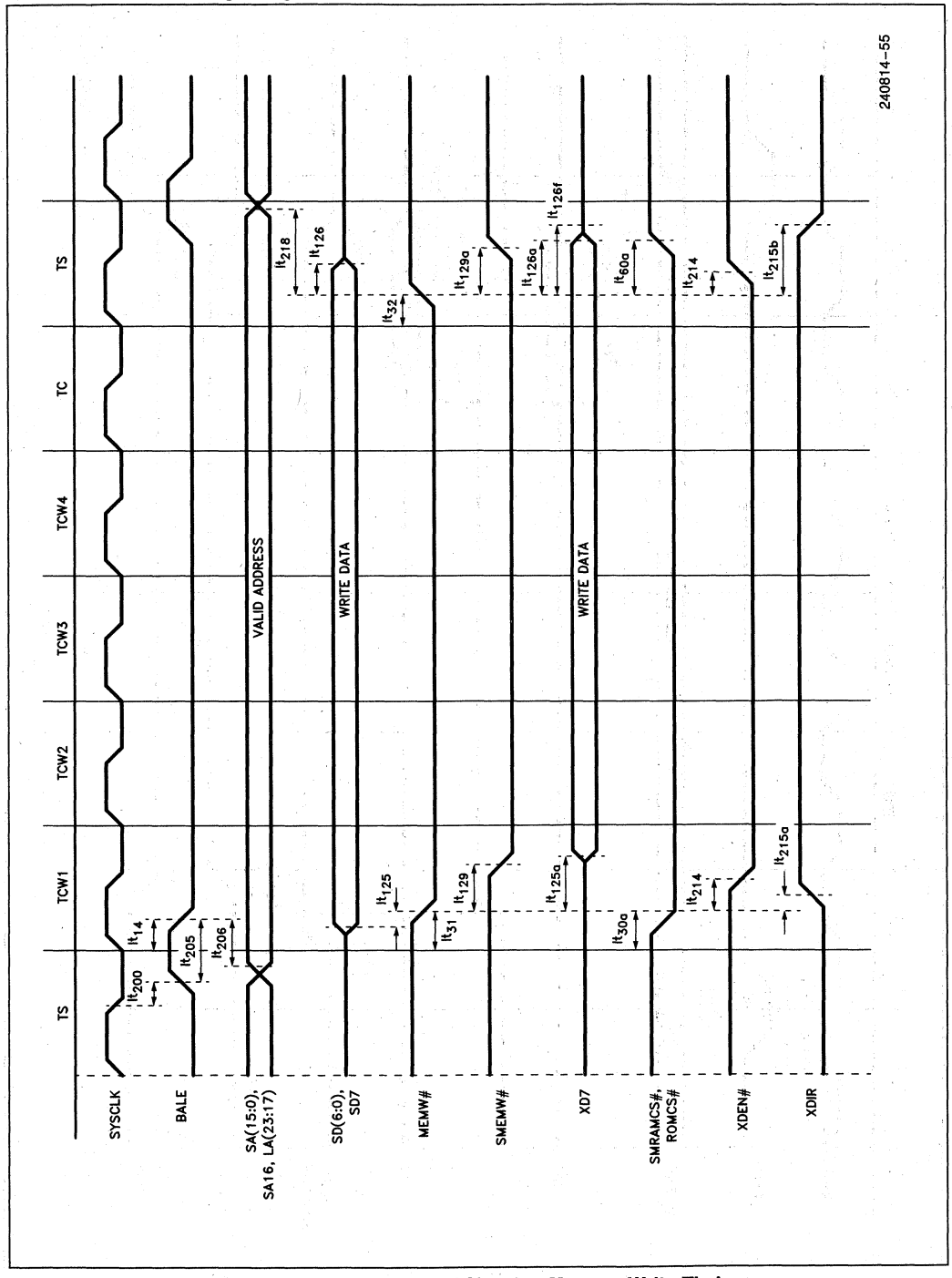


Figure 7.2.12. X-Bus Control Signals—Memory Write Timings

7.2 82360SL Timing Diagrams (Continued)

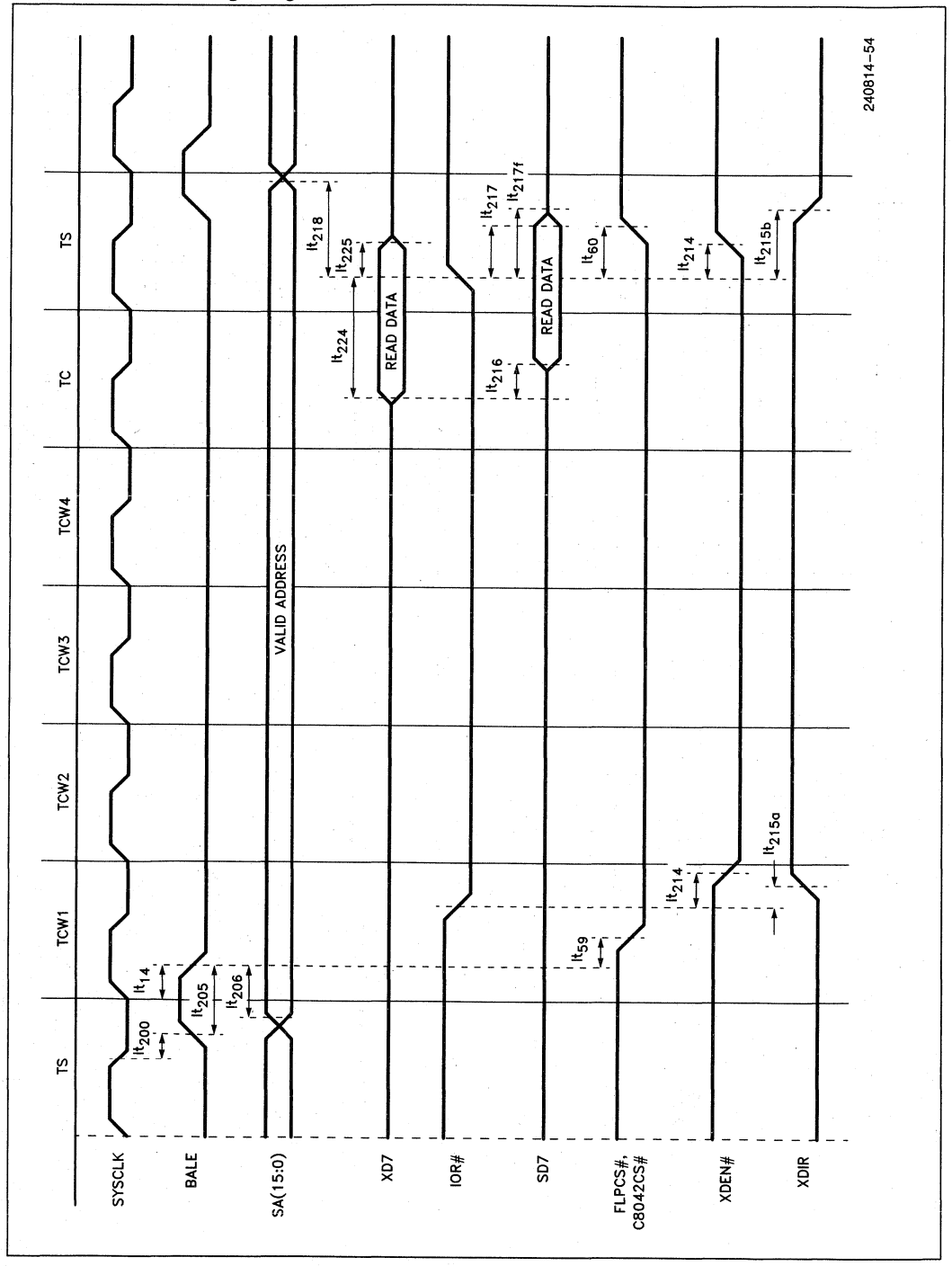


Figure 7.2.13. X-Bus Control Signals—I/O Read Timings

7.2 82360SL Timing Diagrams (Continued)

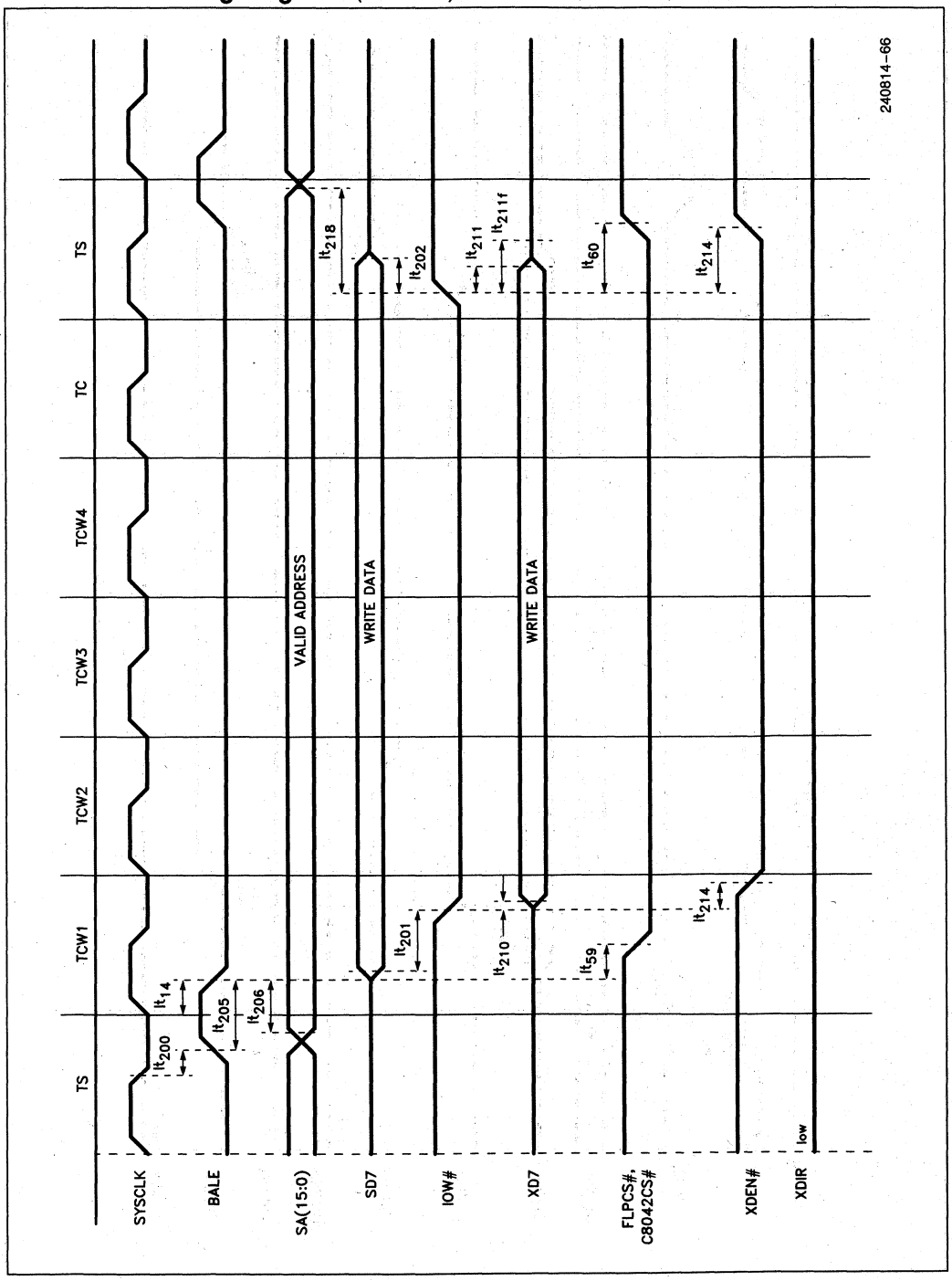


Figure 7.2.14. X-Bus Control Signals—I/O Write Timings

7.2 82360SL Timing Diagrams (Continued)

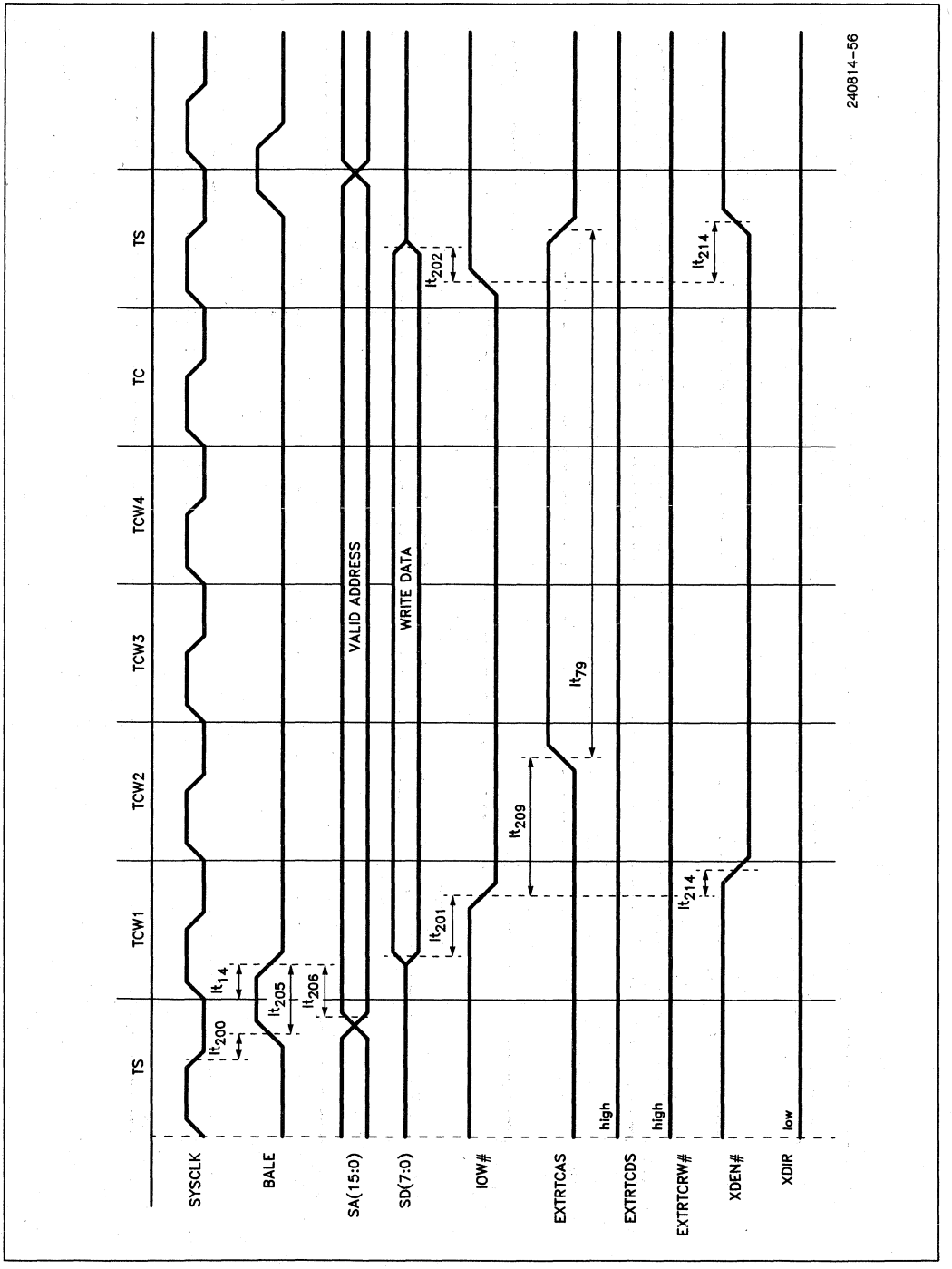


Figure 7.2.15. I/O Port 70 Hex Write—External RTC Timings

7.2 82360SL Timing Diagrams (Continued)

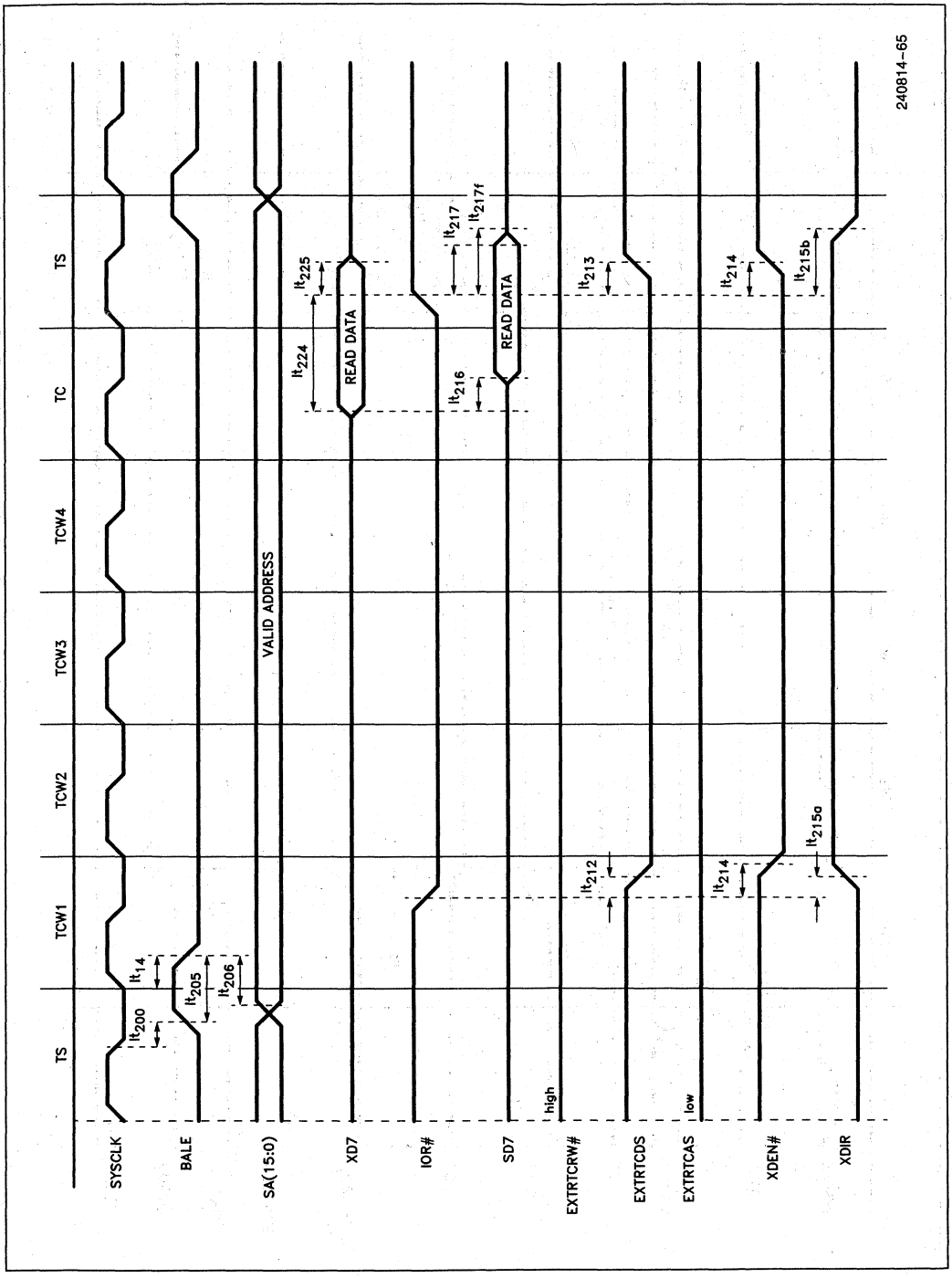


Figure 7.2.16. I/O Port 71 Hex Read—External RTC Timings

7.2 82360SL Timing Diagrams (Continued)

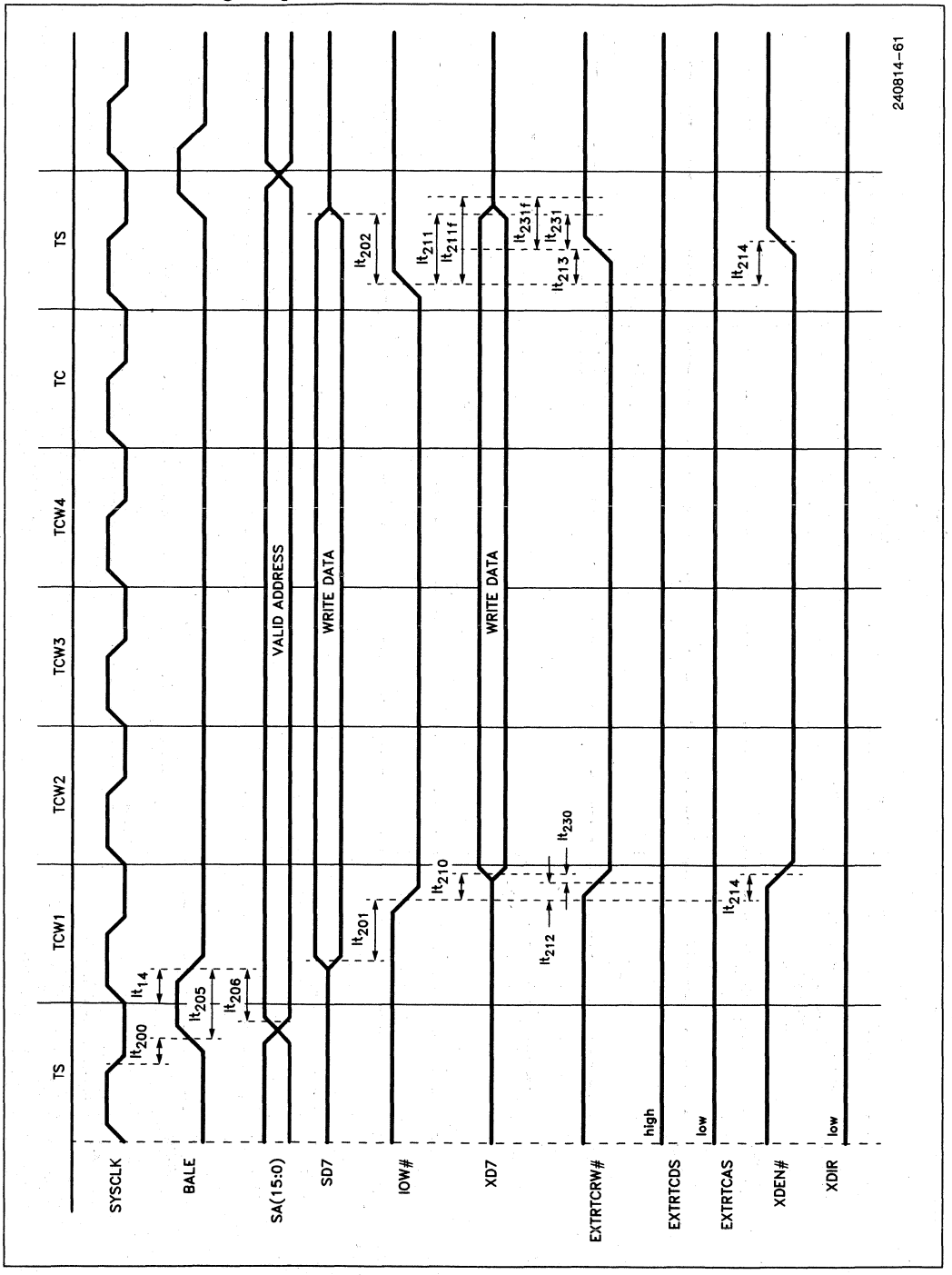


Figure 7.2.17. I/O Port 71 Hex Write—External RTC Timings

7.2 82360SL Timing Diagrams (Continued)

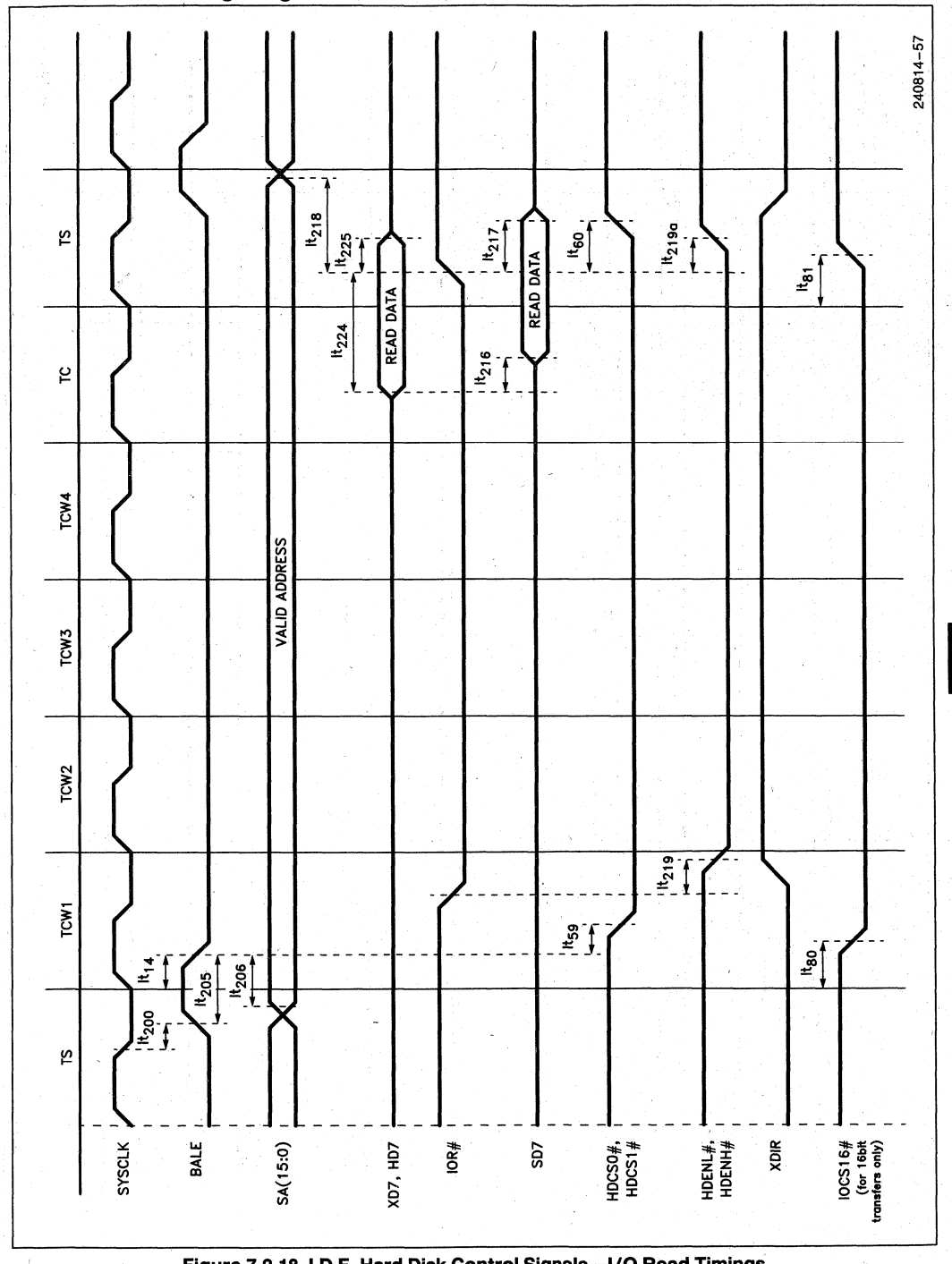


Figure 7.2.18. I.D.E. Hard Disk Control Signals—I/O Read Timings

7.2 82360SL Timing Diagrams (Continued)

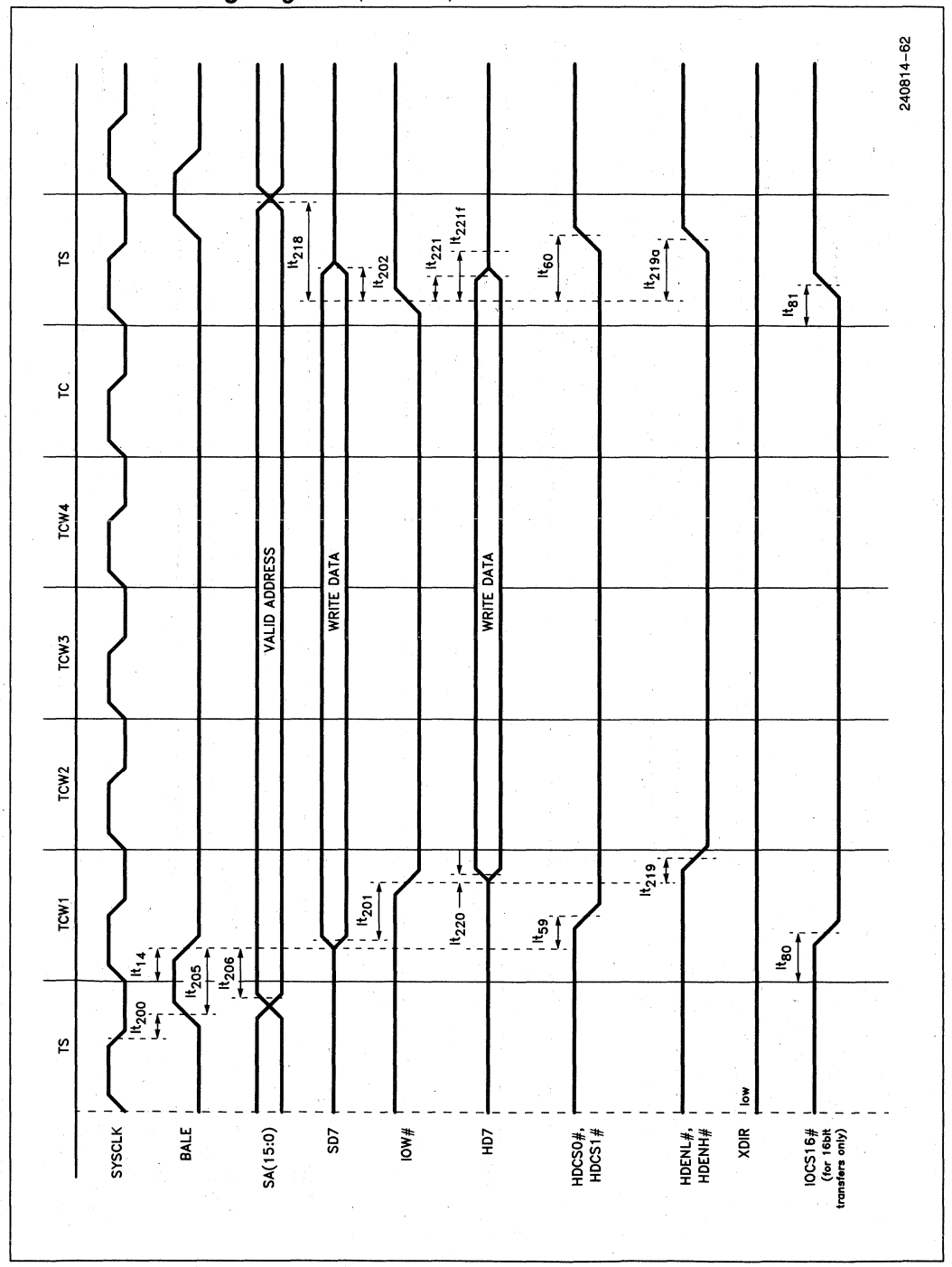


Figure 7.2.19. I.D.E. Hard Disk Control Signals—I/O Write Timings

7.2 82360SL Timing Diagrams (Continued)

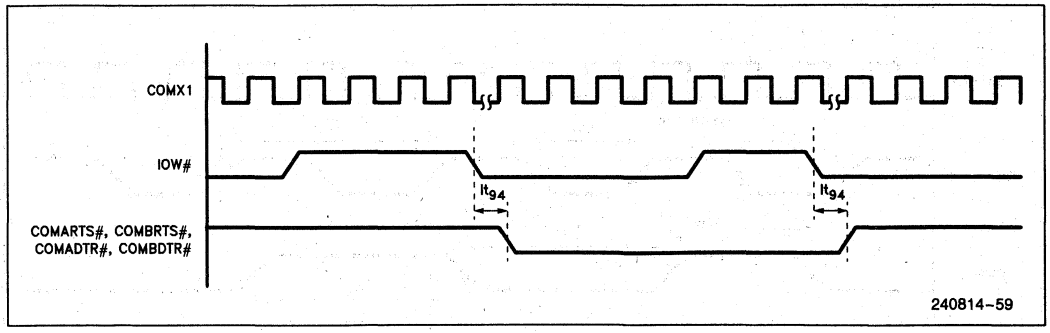


Figure 7.2.20. Serial Port Controller—Modem Control Signal Timings

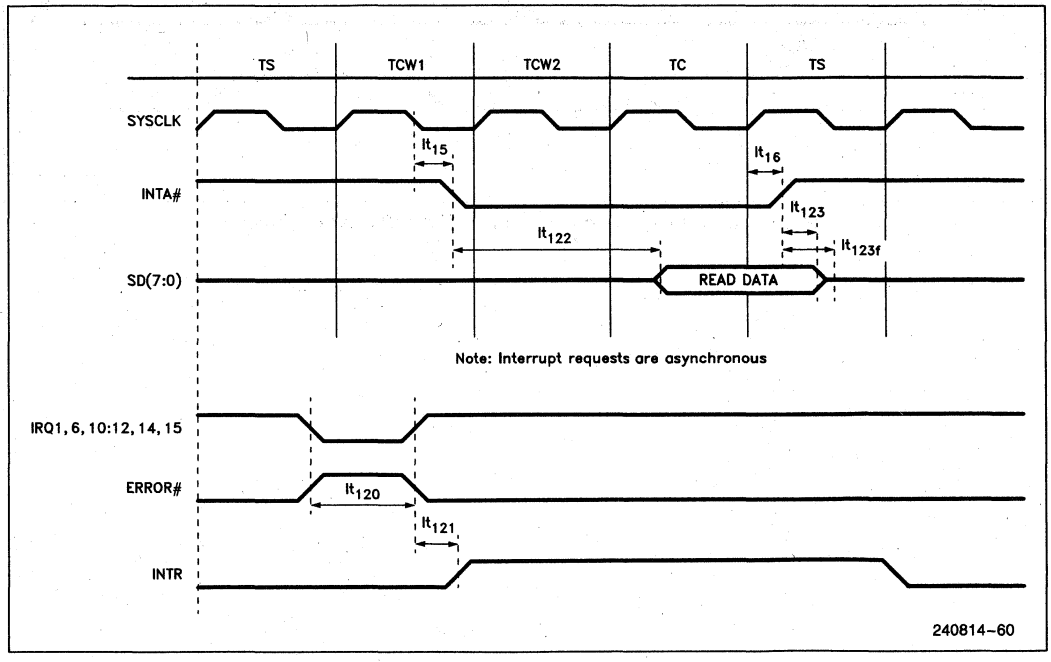


Figure 7.2.21. Interrupt Controller Timings

5

7.2 82360SL Timing Diagrams (Continued)

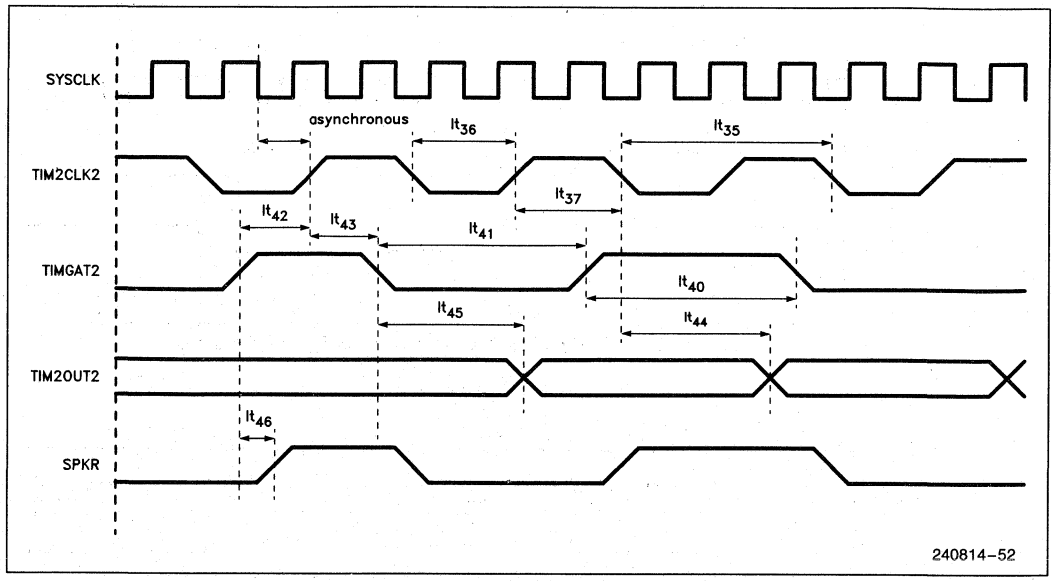


Figure 7.2.22. Programmable Interval Timer/Counter Timings

7.2 82360SL Timing Diagrams (Continued)

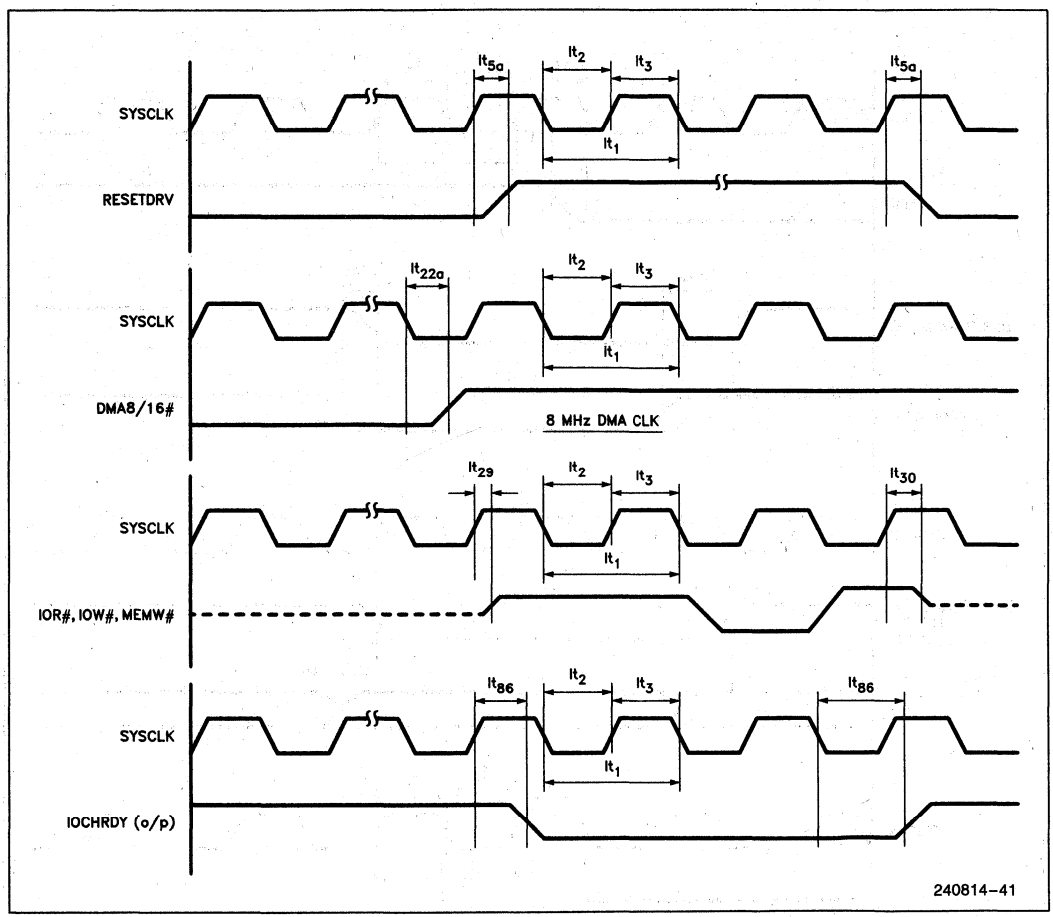


Figure 7.2.23. RESETDRV, DMA8/16#, Command Signals and IOCHRDY with Respect to SYSCLK

7.2 82360SL Timing Diagrams (Continued)

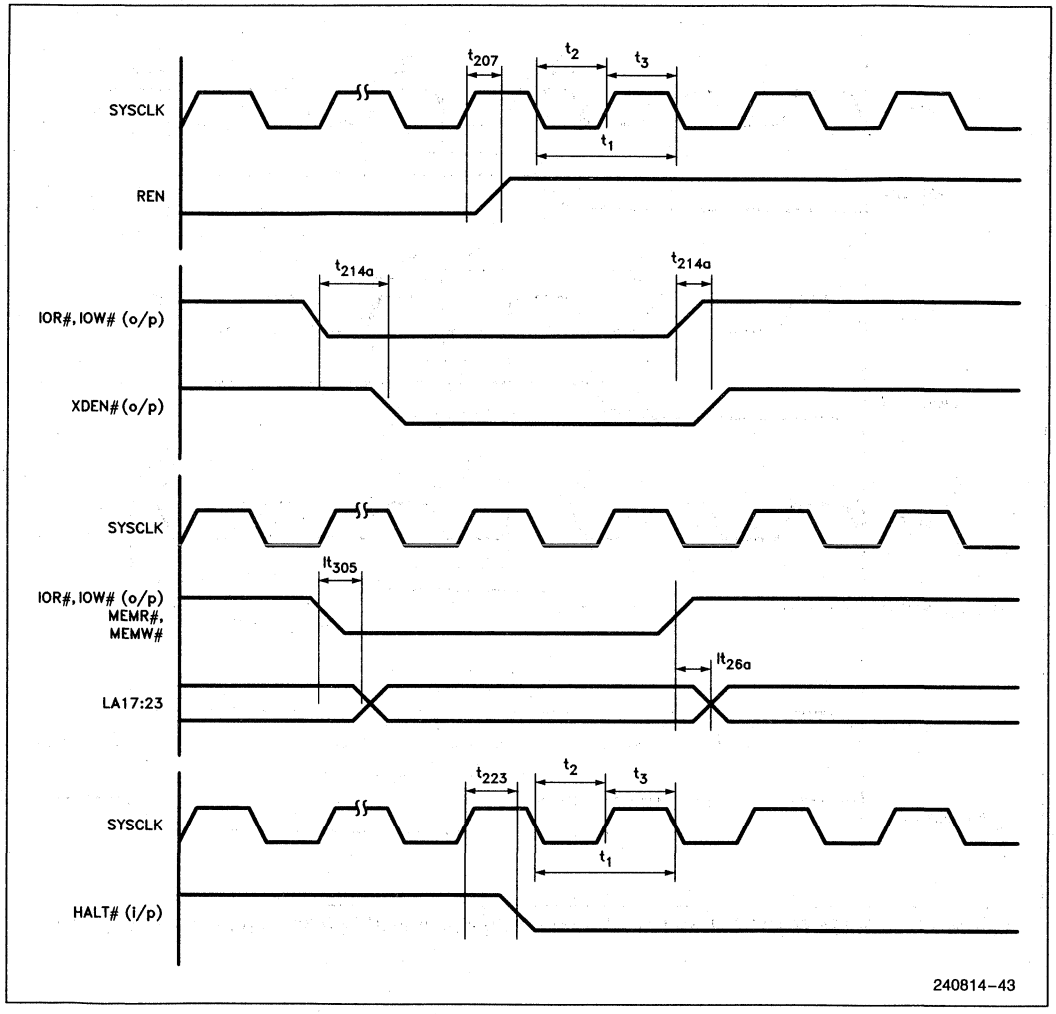


Figure 7.2.24. AEN and HALT with Respect to SYSCLK XDEN# and IOR#/IOW# with respect to LA17-23

7.2 82360SL Timing Diagrams (Continued)

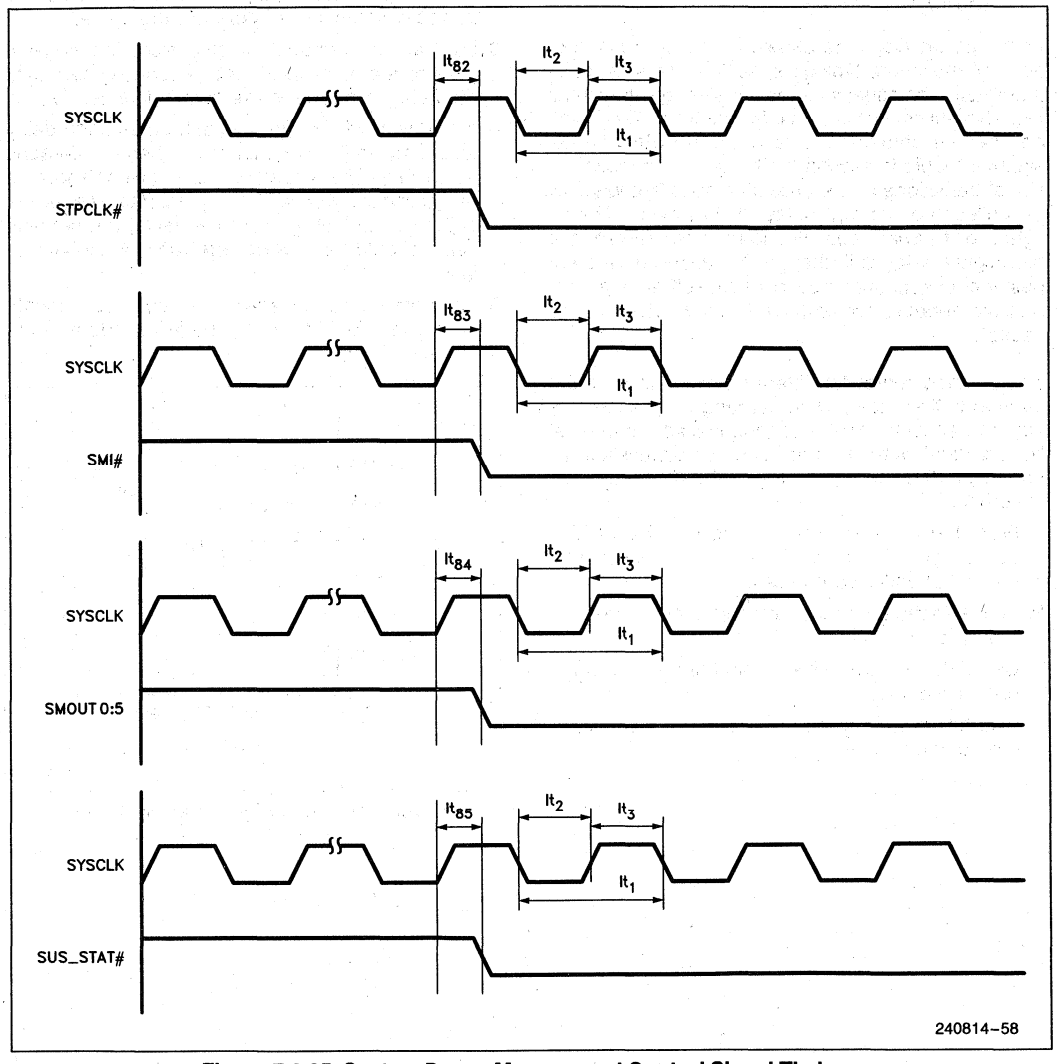


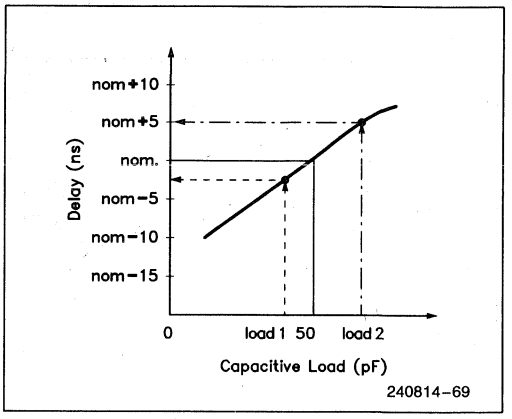
Figure 7.2.25. System Power Management Control Signal Timings

8.0 CAPACITIVE DERATING INFORMATION

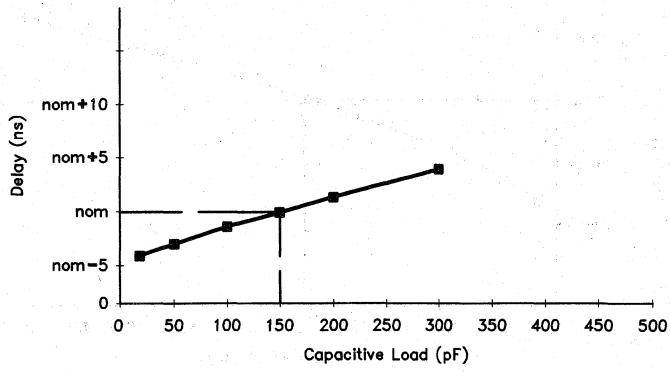
In the timing diagrams shown in the previous section, all maximum timings specified are at a maximum value of capacitive load tested on the signal pin. This maximum value is different for different pins and can be obtained for each pin from the pin assignment table in section 2. The delay introduced to signal transitions at the maximum specified load will be called the nominal delay. If, however, either a lighter or heavier capacitive load is connected to a pin, signal delay will change. To allow the system designer to account for such loading differences, capacitive derating curves have been provided in this section.

The derating curves for different pins depend on the internal buffers used. Nine derating curves are provided to account for the various classes of internal buffers used with different delay characteristics. To use these derating curves, follow the procedure outlined here.

1. From the Pin assignment chart, find the letter in the column "Derating Curve" corresponding to the signal under consideration.
2. In this section, find the derating curve of the correct type.
3. Calculate the capacitive loading on the signal under consideration.
4. Find this load point on the capacitive load axis of the derating curve.
5. Project a vertical line to the derating curve from the load point and draw a horizontal line from the point the vertical line intersects the curve.
6. Estimate the amount of time from the nominal point to the point where the horizontal line meets the delay axis. This is the derating value.
7. If the point where the horizontal meets the delay axis is above the nominal value, then this derating value should be **added** to signal timings shown in the timing diagrams. If the horizontal meets the delay axis below the nominal value, the derating value should be **subtracted** from the signal timings.
8. The derating curves shown can be used in identical manner for both rising and falling edges of the signal.

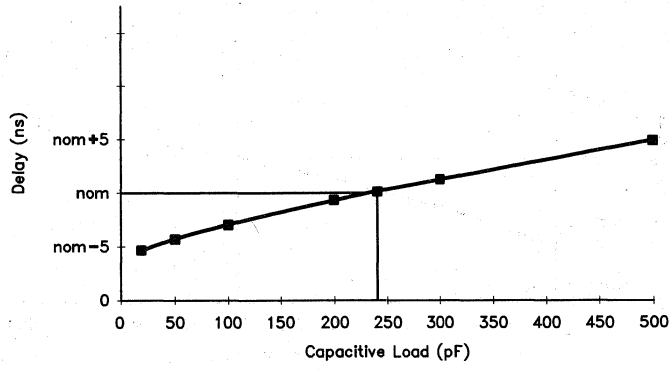


Using The Capacitive Derating Curves



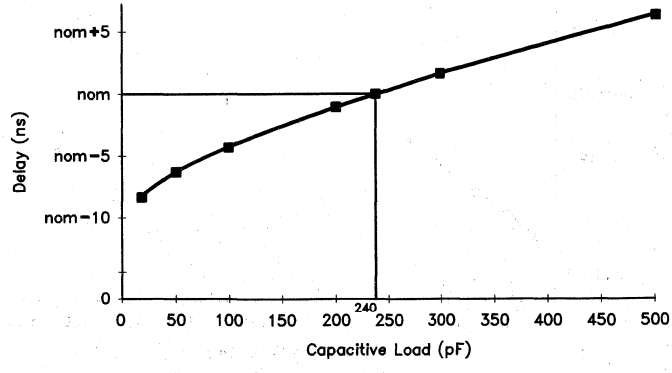
240814-70

Type A



240814-71

Type B

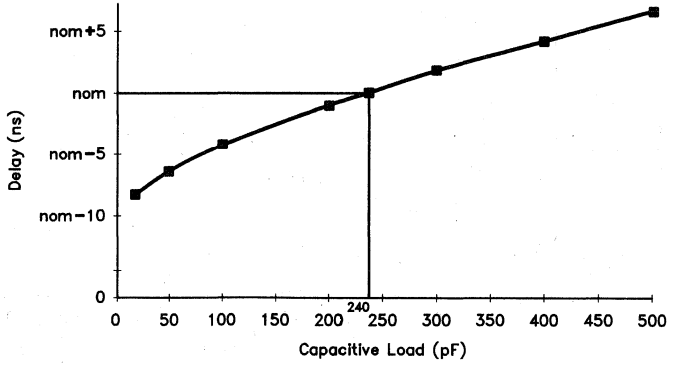


240814-72

Type C

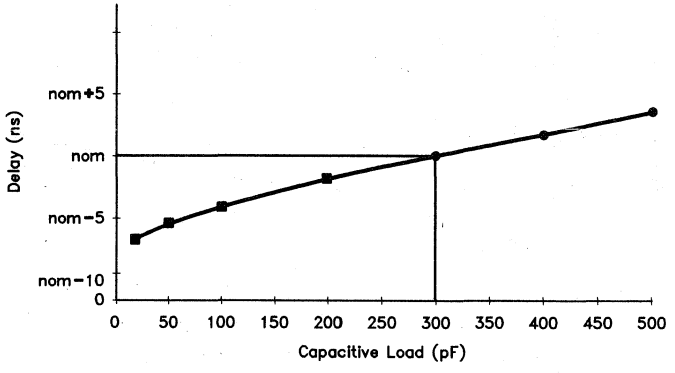
Capacitive Derating Curves

5



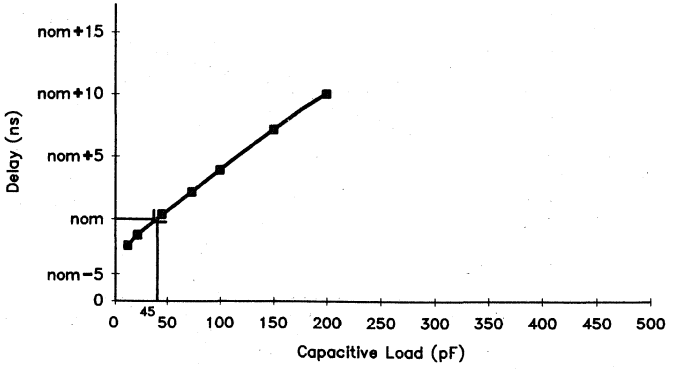
240814-73

Type D



240814-74

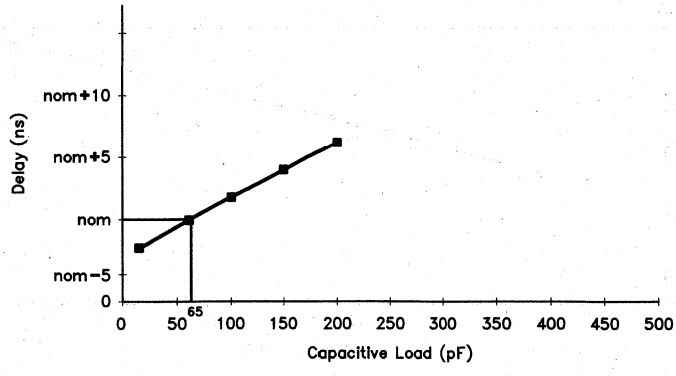
Type E



240814-75

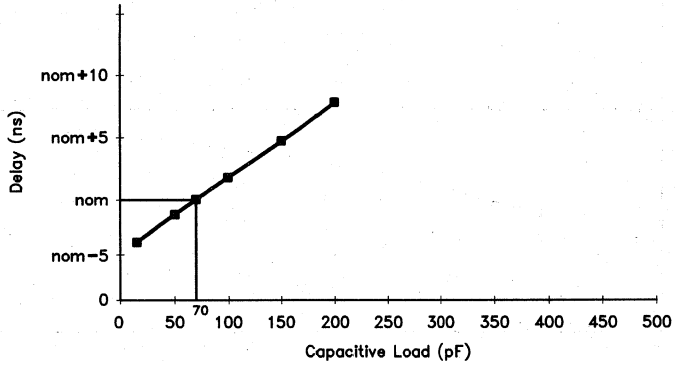
Type F

Capacitive Derating Curves (Continued)



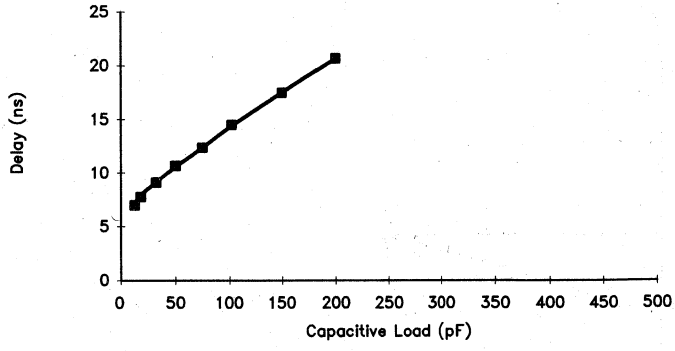
240814-76

Type G



240814-77

Type H

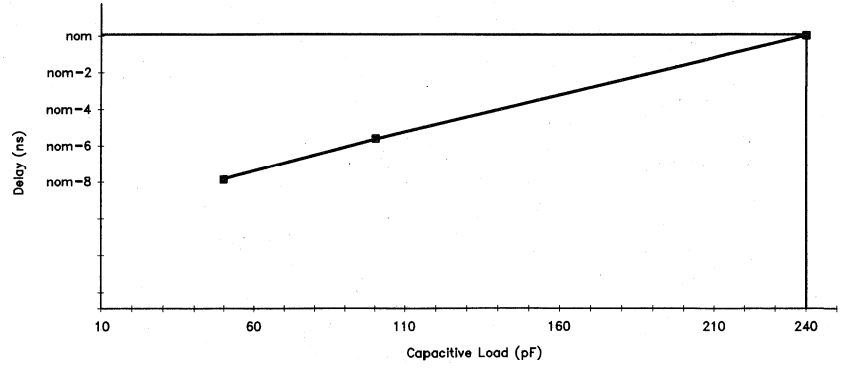


240814-78

Type I

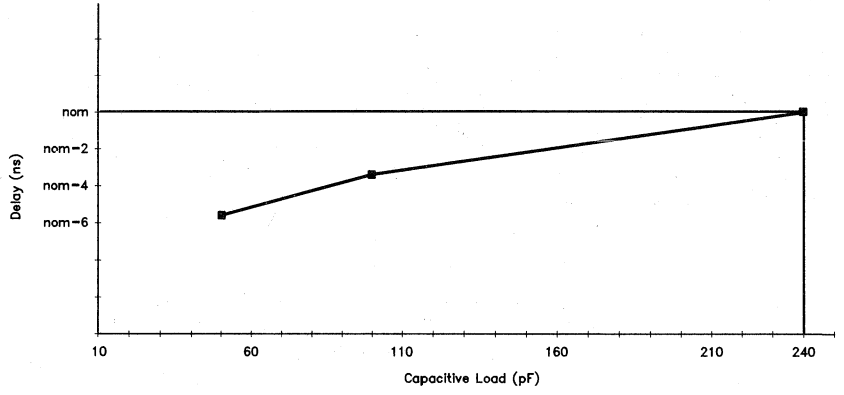
Capacitive Derating Curves (Continued)

5



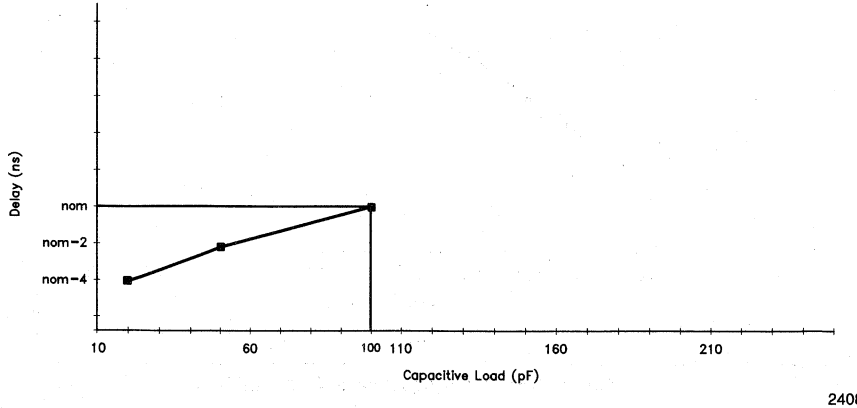
240814-79

Type J



240814-80

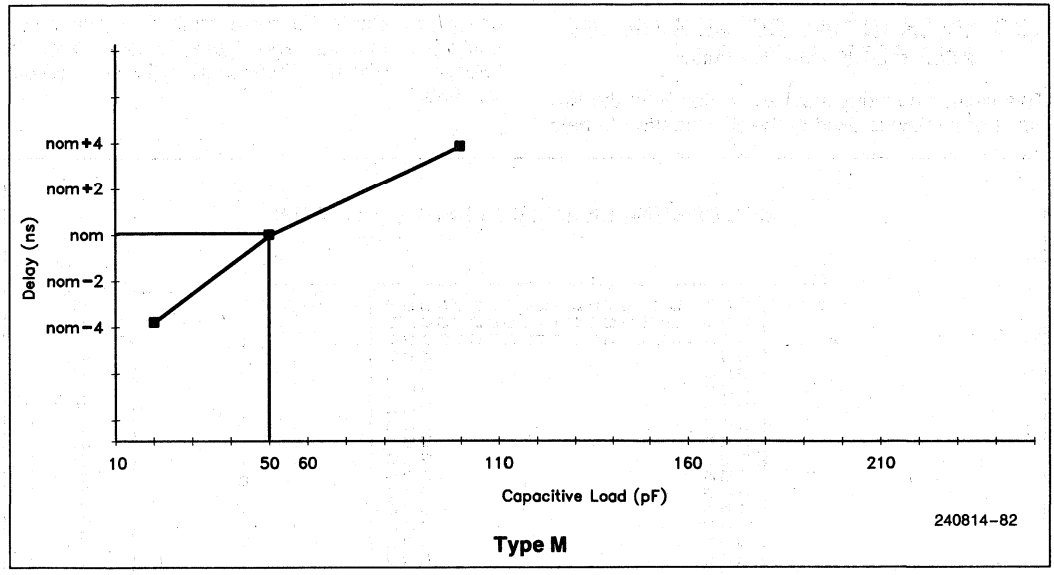
Type K



240814-81

Type L

Capacitive Derating Curves (Continued)



Capacitive Derating Curves (Continued)

9.0 DAMPING RESISTOR REQUIREMENTS

The SL SuperSet has powerful output buffers capable of directly driving large loads. These buffers are designed for fast signal transition times and hence have low output impedence. Due to a mismatch between the output impedance of the buffers and the characteristic impedance of the load (trace capacitance and the total number of devices) voltage overshoot and ringing can occur at signal transitions. By matching the output impedance with the characteristic input impedance and avoiding long trace lengths, the system designer can minimize the transmission line reflections and ringing.

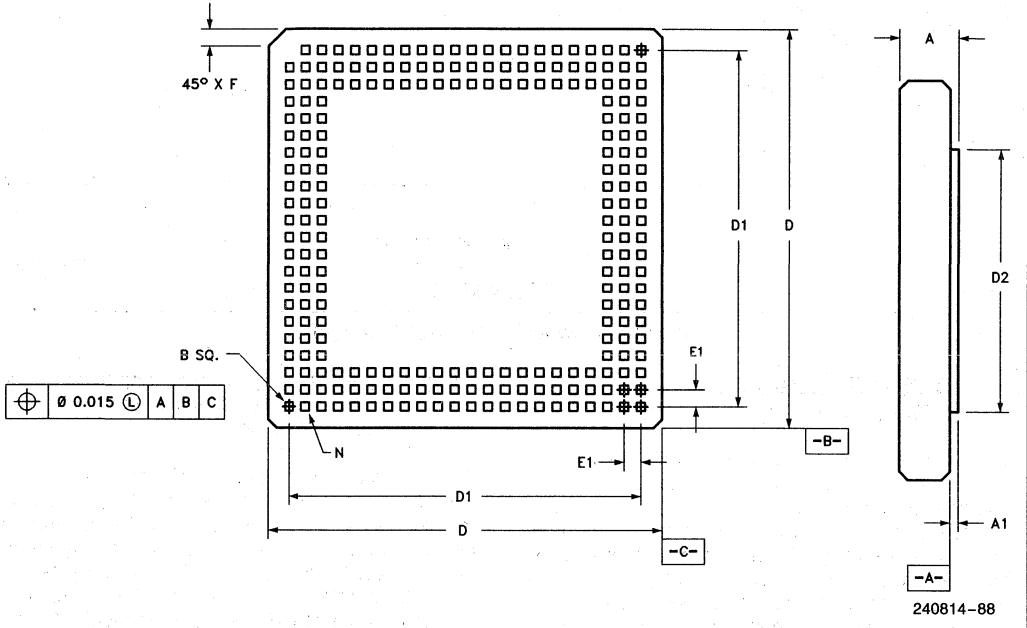
The ringing at signal transitions of address and data lines cause long unstable periods. Ringing on control signals can cause false latching. To minimize the ringing effect series damping resistors may have to be connected. For additional hardware system design information, consult see the 386™ SL Microprocessor SuperSet System Design Guide (Intel Order # 240816).

5

10.0 MECHANICAL DETAILS OF LGA AND PQFP PACKAGES

This section contains mechanical details of the two types of packages used in the SL SuperSet to help design the parts in. For more detailed information on packages and package types, please refer to *"Surface Mount Technology Guide"* (Order #240585)

227L CERAMIC LAND GRID ARRAY (CAVITY UP)



Family: Ceramic Land Grid Array Package						
Symbol	Millimeters			Inches		
	Min	Max	Notes	Min	Max	Notes
1	3.56	4.06		0.097	0.120	
A1	0.33	0.43		0.009	0.017	
B	0.69	0.84		0.027	0.033	
D	28.96	29.46		1.140	1.160	
D1	26.67		Basic	1.050		Basic
D2		24.13			0.950	
e1	1.27		Basic	0.050		Basic
F	1.65	2.16		0.065	0.085	
N	227			227		
Issue	4/17/90					

Figure 10-1a. Principal Dimensions of the 386™ SL CPU in a 227-Lead LGA Package

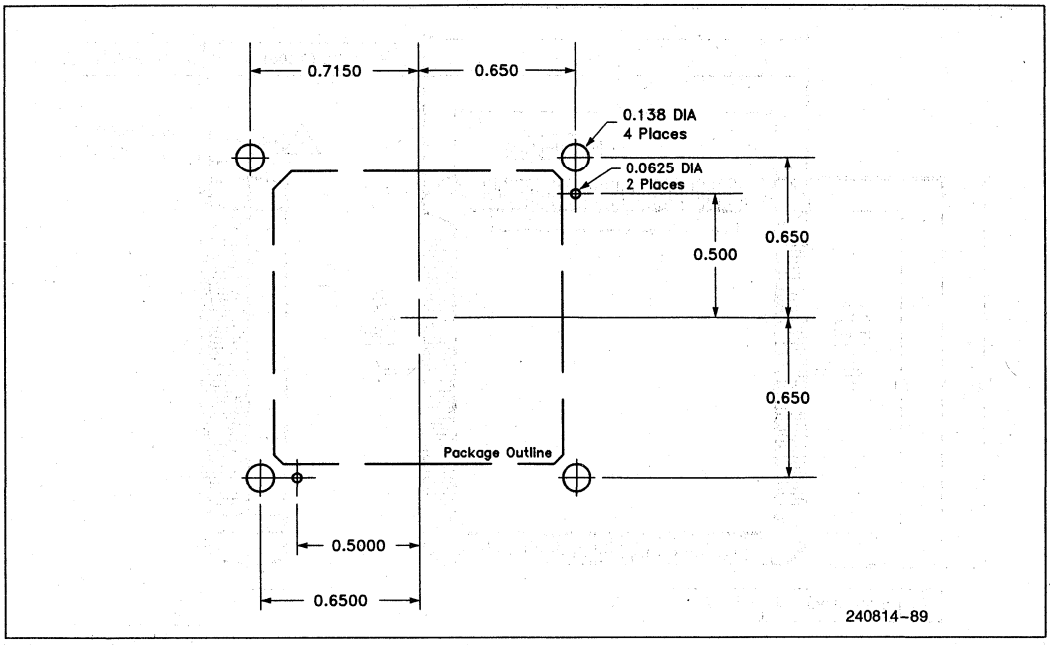
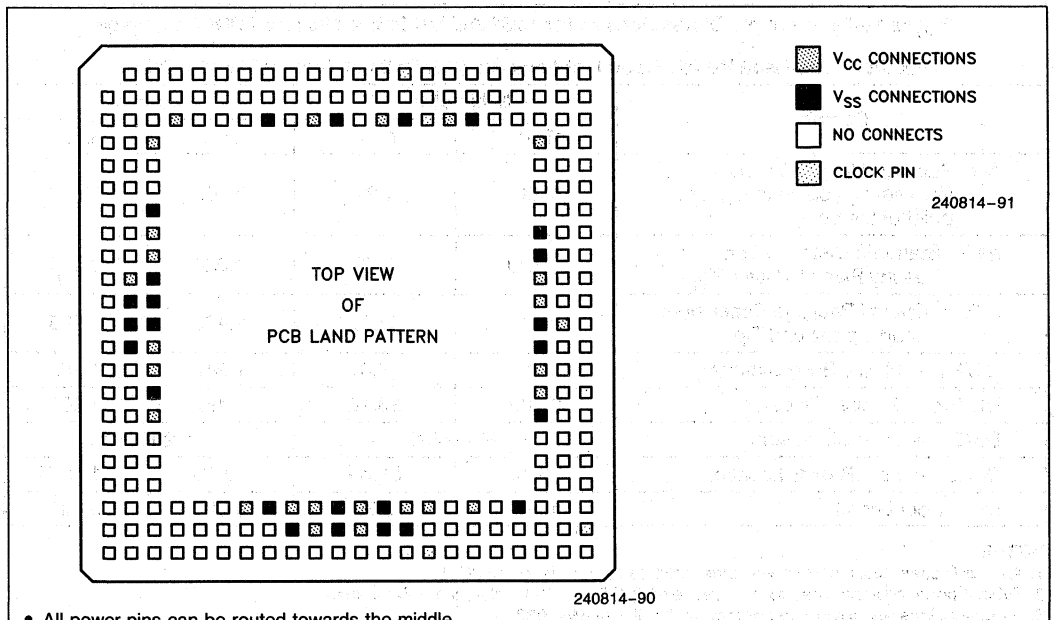


Figure 10-b. Recommended LGA Socket Footprint



- All power pins can be routed towards the middle.
- Outer two rows route outward
- Clock pins should have shortest possible traces, then via to shielded inner layer.

Figure 10-1c. Recommended Signal Routing for LGA Package

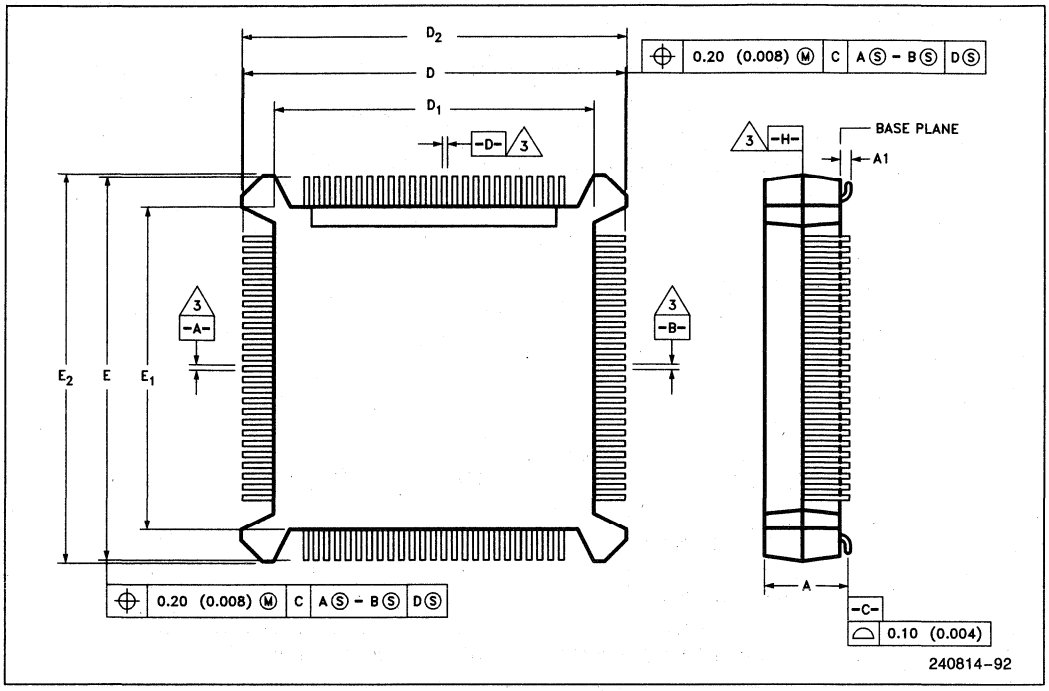


Figure 10-2a. Principle Dimensions of the 82360SL I/O in the 196-Lead PQFP Package

Family: 196-Lead Plastic Quad Flat Package (PQFP) 0.025 Inch (0.635mm) Pitch

Symbol	Millimeters		Inches	
	Min	Max	Min	Max
A = Package Height: Distance from seating plane to highest point of the body	4.06	4.32	0.160	0.170
A1 = Standoff: Distance from Seating Plane to Base Plane	0.51	0.76	0.020	0.030
D/E = Overall Package Dimension: Lead Tip to Lead Tip	37.47	37.72	1.475	1.485
D1/E1 = Plastic Body Dimension	34.21	34.37	1.347	1.353
D2/E2 = Bumper Distance	38.02	38.18	1.497	1.503
D3/E3 = Lead Dimension	30.48 Ref		1.200 Ref	
D4/E4 = Foot Radius Location	36.14	36.49	1.423	1.437
L1 = Foot Length	0.51	0.76	0.020	0.030

NOTES:
1. All PQFP case outlines are being presented as standards to the JEDEC.
2. Typical board footprint area for the 196-lead PQFP is 1.500 inches x 1.5000 inches.
3. All dimensions and tolerance conform to ANSI Y14.5M-1982.
4. Datum Plane -H- located at the molding parting line and coincident with the bottom of the lead where the lead exits the plastic body.
5. Datums A-B and -D- to be determined where the center lead exits the plastic body at datum plane -H-.
6. Controlling dimension in inches.
7. Dimensions D1, D2, E1, and E2 are measured at the molding parting line and do not include mold protrusions.
8. Pin 1 identifier is located within one of the two zones indicated.
9. Measured at datum plane -H-.
10. Measured at seating plane datum -C-.

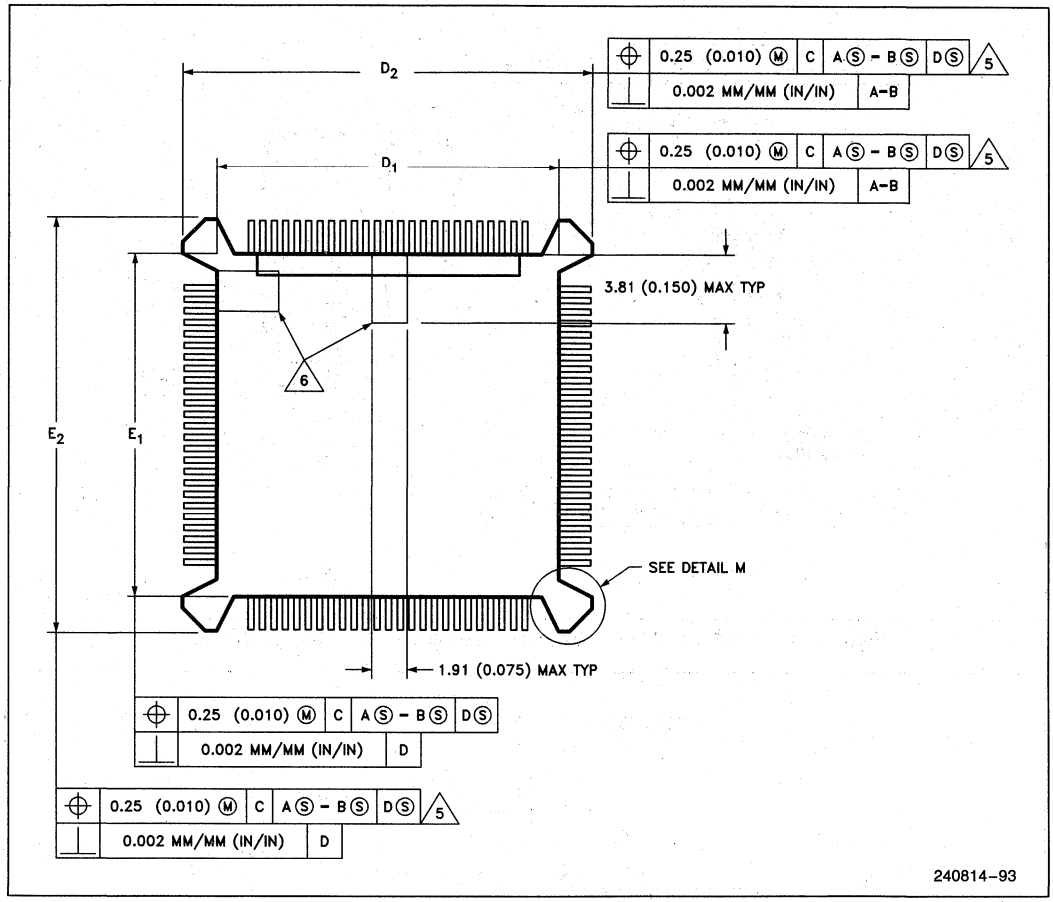


Figure 10-2b. Detailed Dimensions of the 82360SL I/O in the 196-Lead PQFP—Molded Details

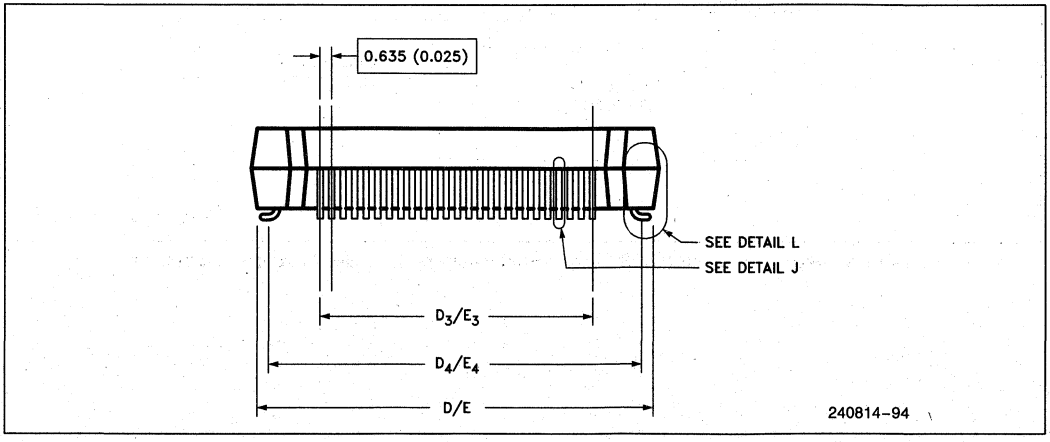


Figure 10-2c. Detailed Dimensions of the 82360SL I/O in the 196-Lead—Terminal Details

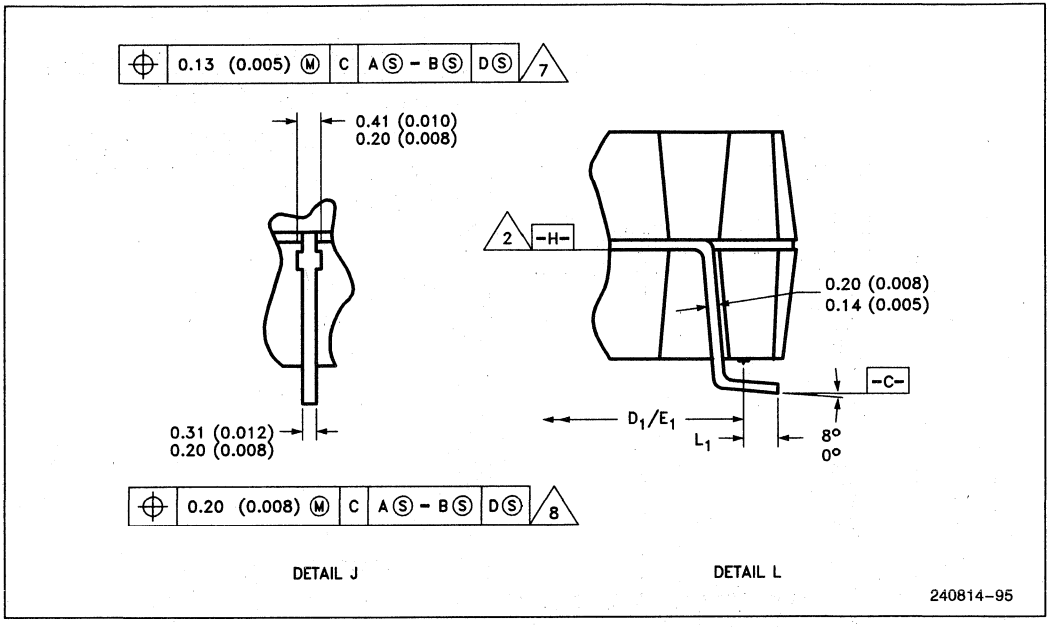


Figure 10-2d. 196-Lead PQFP Mechanical Package Detail—Typical Lead

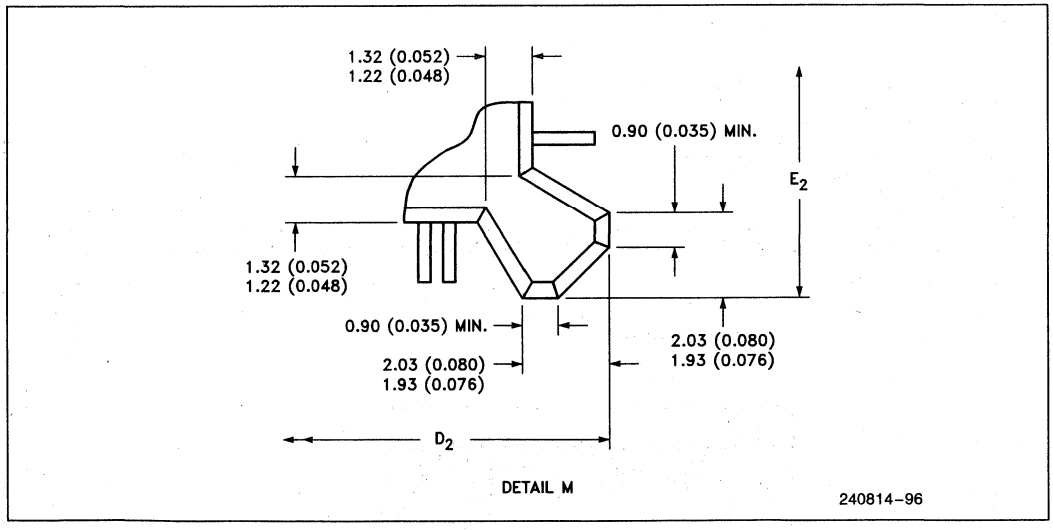


Figure 10-2e. 196-Lead PQFP Mechanical Package Detail—Protective Bumper

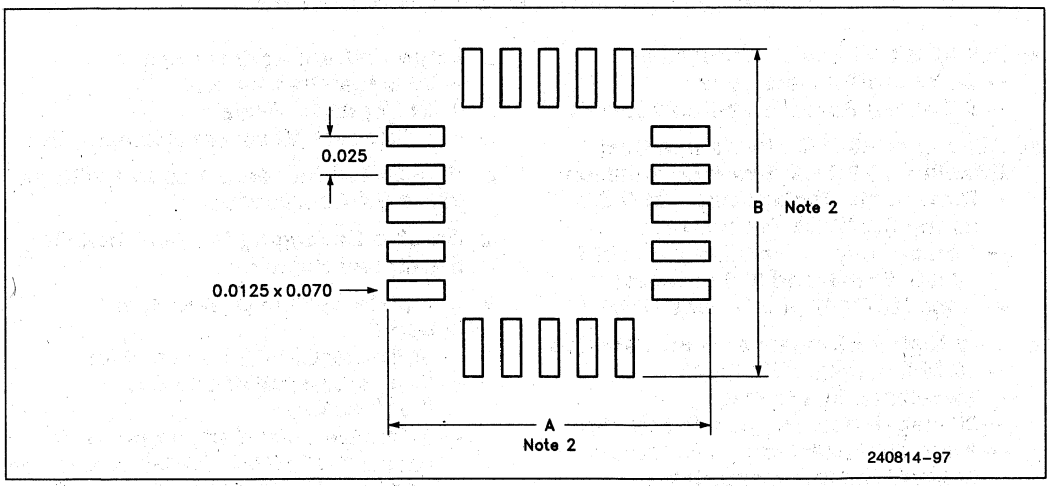


Figure 10-2f. Recommended PQFP Footprint

11.0 REVISION HISTORY

The First Release of the Advanced Information Data Sheet reflects information believed to be accurate as of September 1990.

Please Consult your Local Intel Field Sales Office for the most current design-in information.

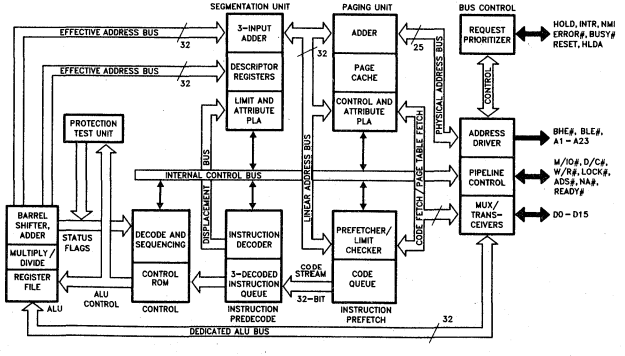
5

386™ SX MICROPROCESSOR

- **Full 32-Bit Internal Architecture**
  - **— 8-, 16-, 32-Bit Data Types**
  - **— 8 General Purpose 32-Bit Registers**
- **Runs Intel386™ Software in a Cost Effective 16-Bit Hardware Environment**
  - **— Runs Same Applications and O.S.'s as the 386™ DX Processor**
  - **— Object Code Compatible with 8086, 80186, 80286, and 386 Processors**
  - **— Runs MS-DOS*, OS/2* and UNIX****
- **Very High Performance 16-Bit Data Bus**
  - **— 16 MHz and 20 MHz Clock**
  - **— Two-Clock Bus Cycles**
  - **— 20 Megabytes/Sec Bus Bandwidth**
  - **— Address Pipelining Allows Use of Slower/Cheaper Memories**
- **Integrated Memory Management Unit**
  - **— Virtual Memory Support**
  - **— Optional On-Chip Paging**
  - **— 4 Levels of Hardware Enforced Protection**
  - **— MMU Fully Compatible with Those of the 80286 and 386 DX CPUs**
- **Virtual 8086 Mode Allows Execution of 8086 Software in a Protected and Paged System**
- **Large Uniform Address Space**
  - **— 16 Megabyte Physical**
  - **— 64 Terabyte Virtual**
  - **— 4 Gigabyte Maximum Segment Size**
- **High Speed Numerics Support with the 387™ SX Coprocessor**
- **On-Chip Debugging Support Including Breakpoint Registers**
- **Complete System Development Support**
  - **— Software: C, PL/M, Assembler**
  - **— Debuggers: PMON-386 DX, ICE™-386 SX**
  - **— Extensive Third-Party Support: C, Pascal, FORTRAN, BASIC, Ada*** on VAX®†, UNIX**, MS-DOS*, and Other Hosts**
- **High Speed CHMOS III and CHMOS IV Technology**
- **Operating Frequency:**
  - **— Standard (386™ SX -20, -16) Min/Max Frequency (4/20, 4/16) MHz**
  - **— Low Power (386™ SX -20, -16, -12) Min/Max Frequency (2/20, 2/16, 2/12) MHz**
- **100-Pin Plastic Quad Flatpack Package**

(See Packaging Outlines and Dimensions #231369)

The 386™ SX Microprocessor is a 32-bit CPU with a 16-bit external data bus and a 24-bit external address bus. The 386 SX CPU brings the high-performance software of the Intel386™ Architecture to midrange systems. It provides the performance benefits of a 32-bit programming architecture with the cost savings associated with 16-bit hardware systems.



240187-47

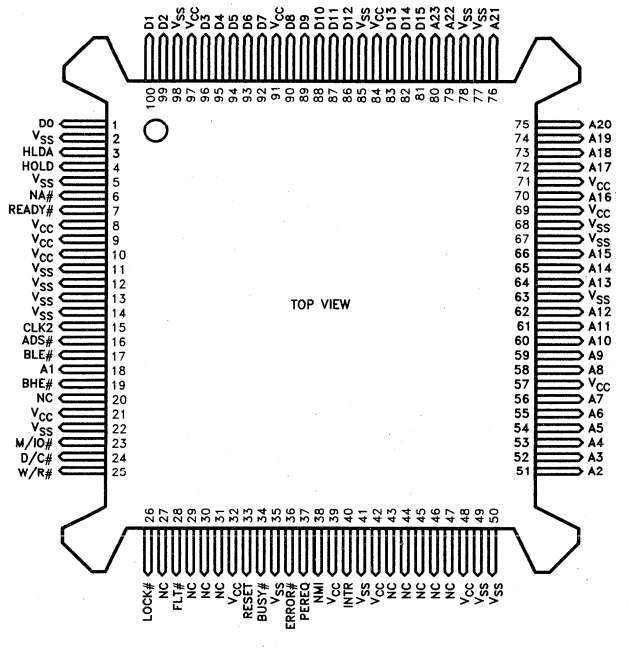
386™ SX Pipelined 32-Bit Microarchitecture

†VAX® is a registered trademark of the Digital Equipment Corporation.
*MS-DOS and OS/2 are trademarks of Microsoft Corporation.
**UNIX is a trademark of AT&T.
***Ada is a trademark of the Department of Defense.

October 1990
Order Number: 240187-006

Chapter 1 PIN DESCRIPTION 5-866
Chapter 2 BASE ARCHITECTURE 5-870
2.1 Register Set 5-870
2.2 Instruction Set 5-873
2.3 Memory Organization 5-874
2.4 Addressing Modes 5-875
2.5 Data Types 5-878
2.6 I/O Space 5-878
2.7 Interrupts and Exceptions 5-880
2.8 Reset and Initialization 5-883
2.9 Testability 5-883
2.10 Debugging Support 5-884
Chapter 3 REAL MODE ARCHITECTURE 5-885
3.1 Memory Addressing 5-885
3.2 Reserved Locations 5-886
3.3 Interrupts 5-886
3.4 Shutdown and Halt 5-886
3.5 LOCK Operations 5-886
Chapter 4 PROTECTED MODE ARCHITECTURE 5-887
4.1 Addressing Mechanism 5-887
4.2 Segmentation 5-887
4.3 Protection 5-892
4.4 Paging 5-896
4.5 Virtual 8086 Environment 5-899
Chapter 5 FUNCTIONAL DATA 5-902
5.1 Signal Description Overview 5-902
5.2 Bus Transfer Mechanism 5-908
5.3 Memory and I/O Spaces 5-908
5.4 Bus Functional Description 5-908
5.5 Self-test Signature 5-926
5.6 Component and Revision Identifiers 5-926
5.7 Coprocessor Interfacing 5-926
Chapter 6 PACKAGE THERMAL SPECIFICATIONS 5-927
Chapter 7 ELECTRICAL SPECIFICATIONS 5-927
7.1 Power and Grounding 5-927
7.2 Maximum Ratings 5-928
7.3 D.C. Specifications 5-929
7.4 A.C. Specifications 5-930
7.5 Designing for ICE™-386 SX Use (Preliminary Data) 5-938
Chapter 8 DIFFERENCES BETWEEN THE 386™ SX Microprocessor and the 386™ DX CPU 5-938
Chapter 9 INSTRUCTION SET 5-939
9.1 386™ SX Microprocessor Instruction Encoding and Clock Count Summary 5-939
9.2 Instruction Encoding 5-954

1.0 PIN DESCRIPTION



NOTE:
NC = No Connect

Figure 1.1. 386™ SX Microprocessor Pin out Top View

Table 1.1. Alphabetical Pin Assignments

Address		Data		Control		N/C	V_{CC}	V_{SS}
A_1	18	D_0	1	ADS#	16	20	8	2
A_2	51	D_1	100	BHE#	19	27	9	5
A_3	52	D_2	99	BLE#	17	29	10	11
A_4	53	D_3	96	BUSY#	34	30	21	12
A_5	54	D_4	95	CLK2	15	31	32	13
A_6	55	D_5	94	D/C#	24	43	39	14
A_7	56	D_6	93	ERROR#	36	44	42	22
A_8	58	D_7	92	FLT#	28	45	48	35
A_9	59	D_8	90	HLDA	3	46	57	41
A_{10}	60	D_9	89	HOLD	4	47	69	49
A_{11}	61	D_{10}	88	INTR	40		71	50
A_{12}	62	D_{11}	87	LOCK#	26		84	63
A_{13}	64	D_{12}	86	M/IO#	23		91	67
A_{14}	65	D_{13}	83	NA#	6		97	68
A_{15}	66	D_{14}	82	NMI	38			77
A_{16}	70	D_{15}	81	PEREQ	37			78
A_{17}	72			READY#	7			85
A_{18}	73			RESET	33			98
A_{19}	74			W/R#	25			
A_{20}	75							
A_{21}	76							
A_{22}	79							
A_{23}	80							

1.0 PIN DESCRIPTION (Continued)

The following are the 386™ SX Microprocessor pin descriptions. The following definitions are used in the pin descriptions:

\# The named signal is active LOW.
I Input signal.
O Output signal.
I/O Input and Output signal.
- No electrical connection.

Symbol	Type	Pin	Name and Function
CLK2	I	15	**CLK2** provides the fundamental timing for the 386™ SX Microprocessor. For additional information see **Clock.**
RESET	I	33	**RESET** suspends any operation in progress and places the 386™ SX Microprocessor in a known reset state. See **Interrupt Signals** for additional information.
D_{15}–D_0	I/O	81-83,86-90, 92-96,99-100,1	**Data Bus** inputs data during memory, I/O and interrupt acknowledge read cycles and outputs data during memory and I/O write cycles. See **Data Bus** for additional information.
A_{23}–A_1	O	80-79,76-72,70, 66-64,62-58, 56-51,18	**Address Bus** outputs physical memory or port I/O addresses. See **Address Bus** for additional information.
W/R#	O	25	**Write/Read** is a bus cycle definition pin that distinguishes write cycles from read cycles. See **Bus Cycle Definition Signals** for additional information.
D/C#	O	24	**Data/Control** is a bus cycle definition pin that distinguishes data cycles, either memory or I/O, from control cycles which are: interrupt acknowledge, halt, and code fetch. See **Bus Cycle Definition Signals** for additional information.
M/IO#	O	23	**Memory/IO** is a bus cycle definition pin that distinguishes memory cycles from input/output cycles. See **Bus Cycle Definition Signals** for additional information.
LOCK#	O	26	**Bus Lock** is a bus cycle definition pin that indicates that other system bus masters are not to gain control of the system bus while it is active. See **Bus Cycle Definition Signals** for additional information.
ADS#	O	16	**Address Status** indicates that a valid bus cycle definition and address (W/R#, D/C#, M/IO#, BHE#, BLE# and A_{23}–A_1 are being driven at the 386™ SX Microprocessor pins. See **Bus Control Signals** for additional information.
NA#	I	6	**Next Address** is used to request address pipelining. See **Bus Control Signals** for additional information.
READY#	I	7	**Bus Ready** terminates the bus cycle. See **Bus Control Signals** for additional information.
BHE#, BLE#	O	19,17	**Byte Enables** indicate which data bytes of the data bus take part in a bus cycle. See **Address Bus** for additional information.

5

1.0 PIN DESCRIPTION (Continued)

Symbol	Type	Pin	Name and Function
HOLD	I	4	**Bus Hold Request** input allows another bus master to request control of the local bus. See **Bus Arbitration Signals** for additional information.
HLDA	O	3	**Bus Hold Acknowledge** output indicates that the 386™ SX Microprocessor has surrendered control of its local bus to another bus master. See **Bus Arbitration Signals** for additional information.
INTR	I	40	**Interrupt Request** is a maskable input that signals the 386™ SX Microprocessor to suspend execution of the current program and execute an interrupt acknowledge function. See **Interrupt Signals** for additional information.
NMI	I	38	**Non-Maskable Interrupt Request** is a non-maskable input that signals the 386™ SX Microprocessor to suspend execution of the current program and execute an interrupt acknowledge function. See **Interrupt Signals** for additional information.
BUSY#	I	34	**Busy** signals a busy condition from a processor extension. See **Coprocessor Interface Signals** for additional information.
ERROR#	I	36	**Error** signals an error condition from a processor extension. See **Coprocessor Interface Signals** for additional information.
PEREQ	I	37	**Processor Extension Request** indicates that the processor has data to be transferred by the 386™ SX Microprocessor. See **Coprocessor Interface Signals** for additional information.
FLT#	I	28	**Float** is an input which forces all bidirectional and output signals, including HLDA, to the tri-state condition. This allows the electrically isolated 386SX PQFP to use ONCE (On-Circuit Emulation) method without removing it from the PCB. See **Float** for additional information.
N/C	-	20, 27, 29-31, 43-47	**No Connects** should always be left unconnected. Connection of a N/C pin may cause the processor to malfunction or be incompatible with future steppings of the 386™ SX Microprocessor.
V_{CC}	I	8-10,21,32,39 42,48,57,69, 71,84,91,97	**System Power** provides the +5V nominal DC supply input.
V_{SS}	I	2,5,11-14,22 35,41,49-50, 63,67-68, 77-78,85,98	**System Ground** provides the 0V connection from which all inputs and outputs are measured.

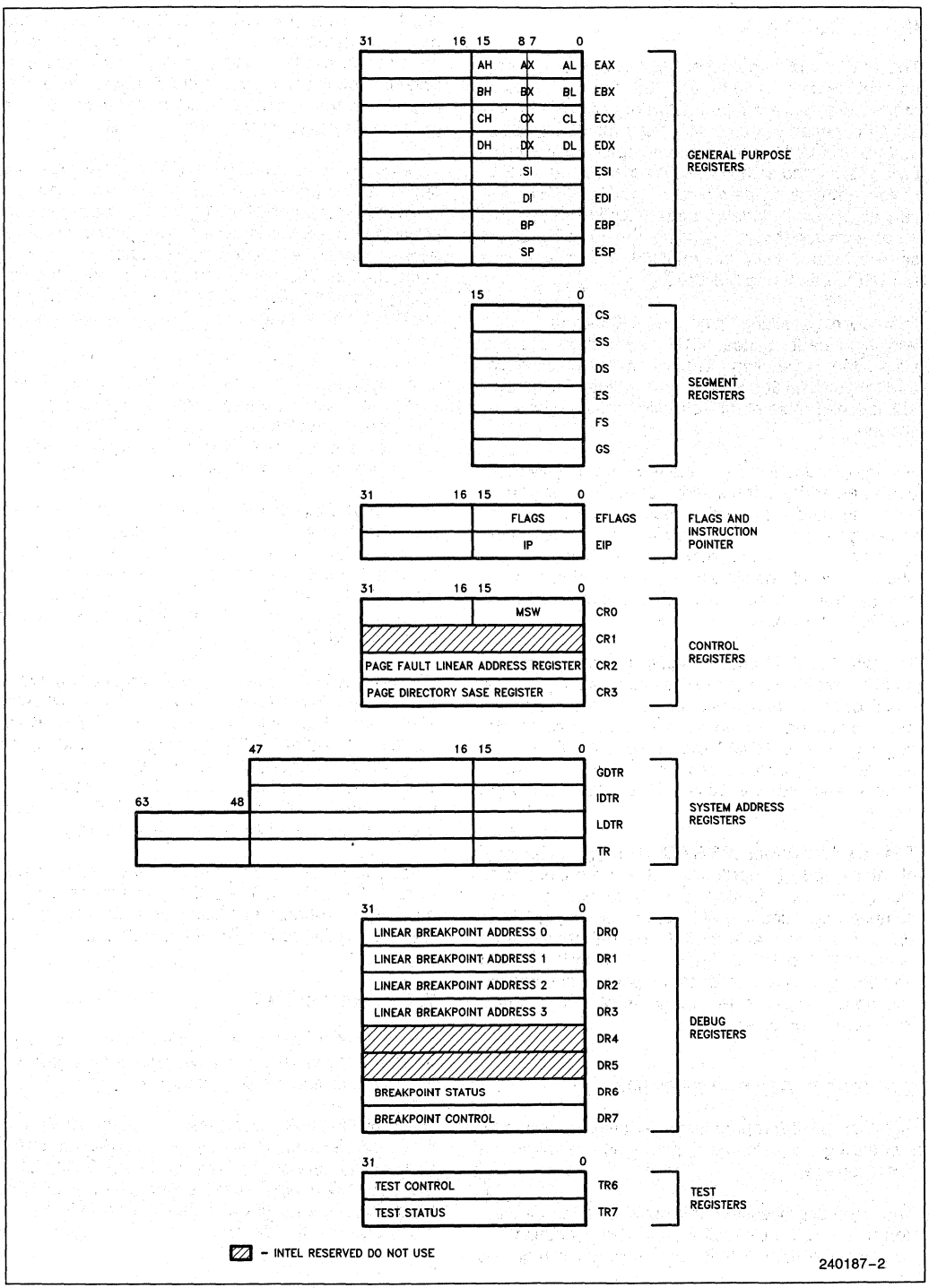


Figure 2.1. 386™ SX Microprocessor Registers

INTRODUCTION

The 386 SX Microprocessor is 100% object code compatible with the 386 DX, 286 and 8086 microprocessors. System manufacturers can provide 386 DX CPU based systems optimized for performance and 386 SX CPU based systems optimized for cost, both sharing the same operating systems and application software. Systems based on the 386 SX CPU can access the world's largest existing microcomputer software base, including the growing 32-bit software base. Only the Intel386 architecture can run UNIX, OS/2 and MS-DOS.

Instruction pipelining, high bus bandwidth, and a very high performance ALU ensure short average instruction execution times and high system throughput. The 386 SX CPU is capable of execution at sustained rates of 2.5–3.0 million instructions per second.

The integrated memory management unit (MMU) includes an address translation cache, advanced multi-tasking hardware, and a four-level hardware-enforced protection mechanism to support operating systems. The virtual machine capability of the 386 SX CPU allows simultaneous execution of applications from multiple operating systems such as MS-DOS and UNIX.

The 386 SX CPU offers on-chip testability and debugging features. Four breakpoint registers allow conditional or unconditional breakpoint traps on code execution or data accesses for powerful debugging of even ROM-based systems. Other testability features include self-test, tri-state of output buffers, and direct access to the page translation cache.

The new Low Power 386 SX CPU brings the benefits of Intel's 386 Microprocessor 32-bit architecture to the mainstream Laptop and Notebook personal computer applications. With its power saving 2 MHz sleep-mode and extended functional temperature range of 0°C to 100°C T_{CASE}, the Lower Power 386 SX CPU specifically satisfies the power consumption and heat dissipation requirements of today's small form factor computers.

2.0 BASE ARCHITECTURE

The 386 SX Microprocessor consists of a central processing unit, a memory management unit and a bus interface.

The central processing unit consists of the execution unit and the instruction unit. The execution unit contains the eight 32-bit general purpose registers which are used for both address calculation and data operations and a 64-bit barrel shifter used to speed shift, rotate, multiply, and divide operations. The instruction unit decodes the instruction opcodes and stores them in the decoded instruction queue for immediate use by the execution unit.

The memory management unit (MMU) consists of a segmentation unit and a paging unit. Segmentation allows the managing of the logical address space by providing an extra addressing component, one that allows easy code and data relocatability, and efficient sharing. The paging mechanism operates beneath and is transparent to the segmentation process, to allow management of the physical address space.

The segmentation unit provides four levels of protection for isolating and protecting applications and the operating system from each other. The hardware enforced protection allows the design of systems with a high degree of integrity.

The 386 SX Microprocessor has two modes of operation: Real Address Mode (Real Mode), and Protected Virtual Address Mode (Protected Mode). In Real Mode the 386 SX Microprocessor operates as a very fast 8086, but with 32-bit extensions if desired. Real Mode is required primarily to set up the processor for Protected Mode operation.

Within Protected Mode, software can perform a task switch to enter into tasks designated as Virtual 8086 Mode tasks. Each such task behaves with 8086 semantics, thus allowing 8086 software (an application program or an entire operating system) to execute. The Virtual 8086 tasks can be isolated and protected from one another and the host 386 SX Microprocessor operating system by use of paging.

Finally, to facilitate high performance system hardware designs, the 386 SX Microprocessor bus interface offers address pipelining and direct Byte Enable signals for each byte of the data bus.

2.1 Register Set

The 386 SX Microprocessor has thirty-four registers as shown in Figure 2-1. These registers are grouped into the following seven categories:

General Purpose Registers: The eight 32-bit general purpose registers are used to contain arithmetic and logical operands. Four of these (EAX, EBX, ECX, and EDX) can be used either in their entirety as 32-bit registers, as 16-bit registers, or split into pairs of separate 8-bit registers.

Segment Registers: Six 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data.

Flags and Instruction Pointer Registers: The two 32-bit special purpose registers in figure 2.1 record or control certain aspects of the 386 SX Microprocessor state. The EFLAGS register includes status and control bits that are used to reflect the outcome of many instructions and modify the semantics of some instructions. The Instruction Pointer, called EIP, is 32 bits wide. The Instruction Pointer controls instruction fetching and the processor automatically increments it after executing an instruction.

Control Registers: The four 32-bit control register are used to control the global nature of the 386 SX Microprocessor. The CR0 register contains bits that set the different processor modes (Protected, Real, Paging and Coprocessor Emulation). CR2 and CR3 registers are used in the paging operation.

System Address Registers: These four special registers reference the tables or segments supported by the 80286/386 SX/386 DX CPU's protection model. These tables or segments are:

GDTR (Global Descriptor Table Register),
IDTR (Interrupt Descriptor Table Register),
LDTR (Local Descriptor Table Register),
TR (Task State Segment Register).

Debug Registers: The six programmer accessible debug registers provide on-chip support for debugging. The use of the debug registers is described in Section 2.10 **Debugging Support**.

Test Registers: Two registers are used to control the testing of the RAM/CAM (Content Addressable Memories) in the Translation Lookaside Buffer portion of the 386 SX Microprocessor. Their use is discussed in **Testability**.

EFLAGS REGISTER

The flag register is a 32-bit register named EFLAGS. The defined bits and bit fields within EFLAGS, shown in Figure 2.2, control certain operations and indicate the status of the 386 SX Microprocessor. The lower 16 bits (bits 0–15) of EFLAGS contain the 16-bit flag register named FLAGS. This is the default flag register used when executing 8086, 80286, or real mode code. The functions of the flag bits are given in Table 2.1.

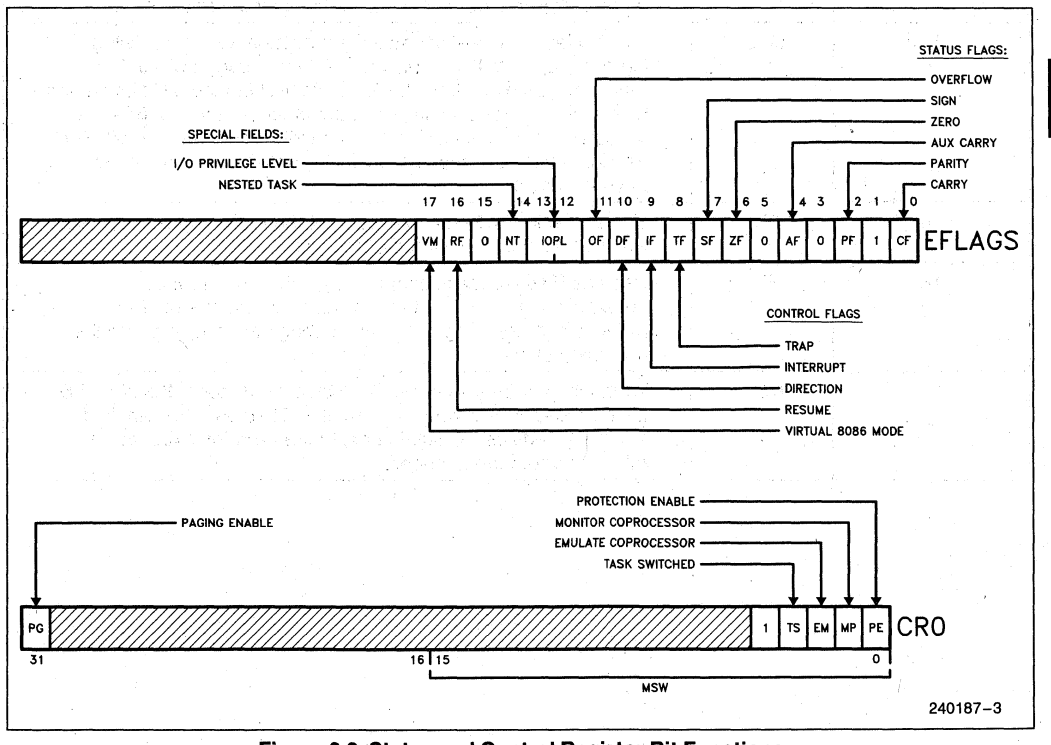


Figure 2.2. Status and Control Register Bit Functions

Table 2.1. Flag Definitions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise.
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise.
4	AF	Auxiliary Carry Flag—Set on carry from or borrow to the low order four bits of AL; cleared otherwise.
6	ZF	Zero Flag—Set if result is zero; cleared otherwise.
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative).
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-Enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto-increment (default) the appropriate index registers when cleared. Setting DF causes auto-decrement.
11	OF	Overflow Flag—Set if the operation resulted in a carry/borrow into the sign bit (high-order bit) of the result but did not result in a carry/borrow out of the high-order bit or vice-versa.
12,13	IOPL	I/O Privilege Level—Indicates the maximum Current Privilege Level (CPL) permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O permission bit map while executing in protected mode. For virtual 86 mode it indicates the maximum CPL allowing alteration of the IF bit. See Section 4.2 for a further discussion and definitions on various privilege levels.
14	NT	Nested Task—Set if the execution of the current task is nested within another task. Cleared otherwise.
16	RF	Resume Flag—Used in conjunction with debug register breakpoints. It is checked at instruction boundaries before breakpoint processing. If set, any debug fault is ignored on the next instruction.
17	VM	Virtual 8086 Mode—If set while in protected mode, the 386™ SX Microprocessor will switch to virtual 8086 operation, handling segment loads as the 8086 does, but generating exception 13 faults on privileged opcodes.

CONTROL REGISTERS

The 386 SX Microprocessor has three control registers of 32 bits, CR0, CR2 and CR3, to hold the machine state of a global nature. These registers are shown in Figures 2.1 and 2.2. The defined CR0 bits are described in Table 2.2.

Table 2.2. CR0 Definitions

Bit Position	Name	Function
0	PE	Protection mode enable—places the 386™ SX Microprocessor into protected mode. If PE is reset, the processor operates again in Real Mode. PE may be set by loading MSW or CR0. PE can be reset only by loading CR0, it cannot be reset by the LMSW instruction.
1	MP	Monitor coprocessor extension—allows WAIT instructions to cause a processor extension not present exception (number 7).
2	EM	Emulate processor extension—causes a processor extension not present exception (number 7) on ESC instructions to allow emulating a processor extension.
3	TS	Task switched—indicates the next instruction using a processor extension will cause exception 7, allowing software to test whether the current processor extension context belongs to the current task.
31	PG	Paging enable bit—is set to enable the on-chip paging unit. It is reset to disable the on-chip paging unit.

2.2 Instruction Set

The instruction set is divided into nine categories of operations:

Data Transfer
Arithmetic
Shift/Rotate
String Manipulation
Bit Manipulation
Control Transfer
High Level Language Support
Operating System Support
Processor Control

These instructions are listed in Table 9.1 **Instruction Set Clock Count Summary**.

All 386 SX Microprocessor instructions operate on either 0, 1, 2 or 3 operands; an operand resides in a register, in the instruction itself, or in memory. Most zero operand instructions (e.g CLI, STI) take only one byte. One operand instructions generally are two bytes long. The average instruction is 3.2 bytes long. Since the 386 SX Microprocessor has a 16 byte prefetch instruction queue, an average of 5 instructions will be prefetched. The use of two operands permits the following types of common instructions:

Register to Register
Memory to Register
Immediate to Register
Memory to Memory
Register to Memory
Immediate to Memory.

The operands can be either 8, 16, or 32 bits long. As a general rule, when executing code written for the 386 SX Microprocessor (32 bit code), operands are 8 or 32 bits; when executing existing 8086 or 80286 code (16-bit code), operands are 8 or 16 bits. Prefixes can be added to all instructions which override the default length of the operands (i.e. use 32-bit operands for 16-bit code, or 16-bit operands for 32-bit code).

2.3 Memory Organization

Memory on the 386 SX Microprocessor is divided into 8-bit quantities (bytes), 16-bit quantities (words), and 32-bit quantities (dwords). Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address. The address of a word or dword is the byte address of the low-order byte.

In addition to these basic data types, the 386 SX Microprocessor supports two larger units of memory: pages and segments. Memory can be divided up into one or more variable length segments, which can be swapped to disk or shared between programs. Memory can also be organized into one or more 4K byte pages. Finally, both segmentation and paging can be combined, gaining the advantages of both systems. The 386 SX Microprocessor supports both pages and segmentation in order to provide maximum flexibility to the system designer. Segmentation and paging are complementary. Segmentation is useful for organizing memory in logical modules, and as such is a tool for the application programmer, while pages are useful to the system programmer for managing the physical memory of a system.

ADDRESS SPACES

The 386 SX Microprocessor has three types of address spaces: **logical**, **linear**, and **physical**. A **logical** address (also known as a **virtual** address) consists of a selector and an offset. A selector is the contents of a segment register. An offset is formed by summing all of the addressing components (BASE, INDEX, DISPLACEMENT), discussed in section 2.4 **Addressing Modes**, into an effective address. This effective address along with the selector is known as the logical address. Since each task on the 386 SX Microprocessor has a maximum of 16K ($2^{14}-1$) selectors, and offsets can be 4 gigabytes (with paging enabled) this gives a total of 2^{46} bits, or 64 terabytes, of **logical** address space per task. The programmer sees the logical address space.

The segmentation unit translates the **logical** address space into a 32-bit **linear** address space. If the paging unit is not enabled then the 32-bit **linear** address is truncated into a 24-bit **physical** address. The **physical address** is what appears on the address pins.

The primary differences between Real Mode and Protected Mode are how the segmentation unit performs the translation of the **logical** address into the **linear** address, size of the address space, and paging capability. In Real Mode, the segmentation unit shifts the selector left four bits and adds the result to the effective address to form the **linear** address. This **linear** address is limited to 1 megabyte. In addition, real mode has no paging capability.

Protected Mode will see one of two different address spaces, depending on whether or not paging is enabled. Every selector has a **logical base** address associated with it that can be up to 32 bits in length. This 32-bit **logical base** address is added to the effective address to form a final 32-bit **linear**

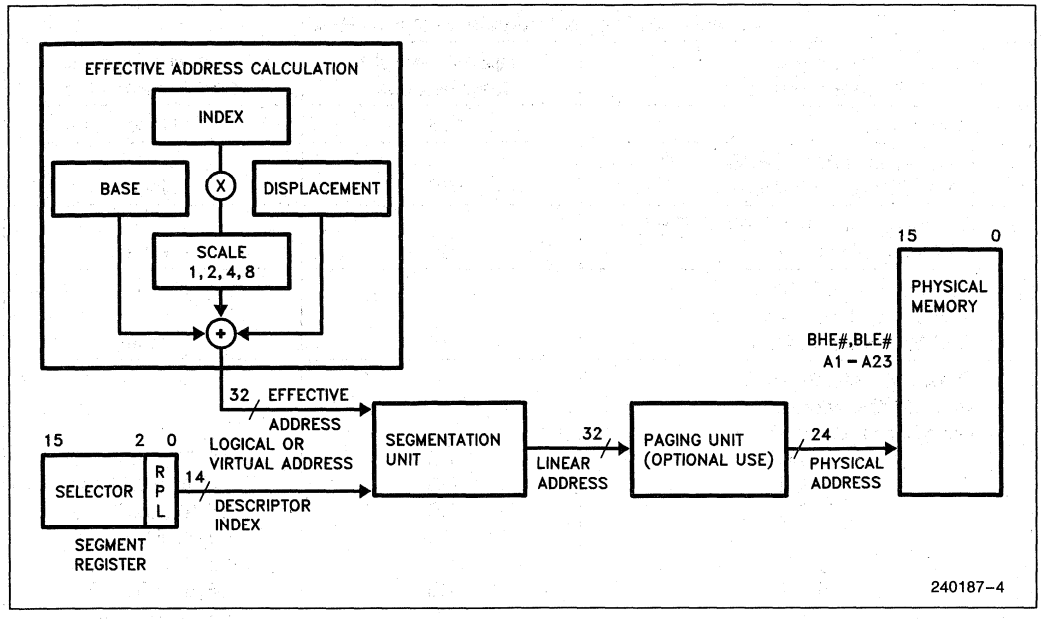


Figure 2.3. Address Translation

address. If paging is disabled this final **linear** address reflects physical memory and is truncated so that only the lower 24 bits of this address are used to address the 16 megabyte memory address space. If paging is enabled this final **linear** address reflects a 32-bit address that is translated through the paging unit to form a 16-megabyte physical address. The **logical base** address is stored in one of two operating system tables (i.e. the Local Descriptor Table or Global Descriptor Table).

Figure 2.3 shows the relationship between the various address spaces.

SEGMENT REGISTER USAGE

The main data structure used to organize memory is the segment. On the 386 SX Microprocessor, segments are variable sized blocks of linear addresses which have certain attributes associated with them. There are two main types of segments, code and data. The segments are of variable size and can be as small as 1 byte or as large as 4 gigabytes (2^{32} bits).

In order to provide compact instruction encoding and increase processor performance, instructions do not need to explicitly specify which segment register is used. The segment register is automatically chosen according to the rules of Table 2.3 (Segment Register Selection Rules). In general, data references use the selector contained in the DS register, stack references use the SS register and instruction fetches use the CS register. The contents of the Instruction Pointer provide the offset. Special segment override prefixes allow the explicit use of a given segment register, and override the implicit rules listed in Table 2.3. The override prefixes also allow the use of the ES, FS and GS segment registers.

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all 6 segments could have the base address set to zero and create a system with a four gigabyte linear address space. This creates a system where the virtual address space is the same as the linear address space. Further details of segmentation are discussed in chapter 4 **PROTECTED MODE ARCHITECTURE.**

2.4 Addressing Modes

The 386 SX Microprocessor provides a total of 8 addressing modes for instructions to specify operands. The addressing modes are optimized to allow the efficient execution of high level languages such as C and FORTRAN, and they cover the vast majority of data references needed by high-level languages.

REGISTER AND IMMEDIATE MODES

Two of the addressing modes provide for instructions that operate on register or immediate operands:

Table 2.3. Segment Register Selection Rules

Type of Memory Reference	Implied (Default) Segment Use	Segment Override Prefixes Possible
Code Fetch	CS	None
Destination of PUSH, PUSHF, INT, CALL, PUSHA Instructons	SS	None
Source of POP, POPA, POPF, IRET, RET Instructions	SS	None
Destination of STOS, MOVE, REP STOS, and REP MOVS instructions	ES	None
Other data references, with effective address using base register of:		
[EAX]	DS	CS,SS,ES,FS,GS
[EBX]	DS	CS,SS,ES,FS,GS
[ECX]	DS	CS,SS,ES,FS,GS
[EDX]	DS	CS,SS,ES,FS,GS
[ESI]	DS	CS,SS,ES,FS,GS
[EDI]	DS	CS,SS,ES,FS,GS
[EBP]	SS	CS,DS,ES,FS,GS
[ESP]	SS	CS,DS,ES,FS,GS

Register Operand Mode: The operand is located in one of the 8, 16 or 32-bit general registers.

Immediate Operand Mode: The operand is included in the instruction as part of the opcode.

32-BIT MEMORY ADDRESSING MODES

The remaining 6 modes provide a mechanism for specifying the effective address of an operand. The linear address consists of two components: the segment base address and an effective address. The effective address is calculated by summing any combination of the following three address elements (see figure 2.3):

DISPLACEMENT: an 8, 16 or 32-bit immediate value, following the instruction.

BASE: The contents of any general purpose register. The base registers are generally used by compilers to point to the start of the local variable area.

INDEX: The contents of any general purpose register except for ESP. The index registers are used to access the elements of an array, or a string of characters. The index register's value can be multiplied by a scale factor, either 1, 2, 4 or 8. The scaled index is especially useful for accessing arrays or structures.

Combinations of these 3 components make up the 6 additional addressing modes. There is no performance penalty for using any of these addressing combinations, since the effective address calculation is pipelined with the execution of other instructions. The one exception is the simultaneous use of Base and Index components which requires one additional clock.

As shown in Figure 2.4, the effective address (EA) of an operand is calculated according to the following formula:

$$EA = Base_{Register} + (Index_{Register} * scaling) + Displacement$$

1. **Direct Mode:** The operand's offset is contained as part of the instruction as an 8, 16 or 32-bit displacement.
2. **Register Indirect Mode:** A BASE register contains the address of the operand.
3. **Based Mode:** A BASE register's contents are added to a DISPLACEMENT to form the operand's offset.
4. **Scaled Index Mode:** An INDEX register's contents are multiplied by a SCALING factor, and the result is added to a DISPLACEMENT to form the operand's offset.

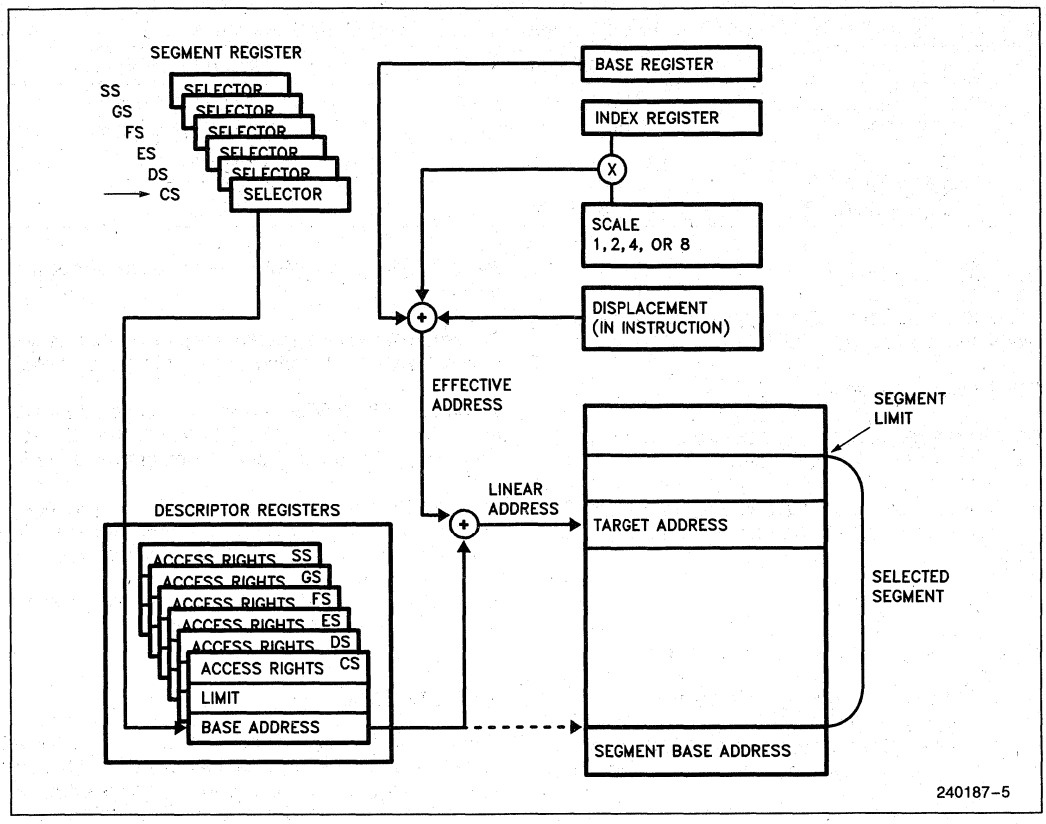


Figure 2.4. Addressing Mode Calculations

5. **Based Scaled Index Mode:** The contents of an INDEX register are multiplied by a SCALING factor, and the result is added to the contents of a BASE register to obtain the operand's offset.
6. **Based Scaled Index Mode with Displacement:** The contents of an INDEX register are multiplied by a SCALING factor, and the result is added to the contents of a BASE register and a DISPLACEMENT to form the operand's offset.

DIFFERENCES BETWEEN 16 AND 32 BIT ADDRESSES

In order to provide software compatibility with the 8086 and the 80286, the 386 SX Microprocessor can execute 16-bit instructions in Real and Protected Modes. The processor determines the size of the instructions it is executing by examining the D bit in a Segment Descriptor. If the D bit is 0 then all operand lengths and effective addresses are assumed to be 16 bits long. If the D bit is 1 then the default length for operands and addresses is 32 bits. In Real Mode the default size for operands and addresses is 16 bits.

Regardless of the default precision of the operands or addresses, the 386 SX Microprocessor is able to execute either 16 or 32-bit instructions. This is specified through the use of override prefixes. Two prefixes, the **Operand Length Prefix** and the **Address Length Prefix**, override the value of the D bit on an individual instruction basis. These prefixes are automatically added by assemblers.

The Operand Length and Address Length Prefixes can be applied separately or in combination to any instruction. The Address Length Prefix does not allow addresses over 64K bytes to be accessed in Real Mode. A memory address which exceeds 0FFFFH will result in a General Protection Fault. An Address Length Prefix only allows the use of the additional 386 SX Microprocessor addressing modes.

When executing 32-bit code, the 386 SX Microprocessor uses either 8 or 32-bit displacements, and any register can be used as base or index registers. When executing 16-bit code, the displacements are either 8 or 16-bits, and the base and index register conform to the 80286 model. Table 2.4 illustrates the differences.

Table 2.4. BASE and INDEX Registers for 16- and 32-Bit Addresses

	16-Bit Addressing	32-Bit Addressing
BASE REGISTER	BX,BP	Any 32-bit GP Register
INDEX REGISTER	SI,DI	Any 32-bit GP Register Except ESP
SCALE FACTOR	None	1, 2, 4, 8
DISPLACEMENT	0, 8, 16-bits	0, 8, 32-bits

2.5 Data Types

The 386 SX Microprocessor supports all of the data types commonly used in high level languages:

Bit: A single bit quantity.

Bit Field: A group of up to 32 contiguous bits, which spans a maximum of four bytes.

Bit String: A set of contiguous bits; on the 386 SX Microprocessor, bit strings can be up to 4 gigabits long.

Byte: A signed 8-bit quantity.

Unsigned Byte: An unsigned 8-bit quantity.

Integer (Word): A signed 16-bit quantity.

Long Integer (Double Word): A signed 32-bit quantity. All operations assume a 2's complement representation.

Unsigned Integer (Word): An unsigned 16-bit quantity.

Unsigned Long Integer (Double Word): An unsigned 32-bit quantity.

Signed Quad Word: A signed 64-bit quantity.

Unsigned Quad Word: An unsigned 64-bit quantity.

Pointer: A 16 or 32-bit offset-only quantity which indirectly references another memory location.

Long Pointer: A full pointer which consists of a 16-bit segment selector and either a 16 or 32-bit offset.

Char: A byte representation of an ASCII Alphanumeric or control character.

String: A contiguous sequence of bytes, words or dwords. A string may contain between 1 byte and 4 gigabytes

BCD: A byte (unpacked) representation of decimal digits 0–9.

Packed BCD: A byte (packed) representation of two decimal digits 0–9 storing one digit in each nibble.

When the 386 SX Microprocessor is coupled with its numerics coprocessor, the 387™ SX, then the following common floating point types are supported:

Floating Point: A signed 32, 64, or 80-bit real number representation. Floating point numbers are supported by the 387™ SX numerics coprocessor.

Figure 2.5 illustrates the data types supported by the 386 SX Microprocessor and the 387 SX.

2.6 I/O Space

The 386 SX Microprocessor has two distinct physical address spaces: physical memory and I/O. Generally, peripherals are placed in I/O space although the 386 SX Microprocessor also supports memory-mapped peripherals. The I/O space consists of 64K bytes which can be divided into 64K 8-bit ports or 32K 16-bit ports, or any combination of ports which add up to no more than 64K bytes. The 64K I/O address space refers to physical addresses rather than linear addresses since I/O instructions do not go through the segmentation or paging hardware. The M/IO# pin acts as an additional address line, thus allowing the system designer to easily determine which address space the processor is accessing.

The I/O ports are accessed by the IN and OUT instructions, with the port address supplied as an immediate 8-bit constant in the instruction or in the DX register. All 8-bit and 16-bit port addresses are zero extended on the upper address lines. The I/O instructions cause the M/IO# pin to be driven LOW. I/O port addresses 00F8H through 00FFH are reserved for use by Intel.

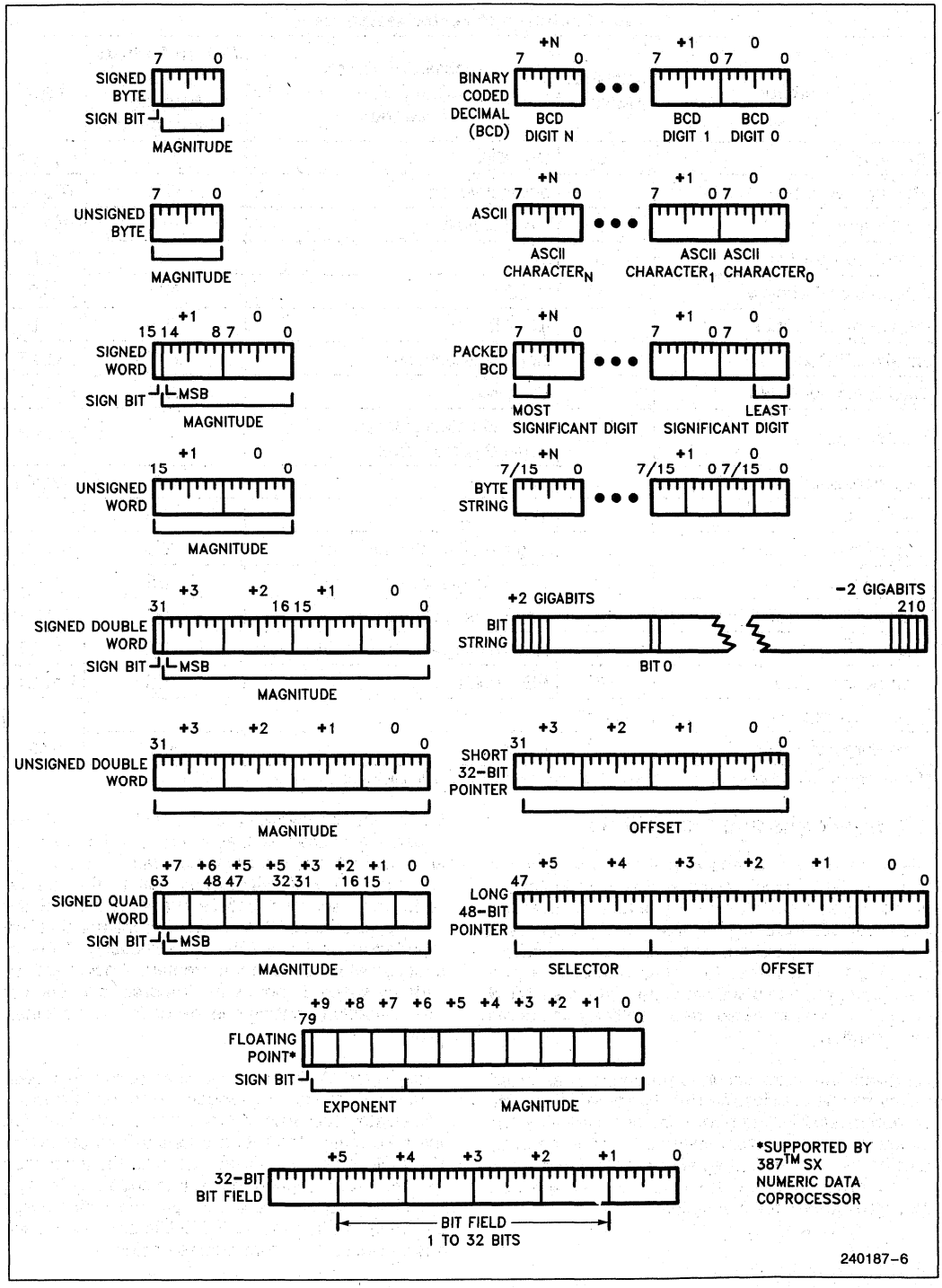


Figure 2.5. 386™ SX Microprocessor Supported Data Types

Table 2.5. Interrupt Vector Assignments

Function	Interrupt Number	Instruction Which Can Cause Exception	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	YES	FAULT
Debug Exception	1	any instruction	YES	TRAP*
NMI Interrupt	2	INT 2 or NMI	NO	NMI
One Byte Interrupt	3	INT	NO	TRAP
Interrupt on Overflow	4	INTO	NO	TRAP
Array Bounds Check	5	BOUND	YES	FAULT
Invalid OP-Code	6	Any illegal instruction	YES	FAULT
Device Not Available	7	ESC, WAIT	YES	FAULT
Double Fault	8	Any instruction that can generate an exception		ABORT
Coprocessor Segment Overrun	9	ESC	NO	ABORT
Invalid TSS	10	JMP, CALL, IRET, INT	YES	FAULT
Segment Not Present	11	Segment Register Instructions	YES	FAULT
Stack Fault	12	Stack References	YES	FAULT
General Protection Fault	13	Any Memory Reference	YES	FAULT
Page Fault	14	Any Memory Access or Code Fetch	YES	FAULT
Coprocessor Error	16	ESC, WAIT	YES	FAULT
Intel Reserved	17–32			
Two Byte Interrupt	0–255	INT n	NO	TRAP

*Some debug exceptions may report both traps on the previous instruction and faults on the next instruction.

2.7 Interrupts and Exceptions

Interrupts and exceptions alter the normal program flow in order to handle external events, report errors or exceptional conditions. The difference between interrupts and exceptions is that interrupts are used to handle asynchronous external events while exceptions handle instruction faults. Although a program can generate a software interrupt via an INT N instruction, the processor treats software interrupts as exceptions.

Hardware interrupts occur as the result of an external event and are classified into two types: maskable or non-maskable. Interrupts are serviced after the execution of the current instruction. After the interrupt handler is finished servicing the interrupt, execution proceeds with the instruction immediately **after** the interrupted instruction.

Exceptions are classified as faults, traps, or aborts, depending on the way they are reported and whether or not restart of the instruction causing the exception is supported. **Faults** are exceptions that are detected and serviced **before** the execution of the faulting instruction. **Traps** are exceptions that are reported immediately **after** the execution of the instruction which caused the problem. **Aborts** are exceptions which do not permit the precise location of the instruction causing the exception to be determined.

Thus, when an interrupt service routine has been completed, execution proceeds from the instruction immediately following the interrupted instruction. On the other hand, the return address from an exception fault routine will always point to the instruction causing the exception and will include any leading instruction prefixes. Table 2.5 summarizes the possible interrupts for the 386 SX Microprocessor and shows where the return address points to.

The 386 SX Microprocessor has the ability to handle up to 256 different interrupts/exceptions. In order to service the interrupts, a table with up to 256 interrupt vectors must be defined. The interrupt vectors are simply pointers to the appropriate interrupt service routine. In Real Mode, the vectors are 4-byte quantities, a Code Segment plus a 16-bit offset; in Protected Mode, the interrupt vectors are 8 byte quantities, which are put in an Interrupt Descriptor Table. Of the 256 possible interrupts, 32 are reserved for use by Intel and the remaining 224 are free to be used by the system designer.

INTERRUPT PROCESSING

When an interrupt occurs, the following actions happen. First, the current program address and Flags are saved on the stack to allow resumption of the interrupted program. Next, an 8-bit vector is supplied to the 386 SX Microprocessor which identifies the appropriate entry in the interrupt table. The table contains the starting address of the interrupt service routine. Then, the user supplied interrupt service routine is executed. Finally, when an IRET instruction is executed the old processor state is restored and program execution resumes at the appropriate instruction.

The 8-bit interrupt vector is supplied to the 386 SX Microprocessor in several different ways: exceptions supply the interrupt vector internally; software INT instructions contain or imply the vector; maskable hardware interrupts supply the 8-bit vector via the interrupt acknowledge bus sequence. Non-Maskable hardware interrupts are assigned to interrupt vector 2.

Maskable Interrupt

Maskable interrupts are the most common way to respond to asynchronous external hardware events. A hardware interrupt occurs when the INTR is pulled HIGH and the Interrupt Flag bit (IF) is enabled. The processor only responds to interrupts between instructions (string instructions have an 'interrupt window' between memory moves which allows interrupts during long string moves). When an interrupt occurs the processor reads an 8-bit vector supplied by the hardware which identifies the source of the interrupt (one of 224 user defined interrupts).

Interrupts through interrupt gates automatically reset IF, disabling INTR requests. Interrupts through Trap Gates leave the state of the IF bit unchanged. Interrupts through a Task Gate change the IF bit according to the image of the EFLAGs register in the task's Task State Segment (TSS). When an IRET instruction is executed, the original state of the IF bit is restored.

Non-Maskable Interrupt

Non-maskable interrupts provide a method of servicing very high priority interrupts. When the NMI input is pulled HIGH it causes an interrupt with an internally supplied vector value of 2. Unlike a normal hardware interrupt, no interrupt acknowledgment sequence is performed for an NMI.

While executing the NMI servicing procedure, the 386 SX Microprocessor will not service any further NMI request or INT requests until an interrupt return (IRET) instruction is executed or the processor is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. The IF bit is cleared at the beginning of an NMI interrupt to inhibit further INTR interrupts.

Software Interrupts

A third type of interrupt/exception for the 386 SX Microprocessor is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the n^{th} vector in the interrupt table.

A special case of the two byte software interrupt INT n is the one byte INT 3, or breakpoint interrupt. By inserting this one byte instruction in a program, the user can set breakpoints in his program as a debugging tool.

A final type of software interrupt is the single step interrupt. It is discussed in **Single Step Trap**.

INTERRUPT AND EXCEPTION PRIORITIES

Interrupts are externally generated events. Maskable Interrupts (on the INTR input) and Non-Maskable Interrupts (on the NMI input) are recognized at instruction boundaries. When NMI and maskable INTR are **both** recognized at the **same** instruction boundary, the 386 SX Microprocessor invokes the NMI service routine first. If maskable interrupts are still enabled after the NMI service routine has been invoked, then the 386 SX Microprocessor will invoke the appropriate interrupt service routine.

As the 386 SX Microprocessor executes instructions, it follows a consistent cycle in checking for exceptions, as shown in Table 2.6. This cycle is repeated as each instruction is executed, and occurs in parallel with instruction decoding and execution.

INSTRUCTION RESTART

The 386 SX Microprocessor fully supports restarting all instructions after Faults. If an exception is detected in the instruction to be executed (exception categories 4 through 10 in Table 2.6), the 386 SX Microprocessor invokes the appropriate exception service routine. The 386 SX Microprocessor is in a state that permits restart of the instruction, for all cases but those given in Table 2.7. Note that all such cases will be avoided by a properly designed operating system.

Table 2.6. Sequence of Exception Checking

Consider the case of the 386™ SX Microprocessor having just completed an instruction. It then performs the following checks before reaching the point where the next instruction is completed:

1. Check for Exception 1 Traps from the instruction just completed (single-step via Trap Flag, or Data Breakpoints set in the Debug Registers).
2. Check for external NMI and INTR.
3. Check for Exception 1 Faults in the next instruction (Instruction Execution Breakpoint set in the Debug Registers for the next instruction).
4. Check for Segmentation Faults that prevented fetching the entire next instruction (exceptions 11 or 13).
5. Check for Page Faults that prevented fetching the entire next instruction (exception 14).
6. Check for Faults decoding the next instruction (exception 6 if illegal opcode; exception 6 if in Real Mode or in Virtual 8086 Mode and attempting to execute an instruction for Protected Mode only; or exception 13 if instruction is longer than 15 bytes, or privilege violation in Protected Mode (i.e. not at IOPL or at CPL = 0).
7. If WAIT opcode, check if TS = 1 and MP = 1 (exception 7 if both are 1).
8. If ESCape opcode for numeric coprocessor, check if EM = 1 or TS = 1 (exception 7 if either are 1).
9. If WAIT opcode or ESCape opcode for numeric coprocessor, check ERROR# input signal (exception 16 if ERROR# input is asserted).
10. Check in the following order for each memory reference required by the instruction:
    a. Check for Segmentation Faults that prevent transferring the entire memory quantity (exceptions 11, 12, 13).
    b. Check for Page Faults that prevent transferring the entire memory quantity (exception 14).

NOTE:
Segmentation exceptions are generated before paging exceptions.

Table 2.7. Conditions Preventing Instruction Restart

1. An instruction causes a task switch to a task whose Task State Segment is **partially** 'not present' (An entirely 'not present' TSS is restartable). Partially present TSS's can be avoided either by keeping the TSS's of such tasks present in memory, or by aligning TSS segments to reside entirely within a single 4K page (for TSS segments of 4K bytes or less).
2. A coprocessor operand wraps around the top of a 64K-byte segment or a 4G-byte segment, and spans three pages, and the page holding the middle portion of the operand is 'not present'. This condition can be avoided by starting **at a page boundary** any segments containing coprocessor operands if the segments are approximately 64K-200 bytes or larger (i.e. large enough for wraparound of the coprocessor operand to possibly occur).

Note that these conditions are avoided by using the operating system designs mentioned in this table.

Table 2.8. Register Values after Reset

Flag Word (EFLAGS)	uuuu0002H	Note 1
Machine Status Word (CR0)	uuuuuu10H	
Instruction Pointer (EIP)	0000FFF0H	
Code Segment (CS)	F000H	Note 2
Data Segment (DS)	0000H	Note 3
Stack Segment (SS)	0000H	
Extra Segment (ES)	0000H	Note 3
Extra Segment (FS)	0000H	
Extra Segment (GS)	0000H	
EAX register	0000H	Note 4
EDX register	component and stepping ID	Note 5
All other registers	undefined	Note 6

NOTES:
1. EFLAG Register. The upper 14 bits of the EFLAGS register are undefined, all defined flag bits are zero.
2. The Code Segment Register (CS) will have its Base Address set to 0FFFF0000H and Limit set to 0FFFFH.
3. The Data and Extra Segment Registers (DS, ES) will have their Base Address set to 000000000H and Limit set to 0FFFFH.
4. If self-test is selected, the EAX register should contain a 0 value. If a value of 0 is not found then the self-test has detected a flaw in the part.
5. EDX register always holds component and stepping identifier.
6. All undefined bits are Intel Reserved and should not be used.

DOUBLE FAULT

A Double Fault (exception 8) results when the processor attempts to invoke an exception service routine for the segment exceptions (10, 11, 12 or 13), but in the process of doing so detects an exception **other than** a Page Fault (exception 14).

One other cause of generating a Double Fault is the 386 SX Microprocessor detecting any other exception when it is attempting to invoke the Page Fault (exception 14) service routine (for example, if a Page Fault is detected when the 386 SX Microprocessor attempts to invoke the Page Fault service routine). Of course, in any functional system, not only in 386 SX Microprocessor-based systems, the entire page fault service routine must remain 'present' in memory.

2.8 Reset and Initialization

When the processor is initialized or Reset the registers have the values shown in Table 2.8. The 386 SX Microprocessor will then start executing instructions near the top of physical memory, at location 0FFFFF0H. When the first Intersegment Jump or Call is executed, address lines A_{20}–A_{23} will drop LOW for CS-relative memory cycles, and the 386 SX Microprocessor will only execute instructions in the lower one megabyte of physical memory. This allows the system designer to use a shadow ROM at the top of physical memory to initialize the system and take care of Resets.

RESET forces the 386 SX Microprocessor to terminate all execution and local bus activity. No instruction execution or bus activity will occur as long as Reset is active. Between 350 and 450 CLK2 periods after Reset becomes inactive, the 386 SX Microprocessor will start executing instructions at the top of physical memory.

5

2.9 Testability

The 386 SX Microprocessor, like the 386 Microprocessor, offers testability features which include a self-test and direct access to the page translation cache.

SELF-TEST

The 386 SX Microprocessor has the capability to perform a self-test. The self-test checks the function of all of the Control ROM and most of the non-random logic of the part. Approximately one-half of the 386 SX Microprocessor can be tested during self-test.

Self-Test is initiated on the 386 SX Microprocessor when the RESET pin transitions from HIGH to LOW, and the BUSY# pin is LOW. The self-test takes about 2^{20} clocks, or approximately 33 milliseconds with a 16 MHz 386 SX CPU. At the completion of self-test the processor performs reset and begins normal operation. The part has successfully passed self-test if the contents of the EAX are zero. If the results of the EAX are not zero then the self-test has detected a flaw in the part.

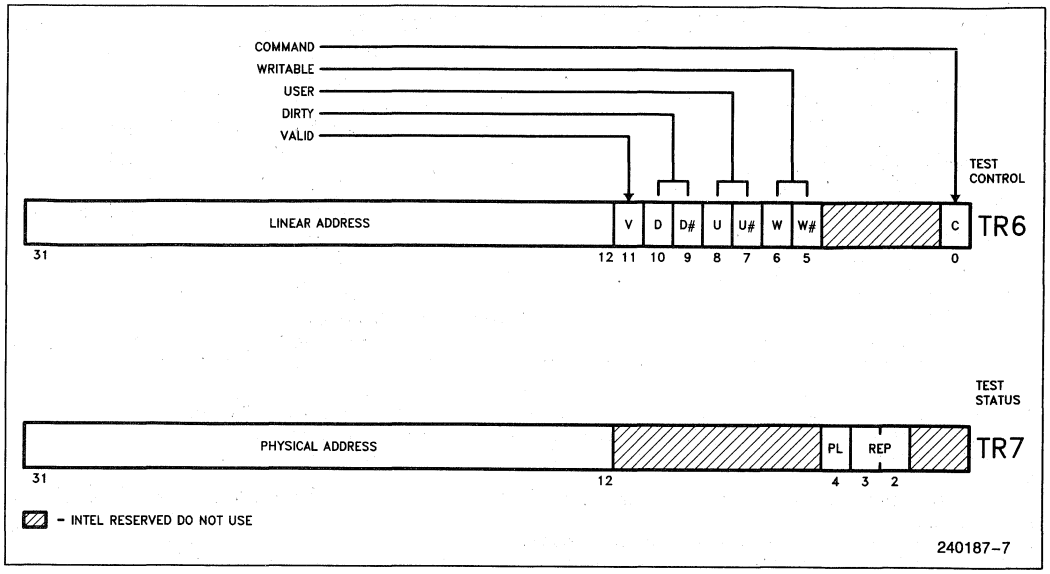


Figure 2.6. Test Registers

TLB TESTING

The 386 SX Microprocessor also provides a mechanism for testing the Translation Lookaside Buffer (TLB) if desired. This particular mechanism may not be continued in the same way in future processors.

There are two TLB testing operations: 1) writing entries into the TLB, and, 2) performing TLB lookups. Two Test Registers, shown in Figure 2.6, are provided for the purpose of testing. TR6 is the "test command register", and TR7 is the "test data register". For a more detailed explanation of testing the TLB, see the 386™ SX Microprocessor Programmer's Reference Manual.

2.10 Debugging Support

The 386 SX Microprocessor provides several features which simplify the debugging process. The three categories of on-chip debugging aids are:

1. The code execution breakpoint opcode (0CCH).
2. The single-step capability provided by the TF bit in the flag register.
3. The code and data breakpoint capability provided by the Debug Registers DR0-3, DR6, and DR7.

BREAKPOINT INSTRUCTION

A single-byte software interrupt (Int 3) breakpoint instruction is available for use by software debuggers. The breakpoint opcode is 0CCh, and generates an exception 3 trap when executed.

SINGLE-STEP TRAP

If the single-step flag (TF, bit 8) in the EFLAG register is found to be set at the end of an instruction, a single-step exception occurs. The single-step exception is auto vectored to exception number 1.

DEBUG REGISTERS

The Debug Registers are an advanced debugging feature of the 386 SX Microprocessor. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution breakpoint can be placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT 3 breakpoint opcode.

The 386 SX Microprocessor contains six Debug Registers, consisting of four breakpoint address registers and two breakpoint control registers. Initially after reset, breakpoints are in the disabled state; therefore, no breakpoints will occur unless the debug registers are programmed. Breakpoints set up in the Debug Registers are auto-vectored to exception 1. Figure 2.7 shows the breakpoint status and control registers.

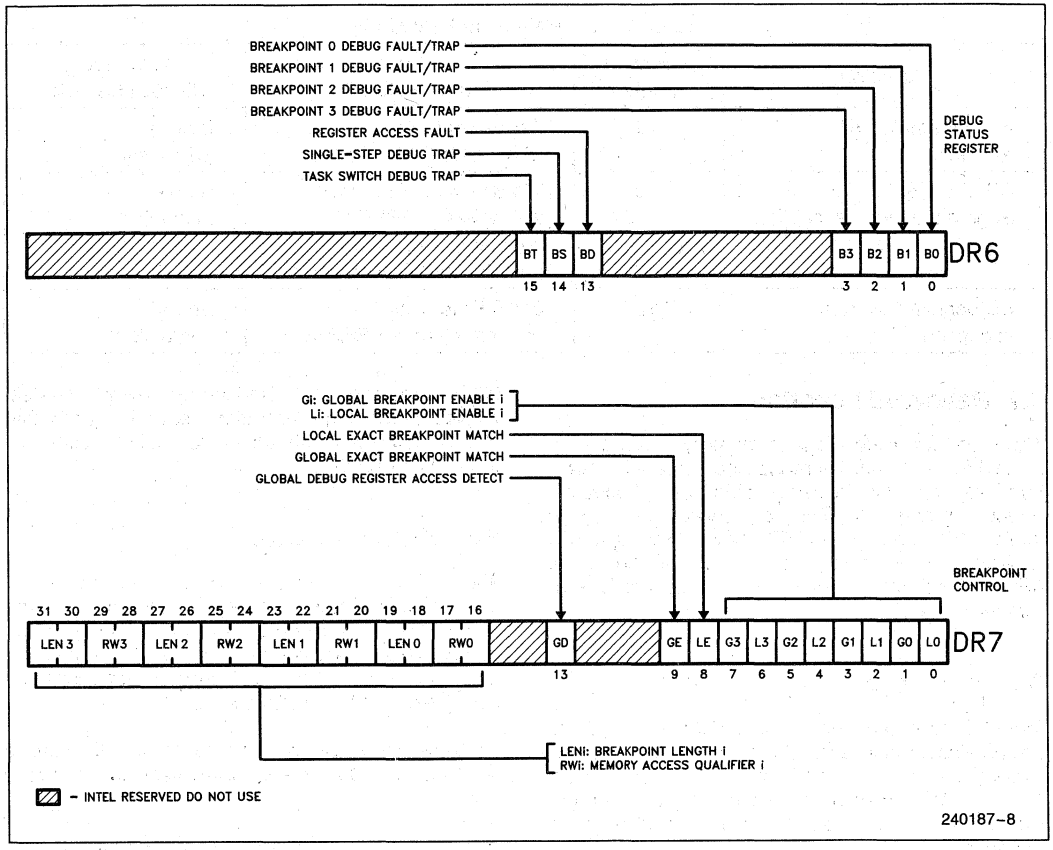


Figure 2.7. Debug Registers

3.0 REAL MODE ARCHITECTURE

When the processor is reset or powered up it is initialized in Real Mode. Real Mode has the same base architecture as the 8086, but allows access to the 32-bit register set of the 386 SX Microprocessor. The addressing mechanism, memory size, and interrupt handling are all identical to the Real Mode on the 80286.

The default operand size in Real Mode is 16 bits, as in the 8086. In order to use the 32-bit registers and addressing modes, override prefixes must be used. In addition, the segment size on the 386 SX Microprocessor in Real Mode is 64K bytes so 32-bit addresses must have a value less then 0000FFFFH. The primary purpose of Real Mode is to set up the processor for Protected Mode operation.

3.1 Memory Addressing

In Real Mode the linear addresses are the same as physical addresses (paging is not allowed). Physical addresses are formed in Real Mode by adding the contents of the appropriate segment register which is shifted left by four bits to an effective address. This addition results in a 20-bit physical address or a 1 megabyte address space. Since segment registers are shifted left by 4 bits, Real Mode segments always start on 16-byte boundaries.

All segments in Real Mode are exactly 64K bytes long, and may be read, written, or executed. The 386 SX Microprocessor will generate an exception 13 if a data operand or instruction fetch occurs past the end of a segment.

Table 3.1. Exceptions in Real Mode

Function	Interrupt Number	Related Instructions	Return Address Location
Interrupt table limit too small	8	INT vector is not within table limit	Before Instruction
CS, DS, ES, FS, GS Segment overrun exception	13	Word memory reference with offset = 0FFFFH. an attempt to execute past the end of CS segment.	Before Instruction
SS Segment overrun exception	12	Stack Reference beyond offset = 0FFFFH	Before Instruction

3.2 Reserved Locations

There are two fixed areas in memory which are reserved in Real address mode: the system initialization area and the interrupt table area. Locations 00000H through 003FFH are reserved for interrupt vectors. Each one of the 256 possible interrupts has a 4-byte jump vector reserved for it. Locations 0FFFFF0H through 0FFFFFFH are reserved for system initialization.

3.3 Interrupts

Many of the exceptions discussed in section 2.7 are not applicable to Real Mode operation; in particular, exceptions 10, 11 and 14 do not occur in Real Mode. Other exceptions have slightly different meanings in Real Mode; Table 3.1 identifies these exceptions.

3.4 Shutdown and Halt

The HLT instruction stops program execution and prevents the processor from using the local bus until restarted. Either NMI, FLT#, INTR with interrupts enabled (IF = 1), or RESET will force the 386 SX Microprocessor out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

Shutdown will occur when a severe error is detected that prevents further processing. In Real Mode, shutdown can occur under two conditions:

1. An interrupt or an exception occurs (Exceptions 8 or 13) and the interrupt vector is larger than the Interrupt Descriptor Table.
2. A CALL, INT or PUSH instruction attempts to wrap around the stack segment when SP is not even.

An NMI input can bring the processor out of shutdown if the Interrupt Descriptor Table limit is large enough to contain the NMI interrupt vector (at least 000FH) and the stack has enough room to contain the vector and flag information (i.e. SP is greater that 0005H). Otherwise, shutdown can only be exited by a processor reset.

3.5 LOCK operation

The LOCK prefix on the 386 SX Microprocessor, even in Real Mode, is more restrictive than on the 80286. This is due to the addition of paging on the 386 SX Microprocessor in Protected Mode and Virtual 8086 Mode. The LOCK prefix is not supported during repeat string instructions.

The only instruction forms where the LOCK prefix is legal on the 386 SX Microprocessor are shown in Table 3.2.

Table 3.2. Legal Instructions for the LOCK Prefix

Opcode	Operands (Dest, Source)
BIT Test and SET/RESET /COMPLEMENT	Mem, Reg/Immediate
XCHG	Reg, Mem
XCHG	Mem, Reg
ADD, OR, ADC, SBB, AND, SUB, XOR	Mem, Reg/Immediate
NOT, NEG, INC, DEC	Mem

An exception 6 will be generated if a LOCK prefix is placed before any instruction form or opcode not listed above. The LOCK prefix allows indivisible read/modify/write operations on memory operands using the instructions above.

The LOCK prefix is not IOPL-sensitive on the 386 SX Microprocessor. The LOCK prefix can be used at any privilege level, but only on the instruction forms listed in Table 3.2.

4.0 PROTECTED MODE ARCHITECTURE

The complete capabilities of the 386 SX Microprocessor are unlocked when the processor operates in Protected Virtual Address Mode (Protected Mode). Protected Mode vastly increases the linear address space to four gigabytes (2^{32} bytes) and allows the running of virtual memory programs of almost unlimited size (64 terabytes (2^{46} bytes)). In addition, Protected Mode allows the 386 SX Microprocessor to run all of the existing 386 DX CPU (using only 16 megabytes of physical memory), 80286 and 8086 CPU's software, while providing a sophisticated memory management and a hardware-assisted protection mechanism. Protected Mode allows the use of additional instructions specially optimized for supporting multitasking operating systems. The base architecture of the 386 SX Microprocessor remains the same; the registers, instructions, and addressing modes described in the previous sections are retained. The main difference between Protected Mode and Real Mode from a programmer's viewpoint is the increased address space and a different addressing mechanism.

4.1 Addressing Mechanism

Like Real Mode, Protected Mode uses two components to form the logical address; a 16-bit selector is used to determine the linear base address of a segment, the base address is added to a 32-bit effective address to form a 32-bit linear address. The linear address is then either used as a 24-bit physical address, or if paging is enabled the paging mechanism maps the 32-bit linear address into a 24-bit physical address.

The difference between the two modes lies in calculating the base address. In Protected Mode, the selector is used to specify an index into an operating system defined table (see Figure 4.1). The table contains the 32-bit base address of a given segment. The physical address is formed by adding the base address obtained from the table to the offset.

Paging provides an additional memory management mechanism which operates only in Protected Mode. Paging provides a means of managing the very large segments of the 386 SX Microprocessor, as paging operates beneath segmentation. The page mechanism translates the protected linear address which comes from the segmentation unit into a physical address. Figure 4.2 shows the complete 386 SX Microprocessor addressing mechanism with paging enabled.

4.2 Segmentation

Segmentation is one method of memory management. Segmentation provides the basis for protection. Segments are used to encapsulate regions of memory which have common attributes. For example, all of the code of a given program could be contained in a segment, or an operating system table may reside in a segment. All information about each segment is stored in an 8 byte data structure called a descriptor. All of the descriptors in a system are contained in descriptor tables which are recognized by hardware.

TERMINOLOGY

The following terms are used throughout the discussion of descriptors, privilege levels and protection:

PL: Privilege Level—One of the four hierarchical privilege levels. Level 0 is the most privileged level and level 3 is the least privileged.

RPL: Requestor Privilege Level—The privilege level of the original supplier of the selector. RPL is determined by the least two significant bits of a selector.

DPL: Descriptor Privilege Level—This is the least privileged level at which a task may access that descriptor (and the segment associated with that descriptor). Descriptor Privilege Level is determined by bits 6:5 in the Access Right Byte of a descriptor.

CPL: Current Privilege Level—The privilege level at which a task is currently executing, which equals the privilege level of the code segment being executed. CPL can also be determined by examining the lowest 2 bits of the CS register, except for conforming code segments.

EPL: Effective Privilege Level—The effective privilege level is the least privileged of the RPL and the DPL. EPL is the numerical maximum of RPL and DPL.

Task: One instance of the execution of a program. Tasks are also referred to as processes.

DESCRIPTOR TABLES

The descriptor tables define all of the segments which are used in a 386 SX Microprocessor system. There are three types of tables which hold descriptors: the Global Descriptor Table, Local Descriptor Table, and the Interrupt Descriptor Table. All of the tables are variable length memory arrays and can vary in size from 8 bytes to 64K bytes. Each table can hold up to 8192 8-byte descriptors. The upper 13 bits of a selector are used as an index into the descriptor table. The tables have registers associated with them which hold the 32-bit linear base address and the 16-bit limit of each table.

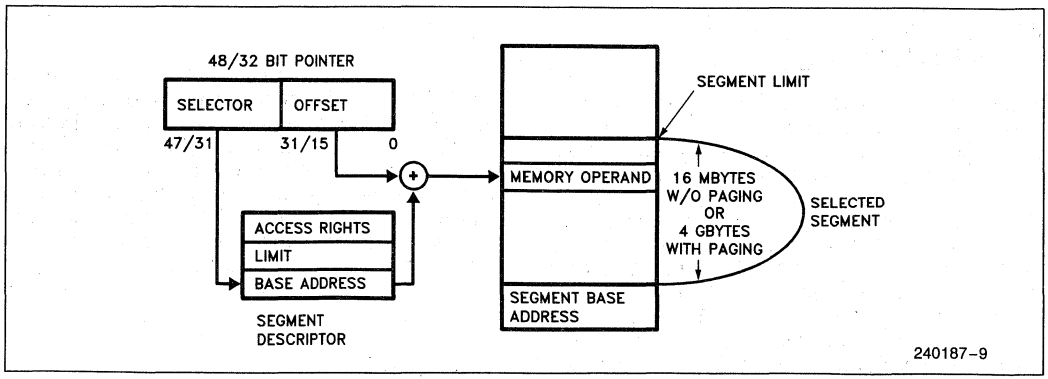


Figure 4.1. Protected Mode Addressing

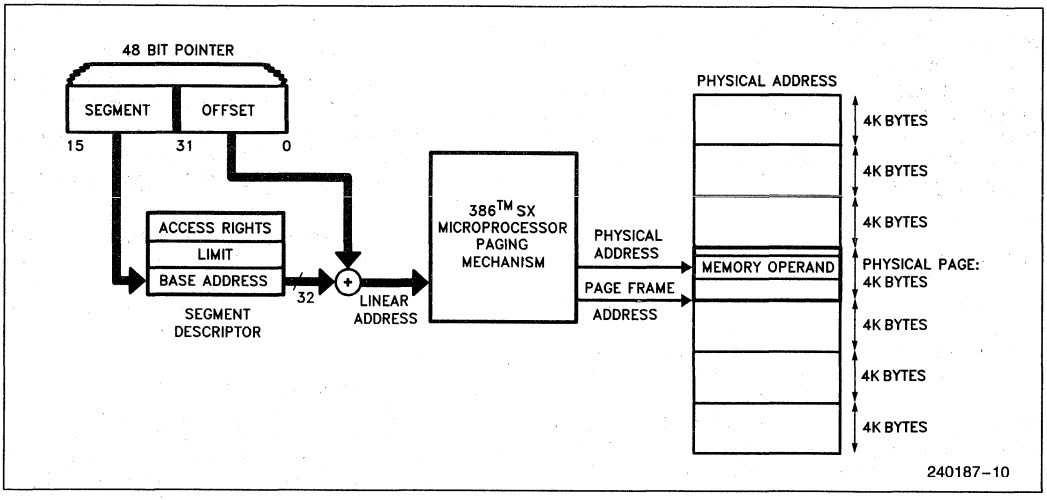


Figure 4.2. Paging and Segmentation

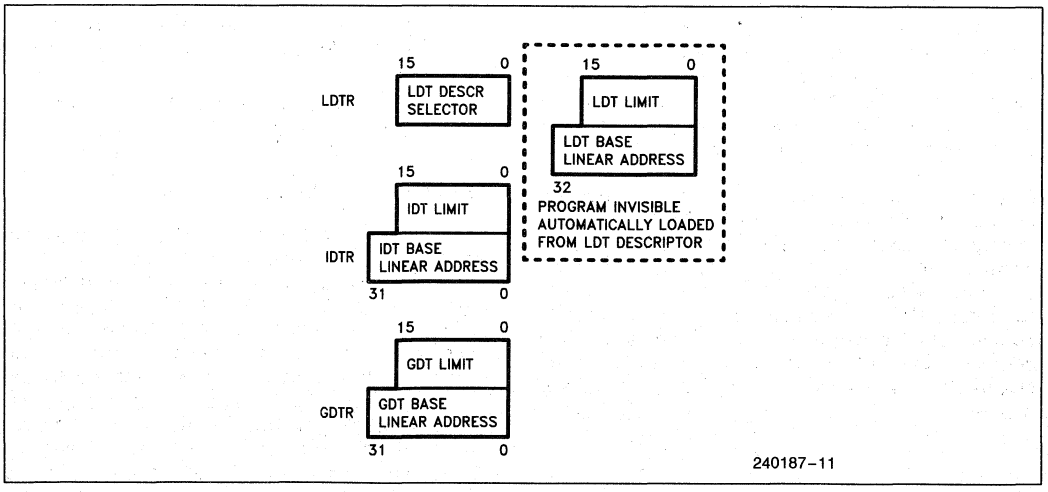


Figure 4.3. Descriptor Table Registers

Each of the tables has a register associated with it: GDTR, LDTR, and IDTR; see Figure 2.1. The LGDT, LLDT, and LIDT instructions load the base and limit of the Global, Local, and Interrupt Descriptor Tables into the appropriate register. The SGDT, SLDT, and SIDT store the base and limit values. These are privileged instructions.

Global Descriptor Table

The Global Descriptor Table (GDT) contains descriptors which are available to all of the tasks in a system. The GDT can contain any type of segment descriptor except for interrupt and trap descriptors. Every 386 SX CPU system contains a GDT.

The first slot of the Global Descriptor Table corresponds to the null selector and is not used. The null selector defines a null pointer value.

Local Descriptor Table

LDTs contain descriptors which are associated with a given task. Generally, operating systems are designed so that each task has a separate LDT. The LDT may contain only code, data, stack, task gate, and call gate descriptors. LDTs provide a mechanism for isolating a given task's code and data segments from the rest of the operating system, while the GDT contains descriptors for segments which are common to all tasks. A segment cannot be accessed by a task if its segment descriptor does not exist in either the current LDT or the GDT. This provides both isolation and protection for a task's segments while still allowing global data to be shared among tasks.

Unlike the 6-byte GDT or IDT registers which contain a base address and limit, the visible portion of the LDT register contains only a 16-bit selector. This selector refers to a Local Descriptor Table descriptor in the GDT (see figure 2.1).

Interrupt Descriptor Table

The third table needed for 386 SX Microprocessor systems is the Interrupt Descriptor Table. The IDT contains the descriptors which point to the location of the up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates, and trap gates. The IDT should be at least 256 bytes in size in order to hold the descriptors for the 32 Intel Reserved Interrupts. Every interrupt used by a system must have an entry in the IDT. The IDT entries are referenced by INT instructions, external interrupt vectors, and exceptions.

DESCRIPTORS

The object to which the segment selector points to is called a descriptor. Descriptors are eight byte quantities which contain attributes about a given region of linear address space. These attributes include the 32-bit base linear address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, the default size of the operands (16-bit or 32-bit), and the type of segment. All of the attribute information about a segment is contained in 12 bits in the segment descriptor. Figure 4.4 shows the general format of a descriptor. All segments on the 386 SX Microprocessor have three attribute fields in common: the P bit, the DPL bit, and the S bit. The P

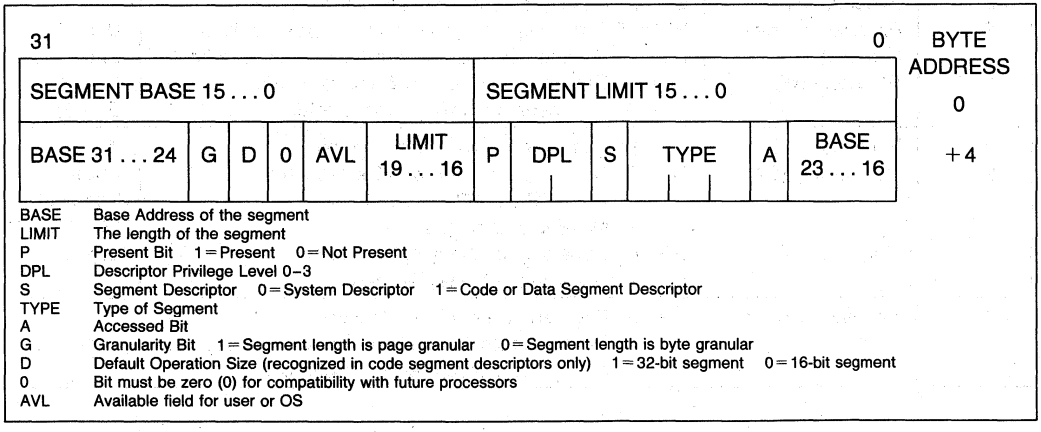


Figure 4.4. Segment Descriptors

(Present) Bit is 1 if the segment is loaded in physical memory. If P=0 then any attempt to access this segment causes a not present exception (exception 11). The Descriptor Privilege Level, DPL, is a two bit field which specifies the protection level, 0–3, associated with a segment.

The 386 SX Microprocessor has two main categories of segments: system segments and non-system segments (for code and data). The segment bit, S, determines if a given segment is a system segment or a code or data segment. If the S bit is 1 then the segment is either a code or data segment; if it is 0 then the segment is a system segment.

Code and Data Descriptors (S = 1)

Figure 4.5 shows the general format of a code and data descriptor and Table 4.1 illustrates how the bits in the Access Right Byte are interpreted.

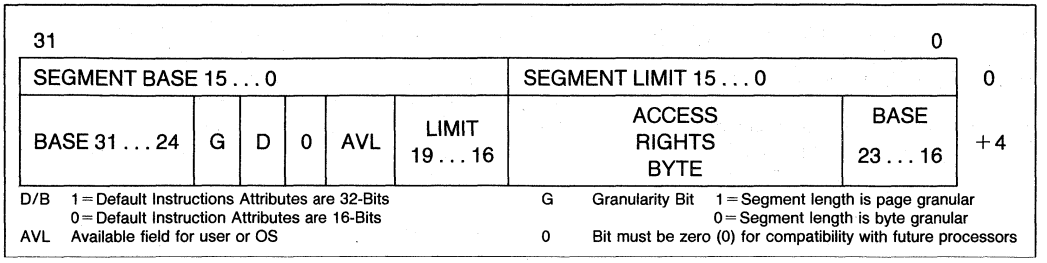
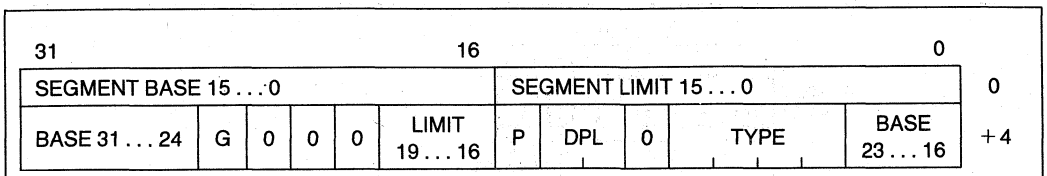


Figure 4.5. Code and Data Descriptors

Table 4.1. Access Rights Byte Definition for Code and Data Descriptors

Bit Position	Name	Function
7	Present (P)	P = 1 Segment is mapped into physical memory. P = 0 No mapping to physical memory exists, base and limt are not used.
6–5	Descriptor Privilege Level (DPL)	Segment privilege attribute used in privilege tests.
4	Segment Descriptor (S)	S = 1 Code or Data (includes stacks) segment descriptor S = 0 System Segment Descriptor or Gate Descriptor
3	Executable (E)	E = 0 Descriptor type is data segment: (If Data Segment (S = 1, E = 0))
2	Expansion Direction (ED)	ED = 0 Expand up segment, offsets must be ≤ limit. ED = 1 Expand down segment, offsets must be > limit.
1	Writeable (W)	W = 0 Data segment may not be written into. W = 1 Data segment may be written into.
3	Executable (E)	E = 1 Descriptor type is code segment: (If Code Segment (S = 1, E = 1))
2	Conforming (C)	C = 1 Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.
1	Readable (R)	R = 0 Code segment may not be read. R = 1 Code segment may be read.
0	Accessed (A)	A = 0 Segment has not been accessed. A = 1 Segment selector has been loaded into segment register or used by selector test instructions.



Type	Defines
0	Invalid
1	Available 80286 TSS
2	LDT
3	Busy 80286 TSS
4	80286 Call Gate
5	Task Gate (for 80286 or 386™ SX Microprocessor Task)
6	80286 Interrupt Gate
7	80286 Trap Gate

Type	Defines
8	Invalid
9	Available 386™ SX Microprocessor TSS
A	Undefined (Intel Reserved)
B	Busy 386™ SX Microprocessor TSS
C	386™ SX Microprocessor Call Gate
D	Undefined (Intel Reserved)
E	386™ SX Microprocessor Interrupt Gate
F	386™ SX Microprocessor Trap Gate

Figure 4.6. System Descriptors

Code and data segments have several descriptor fields in common. The accessed bit, A, is set whenever the processor accesses a descriptor. The granularity bit, G, specifies if a segment length is byte-granular or page-granular.

System Descriptor Formats (S = 0)

System segments describe information about operating system tables, tasks, and gates. Figure 4.6 shows the general format of system segment descriptors, and the various types of system segments. 386 SX system descriptors (which are the same as 386 DX CPU system descriptors) contain a 32-bit base linear address and a 20-bit segment limit. 80286 system descriptors have a 24-bit base address and a 16-bit segment limit. 80286 system descriptors are identified by the upper 16 bits being all zero.

Differences Between 386™ SX Microprocessor and 80286 Descriptors

In order to provide operating system compatibility with the 80286 the 386 SX CPU supports all of the 80286 segment descriptors. The 80286 system segment descriptors contain a 24-bit base address and 16-bit limit, while the 386 SX CPU system segment descriptors have a 32-bit base address, a 20-bit limit field, and a granularity bit. The word count field specifies the number of 16-bit quantities to copy for 80286 call gates and 32-bit quantities for 386 SX CPU call gates.

Selector Fields

A selector in Protected Mode has three fields: Local or Global Descriptor Table indicator (TI), Descriptor Entry Index (Index), and Requestor (the selector's) Privilege Level (RPL) as shown in Figure 4.7. The TI bit selects either the Global Descriptor Table or the Local Descriptor Table. The Index selects one of 8k descriptors in the appropriate descriptor table. The RPL bits allow high speed testing of the selector's privilege attributes.

Segment Descriptor Cache

In addition to the selector value, every segment register has a segment descriptor cache register associated with it. Whenever a segment register's contents are changed, the 8-byte descriptor associated with that selector is automatically loaded (cached) on the chip. Once loaded, all references to that segment use the cached descriptor information instead of reaccessing the descriptor. The contents of the descriptor cache are not visible to the programmer. Since descriptor caches only change when a segment register is changed, programs which modify the descriptor tables must reload the appropriate segment registers after changing a descriptor's value.

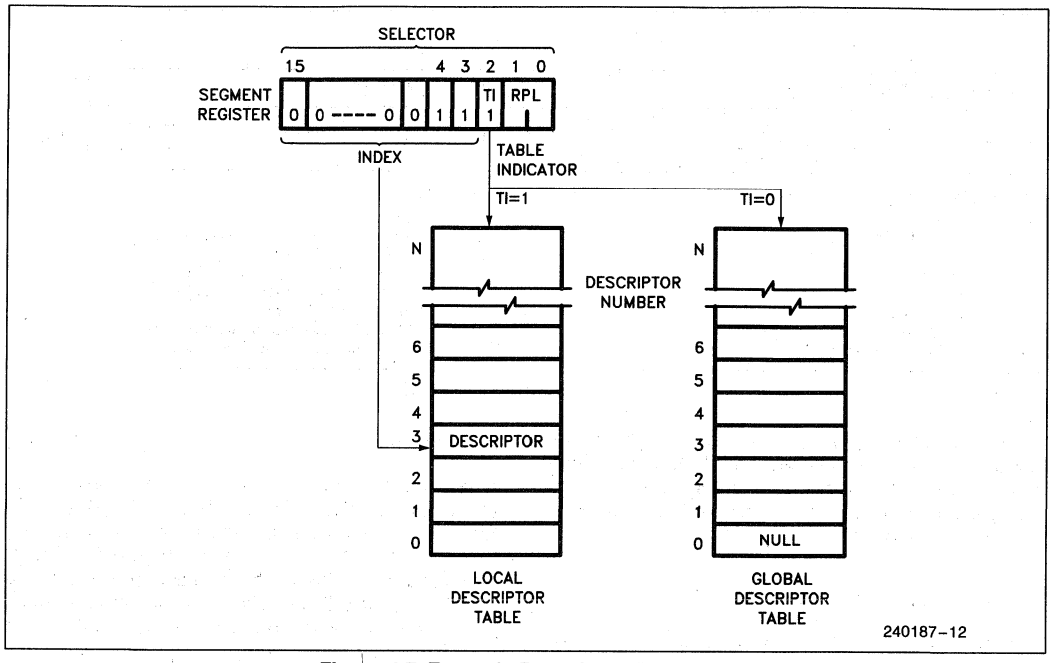


Figure 4.7. Example Descriptor Selection

4.3 Protection

The 386 SX Microprocessor has four levels of protection which are optimized to support a multi-tasking operating system and to isolate and protect user programs from each other and the operating system. The privilege levels control the use of privileged instructions, I/O instructions, and access to segments and segment descriptors. The 386 SX Microprocessor also offers an additional type of protection on a page basis when paging is enabled.

The four-level hierarchical privilege system is an extension of the user/supervisor privilege mode commonly used by minicomputers. The user/supervisor mode is fully supported by the 386 SX Microprocessor paging mechanism. The privilege levels (PL) are numbered 0 through 3. Level 0 is the most privileged level.

RULES OF PRIVILEGE

The 386 SX Microprocessor controls access to both data and procedures between levels of a task, according to the following rules.

— Data stored in a segment with privilege level **p** can be accessed only by code executing at a privilege level at least as privileged as **p**.

— A code segment/procedure with privilege level **p** can only be called by a task executing at the same or a lesser privilege level than **p**.

PRIVILEGE LEVELS

At any point in time, a task on the 386 SX Microprocessor always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies what the task's privilege level is. A task's CPL may only be changed by control transfers through gate descriptors to a code segment with a different privilege level. Thus, an application program running at PL=3 may call an operating system routine at PL=1 (via a gate) which would cause the task's CPL to be set to 1 until the operating system routine was finished.

Selector Privilege (RPL)

The privilege level of a selector is specified by the RPL field. The selector's RPL is only used to establish a less trusted privilege level than the current privilege level of the task for the use of a segment. This level is called the task's effective privilege level (EPL). The EPL is defined as being the least privileged (numerically larger) level of a task's CPL and a selector's RPL. The RPL is most commonly used to verify that pointers passed to an operating system procedure do not access data that is of higher privilege than the procedure that originated the pointer. Since the originator of a selector can specify any RPL value, the Adjust RPL (ARPL) instruction is provided to force the RPL bits to the originator's CPL.

Table 4.2. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL	CALL	Call Gate	GDT/LDT
	Interrupt instruction Exception External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
	CALL, JMP	Task State Segment	GDT
Task Switch	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag register) = 0
**NT (Nested Task bit of flag register) = 1

I/O Privilege

The I/O privilege level (IOPL) lets the operating system code executing at CPL = 0 define the least privileged level at which I/O instructions can be used. An exception 13 (General Protection Violation) is generated if an I/O instruction is attempted when the CPL of the task is less privileged then the IOPL. The IOPL is stored in bits 13 and 14 of the EFLAGS register. The following instructions cause an exception 13 if the CPL is greater than IOPL: IN, INS, OUT, OUTS, STI, CLI, LOCK prefix.

Descriptor Access

There are basically two types of segment accesses: those involving code segments such as control transfers, and those involving data accesses. Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL as described above.

Any time an instruction loads a data segment register (DS, ES, FS, GS) the 386 SX Microprocessor makes protection validation checks. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segment or readable code segments.

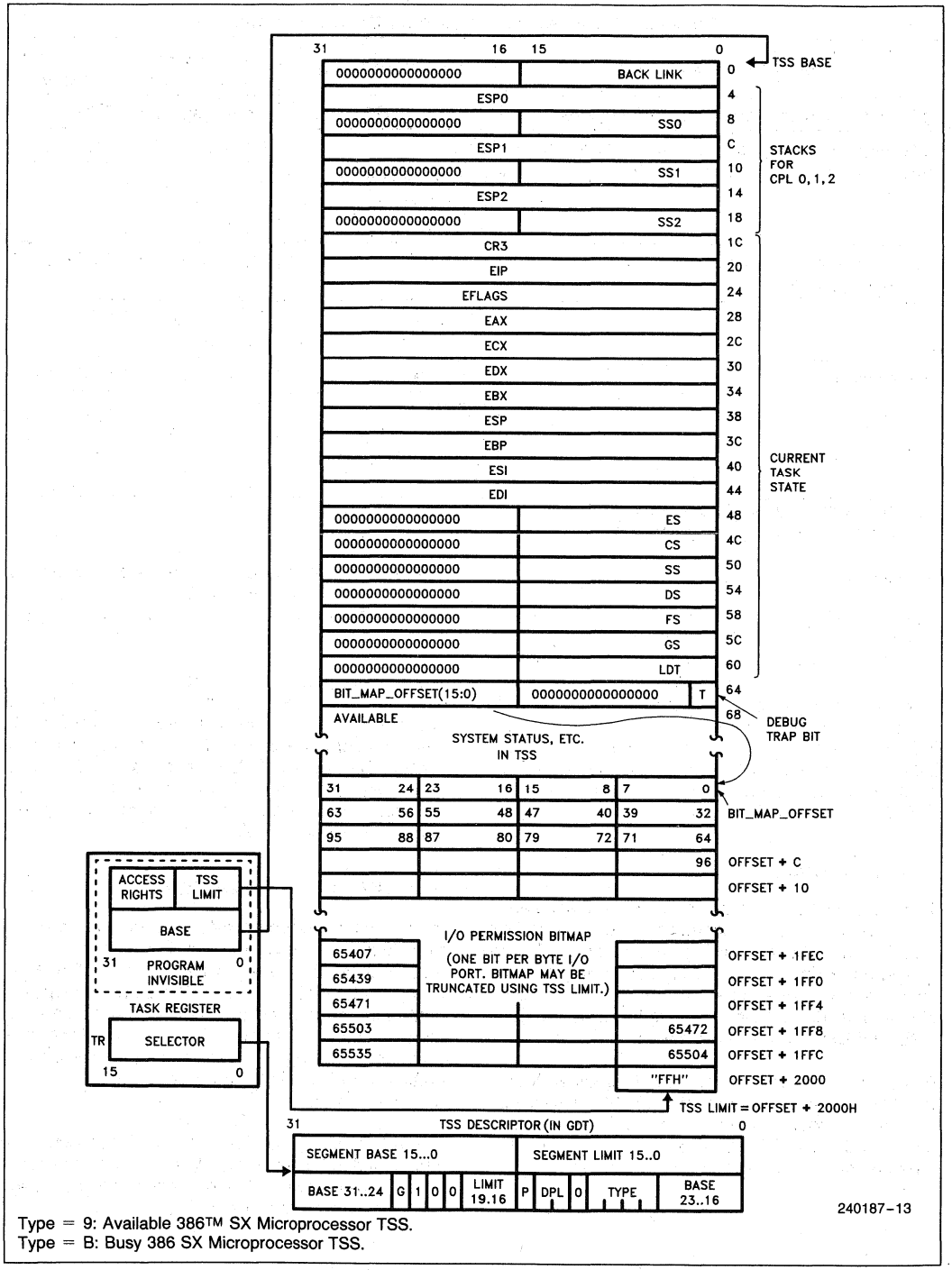
Finally the privilege validation checks are performed. The CPL is compared to the EPL and if the EPL is more privileged than the CPL, an exception 13 (general protection fault) is generated.

The rules regarding the stack segment are slightly different than those involving data segments. Instructions that load selectors into SS must refer to data segment descriptors for writeable data segments. The DPL and RPL must equal the CPL of all other descriptor types or a privilege level violation will cause an exception 13. A stack not present fault causes an exception 12.

PRIVILEGE LEVEL TRANSFERS

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call or a jump to another routine. There are five types of control transfers which are summarized in Table 4.2. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only by control transfers, using gates, task switches, and interrupt or trap gates.

Control transfers can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules will cause an exception 13.



Type = 9: Available 386™ SX Microprocessor TSS.
Type = B: Busy 386 SX Microprocessor TSS.

Figure 4.8. 386™ SX Microprocessor TSS and TSS Registers

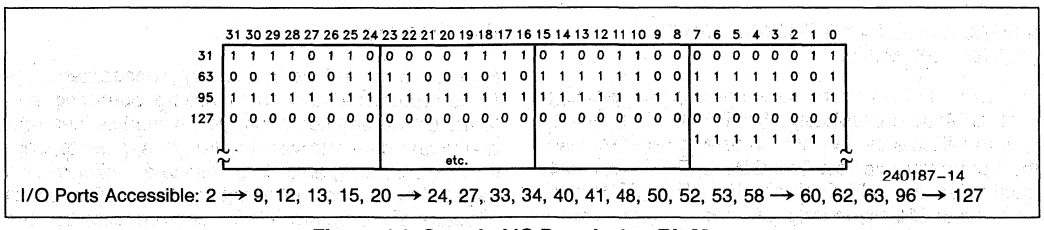


Figure 4.9. Sample I/O Permission Bit Map

CALL GATES

Gates provide protected indirect CALLs. One of the major uses of gates is to provide a secure method of privilege transfers within a task. Since the operating system defines all of the gates in a system, it can ensure that all gates only allow entry into a few trusted procedures.

TASK SWITCHING

A very important attribute of any multi-tasking/multi-user operating system is its ability to rapidly switch between tasks or processes. The 386 SX Microprocessor directly supports this operation by providing a task switch instruction in hardware. The task switch operation saves the entire state of the machine (all of the registers, address space, and a link to the previous task), loads a new execution state, performs protection checks, and commences execution in the new task. Like transfer of control by gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap, or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot.

The TSS descriptor points to a segment (see Figure 4.8) containing the entire execution state. A task gate descriptor contains a TSS selector. The 386 SX Microprocessor supports both 286 and 386 SX CPU TSSs. The limit of a 386 SX Microprocessor TSS must be greater than 64H (2BH for a 286 TSS), and can be as large as 16 megabytes. In the additional TSS space, the operating system is free to store additional information such as the reason the task is inactive, time the task has spent running, or open files belonging to the task.

Each task must have a TSS associated with it. The current TSS is identified by a special register in the 386 SX Microprocessor called the Task State Segment Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TSS descriptor are loaded whenever TR is loaded with a new selector. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to the task which was interrupted. The currently executing task's state is saved in the TSS and the old task state is restored from its TSS.

Several bits in the flag register and machine status word (CR0) give information about the state of a task which is useful to the operating system. The Nested Task bit, NT, controls the function of the IRET instruction. If NT=0 the IRET instruction performs the regular return. If NT=1 IRET performs a task switch operation back to the previous task. The NT bit is set or reset in the following fashion:

> When a CALL or INT instruction initiates a task switch, the new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT (The NT bit will be restored after execution of the interrupt handler). NT may also be set or cleared by POPF or IRET instructions.

The 386 SX Microprocessor task state segment is marked busy by changing the descriptor type field from TYPE 9 to TYPE 0BH. A 286 TSS is marked busy by changing the descriptor type field from TYPE 1 to TYPE 3. Use of a selector that references a busy task state segment causes an exception 13.

The VM (Virtual Mode) bit is used to indicate if a task is a Virtual 8086 task. If VM=1 then the tasks will use the Real Mode addressing mechanism. The virtual 8086 environment is only entered and exited by a task switch.

The coprocessor's state is not automatically saved when a task switch occurs. The Task Switched Bit, TS, in the CR0 register helps deal with the coprocessor's state in a multi-tasking environment. Whenever the 386 SX Microprocessor switches task, it sets the TS bit. The 386 SX Microprocessor detects the first use of a processor extension instruction after a task switch and causes the processor extension not available exception 7. The exception handler for exception 7 may then decide whether to save the state of the coprocessor.

The T bit in the 386 SX Microprocessor TSS indicates that the processor should generate a debug exception when switching to a task. If T=1 then upon entry to a new task a debug exception 1 will be generated.

INITIALIZATION AND TRANSITION TO PROTECTED MODE

Since the 386 SX Microprocessor begins executing in Real Mode immediately after RESET it is necessary to initialize the system tables and registers with the appropriate values. The GDT and IDT registers must refer to a valid GDT and IDT. The IDT should be at least 256 bytes long, and the GDT must contain descriptors for the initial code and data segments.

Protected Mode is enabled by loading CR0 with PE bit set. This can be accomplished by using the **MOV CR0, R/M** instruction. After enabling Protected Mode, the next instruction should execute an intersegment JMP to load the CS register and flush the instruction decode queue. The final step is to load all of the data segment registers with the initial selector values.

An alternate approach to entering Protected Mode is to use the built in task-switch to load all of the registers. In this case the GDT would contain two TSS descriptors in addition to the code and data descriptors needed for the first task. The first JMP instruction in Protected Mode would jump to the TSS causing a task switch and loading all of the registers with the values stored in the TSS. The Task State Segment Register should be initialized to point to a valid TSS descriptor.

4.4 Paging

Paging is another type of memory management useful for virtual memory multi-tasking operating systems. Unlike segmentation, which modularizes programs and data into variable length segments, paging divides programs into multiple uniform size pages. Pages bear no direct relation to the logical structure of a program. While segment selectors can be considered the logical 'name' of a program module or data structure, a page most likely corresponds to only a portion of a module or data structure.

PAGE ORGANIZATION

The 386 SX Microprocessor uses two levels of tables to translate the linear address (from the segmentation unit) into a physical address. There are three components to the paging mechanism of the 386 SX Microprocessor: the page directory, the page tables, and the page itself (page frame). All memory-resident elements of the 386 SX Microprocessor paging mechanism are the same size, namely 4K bytes. A uniform size for all of the elements simplifies memory allocation and reallocation schemes, since there is no problem with memory fragmentation. Figure 4.10 shows how the paging mechanism works.

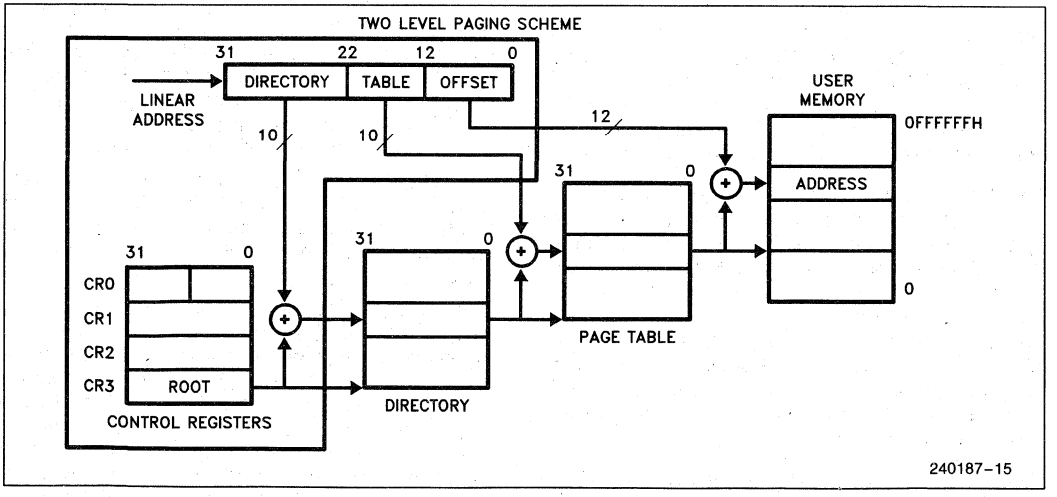


Figure 4.10. Paging Mechanism

31 12	11 10 9	8	7	6	5	4	3	2	1	0
PAGE TABLE ADDRESS 31..12	System Software Defineable	0	0	D	A	0	0	U / S	R / W	P

Figure 4.11. Page Directory Entry (Points to Page Table)

31 12	11 10	9	8	7	6	5	4	3	2	1	0
PAGE FRAME ADDRESS 31..12	System Software Defineable		0	0	D	A	0	0	U — S	R — W	P

Figure 4.12. Page Table Entry (Points to Page)

Page Fault Register

CR2 is the Page Fault Linear Address register. It holds the 32-bit linear address which caused the last Page Fault detected.

Page Descriptor Base Register

CR3 is the Page Directory Physical Base Address Register. It contains the physical starting address of the Page Directory (this value is truncated to a 24-bit value associated with the 386 SX CPU's 16 megabyte physical memory limitation). The lower 12 bits of CR3 are always zero to ensure that the Page Directory is always page aligned. Loading it with a **MOV CR3, reg** instruction causes the page table entry cache to be flushed, as will a task switch through a TSS which changes the value of CR0.

Page Directory

The Page Directory is 4k bytes long and allows up to 1024 page directory entries. Each page directory entry contains information about the page table and the address of the next level of tables, the Page Tables. The contents of a Page Directory Entry are shown in figure 4.11. The upper 10 bits of the linear address (A_{31}–A_{22}) are used as an index to select the correct Page Directory Entry.

The page table address contains the upper 20 bits of a 32-bit physical address that is used as the base address for the next set of tables, the page tables. The lower 12 bits of the page table address are zero so that the page table addresses appear on 4 kbyte boundaries. For a 386 DX CPU system the upper 20 bits will select one of 2^{20} page tables, but for a 386 SX Microprocessor system the upper 20 bits only select one of 2^{12} page tables. Again, this is because the 386 SX Microprocessor is limited to a 24-bit physical address and the upper 8 bits (A_{24}–A_{31}) are truncated when the address is output on its 24 address pins.

Page Tables

Each Page Table is 4K bytes long and allows up to 1024 Page table Entries. Each page table entry contains information about the Page Frame and its address. The contents of a Page Table Entry are shown in figure 4.12. The middle 10 bits of the linear address (A_{21}–A_{12}) are used as an index to select the correct Page Table Entry.

The Page Frame Address contains the upper 20 bits of a 32-bit physical address that is used as the base address for the Page Frame. The lower 12 bits of the Page Frame Address are zero so that the Page Frame addresses appear on 4 kbyte boundaries. For an 386 DX CPU system the upper 20 bits will select one of 2^{20} Page Frames, but for an 386 SX Microprocessor system the upper 20 bits only select one of 2^{12} Page Frames. Again, this is because the 386 SX Microprocessor is limited to a 24-bit physical address space and the upper 8 bits (A_{24}–A_{31}) are truncated when the address is output on its 24 address pins.

Page Directory/Table Entries

The lower 12 bits of the Page Table Entries and Page Directory Entries contain statistical information about pages and page tables respectively. The P (Present) bit indicates if a Page Directory or Page Table entry can be used in address translation. If P=1, the entry can be used for address translation. If P=0, the entry cannot be used for translation. All of the other bits are available for use by the software. For example, the remaining 31 bits could be used to indicate where on disk the page is stored.

The A (Accessed) bit is set by the 386 SX CPU for both types of entries before a read or write access occurs to an address covered by the entry. The D (Dirty) bit is set to 1 before a write to an address covered by that page table entry occurs. The D bit is undefined for Page Directory Entries. When the P, A and D bits are updated by the 386 SX CPU, the processor generates a Read- Modify-Write cycle which locks the bus and prevents conflicts with other processors or peripherals. Software which modifies these bits should use the LOCK prefix to ensure the integrity of the page tables in multi-master systems.

The 3 bits marked system software definable in Figures 4.11 and Figure 4.12 are software definable. System software writers are free to use these bits for whatever purpose they wish.

PAGE LEVEL PROTECTION (R/W, U/S BITS)

The 386 SX Microprocessor provides a set of protection attributes for paging systems. The paging mechanism distinguishes between two levels of protection: User, which corresponds to level 3 of the segmentation based protection, and supervisor which encompasses all of the other protection levels (0, 1, 2). Programs executing at Level 0, 1 or 2 bypass the page protection, although segmentation-based protection is still enforced by the hardware.

The U/S and R/W bits are used to provide User/Supervisor and Read/Write protection for individual pages or for all pages covered by a Page Table Directory Entry. The U/S and R/W bits in the second level Page Table Entry apply only to the page described by that entry. While the U/S and R/W bits in the first level Page Directory Table apply to all pages described by the page table pointed to by that directory entry. The U/S and R/W bits for a given page are obtained by taking the most restrictive of the U/S and R/W from the Page Directory Table Entries and using these bits to address the page.

TRANSLATION LOOKASIDE BUFFER

The 386 SX Microprocessor paging hardware is designed to support demand paged virtual memory systems. However, performance would degrade substantially if the processor was required to access two levels of tables for every memory reference. To solve this problem, the 386 SX Microprocessor keeps a cache of the most recently accessed pages, this cache is called the Translation Lookaside Buffer (TLB). The TLB is a four-way set associative 32-entry page table cache. It automatically keeps the most commonly used page table entries in the processor. The 32-entry TLB coupled with a 4K page size results in coverage of 128K bytes of memory addresses. For many common multi-tasking systems, the TLB will have a hit rate of greater than 98%. This means that the processor will only have to access the two-level page structure for less than 2% of all memory references.

PAGING OPERATION

The paging hardware operates in the following fashion. The paging unit hardware receives a 32-bit linear address from the segmentation unit. The upper 20 linear address bits are compared with all 32 entries in the TLB to determine if there is a match. If there is a match (i.e. a TLB hit), then the 24-bit physical address is calculated and is placed on the address bus.

If the page table entry is not in the TLB, the 386 SX Microprocessor will read the appropriate Page Directory Entry. If P=1 on the Page Directory Entry, indicating that the page table is in memory, then the 386 SX Microprocessor will read the appropriate Page Table Entry and set the Access bit. If P=1 on the Page Table Entry, indicating that the page is in memory, the 386 SX Microprocessor will update the Access and Dirty bits as needed and fetch the operand. The upper 20 bits of the linear address, read from the page table, will be stored in the TLB for future accesses. If P=0 for either the Page Directory Entry or the Page Table Entry, then the processor will generate a page fault Exception 14.

The processor will also generate a Page Fault (Exception 14) if the memory reference violated the page protection attributes. CR2 will hold the linear address which caused the page fault. Since Exception 14 is classified as a fault, CS:EIP will point to the instruction causing the page-fault. The 16-bit error code pushed as part of the page fault handler will contain status bits which indicate the cause of the page fault.

The 16-bit error code is used by the operating system to determine how to handle the Page Fault. Figure 4.13 shows the format of the Page Fault error code and the interpretation of the bits. Even though the bits in the error code (U/S, W/R, and P) have similar names as the bits in the Page Directory/Table Entries, the interpretation of the error code bits is different. Figure 4.14 indicates what type of access caused the page fault.

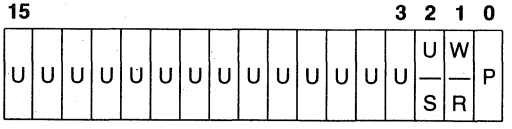


Figure 4.13. Page Fault Error Code Format

U/S: The U/S bit indicates whether the access causing the fault occurred when the processor was executing in User Mode (U/S = 1) or in Supervisor mode (U/S = 0)

W/R: The W/R bit indicates whether the access causing the fault was a Read (W/R = 0) or a Write (W/R = 1)

P: The P bit indicates whether a page fault was caused by a not-present page (P = 0), or by a page level protection violation (P = 1)

U = Undefined

U/S	W/R	Access Type
0	0	Supervisor* Read
0	1	Supervisor Write
1	0	User Read
1	1	User Write

*Descriptor table access will fault with U/S = 0, even if the program is executing at level 3.

Figure 4.14. Type of Access Causing Page Fault

OPERATING SYSTEM RESPONSIBILITIES

When the operating system enters or exits paging mode (by setting or resetting bit 31 in the CR0 register) a short JMP must be executed to flush the 386 SX Microprocessor's prefetch queue. This ensures that all instructions executed after the address mode change will generate correct addresses.

The 386 SX Microprocessor takes care of the page address translation process, relieving the burden from an operating system in a demand-paged system. The operating system is responsible for setting up the initial page tables and handling any page faults. The operating system also is required to invalidate (i.e. flush) the TLB when any changes are made to any of the page table entries. The operating system must reload CR3 to cause the TLB to be flushed.

Setting up the tables is simply a matter of loading CR3 with the address of the Page Directory, and allocating space for the Page Directory and the Page Tables. The primary responsibility of the operating system is to implement a swapping policy and handle all of the page faults.

A final concern of the operating system is to ensure that the TLB cache matches the information in the paging tables. In particular, any time the operating systems sets the P (Present) bit of page table entry to zero. The TLB must be flushed by reloading CR3. Operating systems may want to take advantage of the fact that CR3 is stored as part of a TSS, to give every task or group of tasks its own set of page tables.

4.5 Virtual 8086 Environment

The 386 SX Microprocessor allows the execution of 8086 application programs in both Real Mode and in the Virtual 8086 Mode. The Virtual 8086 Mode allows the execution of 8086 applications, while still allowing the system designer to take full advantage of the 386 SX CPU's protection mechanism.

VIRTUAL 8086 ADDRESSING MECHANISM

One of the major differences between 386 SX CPU Real and Protected modes is how the segment selectors are interpreted. When the processor is executing in Virtual 8086 Mode, the segment registers are used in a fashion identical to Real Mode. The contents of the segment register are shifted left 4 bits and added to the offset to form the segment base linear address.

The 386 SX Microprocessor allows the operating system to specify which programs use the 8086 address mechanism and which programs use Protected Mode addressing on a per task basis. Through the use of paging, the one megabyte address space of the Virtual Mode task can be mapped to anywhere in the 4 gigabyte linear address space of the 386 SX Microprocessor. Like Real Mode, Virtual Mode addresses that exceed one megabyte will cause an exception 13. However, these restrictions should not prove to be important, because most tasks running in Virtual 8086 Mode will simply be existing 8086 application programs.

PAGING IN VIRTUAL MODE

The paging hardware allows the concurrent running of multiple Virtual Mode tasks, and provides protection and operating system isolation. Although it is not strictly necessary to have the paging hardware enabled to run Virtual Mode tasks, it is needed in order to run multiple Virtual Mode tasks or to relocate the address space of a Virtual Mode task to physical address space greater than one megabyte.

The paging hardware allows the 20-bit linear address produced by a Virtual Mode program to be divided into as many as 256 pages. Each one of the pages can be located anywhere within the maximum 16 megabyte physical address space of the 386 SX Microprocessor. In addition, since CR3 (the Page Directory Base Register) is loaded by a task switch, each Virtual Mode task can use a different mapping scheme to map pages to different physical locations. Finally, the paging hardware allows the sharing of the 8086 operating system code between multiple 8086 applications.

PROTECTION AND I/O PERMISSION BIT MAP

All Virtual Mode programs execute at privilege level 3. As such, Virtual Mode programs are subject to all of the protection checks defined in Protected Mode. This is different than Real Mode, which implicitly is executing at privilege level 0. Thus, an attempt to execute a privileged instruction in Virtual Mode will cause an exception 13 fault.

The following are privileged instructions, which may be executed only at Privilege Level 0. Attempting to execute these instructions in Virtual 8086 Mode (or anytime CPL≥0) causes an exception 13 fault:

```
LIDT;   MOV DRn,REG;   MOV reg,DRn;
LGDT;   MOV TRn,reg;   MOV reg,TRn;
LMSW;   MOV CRn,reg;   MOV reg,CRn;

CLTS;
HLT;
```

Several instructions, particularly those applying to the multitasking and the protection model, are available only in Protected Mode. Therefore, attempting to execute the following instructions in Real Mode or in Virtual 8086 Mode generates an exception 6 fault:

```
LTR;    STR;
LLDT;   SLDT;
LAR;    VERR;
LSL;    VERW;
ARPL;
```

The instructions which are IOPL sensitive in Protected Mode are:

```
IN;     STI;
OUT;    CLI
INS;
OUTS;
REP INS;
REP OUTS;
```

In Virtual 8086 Mode the following instructions are IOPL-sensitive:

```
INT n;  STI;
PUSHF;  CLI;
POPF;   IRET;
```

The PUSHF, POPF, and IRET instructions are IOPL-sensitive in Virtual 8086 Mode only. This provision allows the IF flag to be virtualized to the virtual 8086 Mode program. The INT n software interrupt instruction is also IOPL-sensitive in Virtual 8086 mode. Note that the INT 3, INTO, and BOUND instructions are not IOPL-sensitive in Virtual 8086 Mode.

The I/O instructions that directly refer to addresses in the processor's I/O space are IN, INS, OUT, and OUTS. The 386 SX Microprocessor has the ability to selectively trap references to specific I/O addresses. The structure that enables selective trapping is the *I/O Permission Bit Map* in the TSS segment (see Figures 4.8 and 4.9). The I/O permission map is a bit vector. The size of the map and its location in the TSS segment are variable. The processor locates the I/O permission map by means of the **I/O map base** field in the fixed portion of the TSS. The **I/O map base** field is 16 bits wide and contains the offset of the beginning of the I/O permission map.

In protected mode when an I/O instruction (IN, INS, OUT or OUTS) is encountered, the processor first checks whether CPL ≤ IOPL. If this condition is true, the I/O operation may proceed. If not true, the processor checks the I/O permission map (in Virtual 8086 Mode, the processor consults the map without regard for the IOPL).

Each bit in the map corresponds to an I/O port byte address; for example, the bit for port 41 is found at **I/O map base** + 5, bit offset 1. The processor tests all the bits that correspond to the I/O addresses spanned by an I/O operation; for example, a double word operation tests four bits corresponding to four adjacent byte addresses. If any tested bit is set, the processor signals a general protection exception. If all the tested bits are zero, the I/O operations may proceed.

It is not necessary for the I/O permission map to represent all the I/O addresses. I/O addresses not spanned by the map are treated as if they had one-bits in the map. The **I/O map base** should be at least one byte less than the TSS limit, the last byte beyond the I/O mapping information must contain all 1's.

Because the I/O permission map is in the TSS segment, different tasks can have different maps. Thus, the operating system can allocate ports to a task by changing the I/O permission map in the task's TSS.

IMPORTANT IMPLEMENTATION NOTE: Beyond the last byte of I/O mapping information in the I/O permission bit map **must** be a byte containing all 1's. The byte of all 1's must be within the limit of the 386 SX CPU TSS segment (see Figure 4.8).

Interrupt Handling

In order to fully support the emulation of an 8086 machine, interrupts in Virtual 8086 Mode are handled in a unique fashion. When running in Virtual Mode all interrupts and exceptions involve a privilege change back to the host 386 SX Microprocessor operating system. The 386 SX Microprocessor operating system determines if the interrupt comes from a Protected Mode application or from a Virtual Mode program by examining the VM bit in the EFLAGS image stored on the stack.

When a Virtual Mode program is interrupted and execution passes to the interrupt routine at level 0, the VM bit is cleared. However, the VM bit is still set in the EFLAG image on the stack.

The 386 SX Microprocessor operating system in turn handles the exception or interrupt and then returns control to the 8086 program. The 386 SX Microprocessor operating system may choose to let the 8086 operating system handle the interrupt or it may emulate the function of the interrupt handler. For example, many 8086 operating system calls are accessed by PUSHing parameters on the stack, and then executing an INT n instruction. If the IOPL is set to 0 then all INT n instructions will be intercepted by the 386 SX Microprocessor operating system.

An 386 SX Microprocessor operating system can provide a Virtual 8086 Environment which is totally transparent to the application software by intercepting and then emulating 8086 operating system's calls, and intercepting IN and OUT instructions.

Entering and Leaving Virtual 8086 Mode

Virtual 8086 mode is entered by executing a 32-bit IRET instruction at CPL = 0 where the stack has a 1 in the VM bit of its EFLAGS image, or a Task Switch (at any CPL) to a 386 SX Microprocessor task whose 386 SX CPU TSS has a EFLAGS image containing a 1 in the VM bit position while the processor is executing in the Protected Mode. POPF does not affect the VM bit but a PUSHF always pushes a 0 in the VM bit.

The transition out of Virtual 8086 mode to protected mode occurs only on receipt of an interrupt or exception. In Virtual 8086 mode, all interrupts and exceptions vector through the protected mode IDT, and enter an interrupt handler in protected mode. As part of the interrupt processing the VM bit is cleared.

Because the matching IRET must occur from level 0, Interrupt or Trap Gates used to field an interrupt or exception out of Virtual 8086 mode must perform an inter-level interrupt only to level 0. Interrupt or Trap Gates through conforming segments, or through segments with DPL > 0, will raise a GP fault with the CS selector as the error code.

Task Switches To/From Virtual 8086 Mode

Tasks which can execute in Virtual 8086 mode must be described by a TSS with the 386 SX CPU format (type 9 or 11 descriptor). A task switch out of virtual 8086 mode will operate exactly the same as any other task switch out of a task with a 386 SX CPU TSS. All of the programmer visible state, including the EFLAGS register with the VM bit set to 1, is stored in the TSS. The segment registers in the TSS will contain 8086 segment base values rather than selectors.

A task switch into a task described by a 386 SX CPU TSS will have an additional check to determine if the incoming task should be resumed in Virtual 8086 mode. Tasks described by 286 format TSSs cannot be resumed in Virtual 8086 mode, so no check is required there (the FLAGS image in 286 format TSS has only the low order 16 FLAGS bits). Before loading the segment register images from a 386 SX CPU TSS, the FLAGS image is loaded, so that the segment registers are loaded from the TSS image as 8086 segment base values. The task is now ready to resume in Virtual 8086 mode.

Transitions Through Trap and Interrupt Gates, and IRET

A task switch is one way to enter or exit Virtual 8086 mode. The other method is to exit through a Trap or Interrupt gate, as part of handling an interrupt, and to enter as part of executing an IRET instruction. The transition out must use a 386 SX CPU Trap Gate (Type 14), or 386 SX CPU Interrupt Gate (Type 15), which must point to a non-conforming level 0 segment (DPL = 0) in order to permit the trap handler to IRET back to the Virtual 8086 program. The Gate must point to a non-conforming level 0 segment to perform a level switch to level 0 so that the matching IRET can change the VM bit. 386 SX CPU gates must be used since 286 gates save only the low 16 bits of the EFLAGS register (the VM bit will not be saved). Also, the 16-bit IRET used to terminate the 286 interrupt handler will pop only the lower 16 bits from FLAGS, and will not affect the VM bit. The action taken for a 386 SX CPU Trap or Interrupt gate if an interrupt occurs while the task is executing in virtual 8086 mode is given by the following sequence:

1. Save the FLAGS register in a temp to push later. Turn off the VM, TF, and IF bits.
2. Interrupt and Trap gates must perform a level switch from 3 (where the Virtual 8086 Mode program executes) to level 0 (so IRET can return).
3. Push the 8086 segment register values onto the new stack, in this order: GS, FS, DS, ES. These are pushed as 32-bit quantities. Then load these 4 registers with null selectors (0).
4. Push the old 8086 stack pointer onto the new stack by pushing the SS register (as 32-bits), then pushing the 32-bit ESP register saved above.
5. Push the 32-bit EFLAGS register saved in step 1.
6. Push the old 8086 instruction onto the new stack by pushing the CS register (as 32-bits), then pushing the 32-bit EIP register.
7. Load up the new CS:EIP value from the interrupt gate, and begin execution of the interrupt routine in protected mode.

The transition out of V86 mode performs a level change and stack switch, in addition to changing back to protected mode. Also all of the 8086 segment register images are stored on the stack (behind the SS:ESP image), and then loaded with null (0) selectors before entering the interrupt handler. This will permit the handler to safely save and restore the DS, ES, FS, and GS registers as 286 selectors. This is needed so that interrupt handlers which don't care about the mode of the interrupted program can use the same prologue and epilogue code for state saving regardless of whether or not a 'native' mode or Virtual 8086 Mode program was interrupted. Restoring null selectors to these registers

before executing the IRET will cause a trap in the interrupt handler. Interrupt routines which expect or return values in the segment registers will have to obtain/return values from the 8086 register images pushed onto the new stack. They will need to know the mode of the interrupted program in order to know where to find/return segment registers, and also to know how to interpret segment register values.

The IRET instruction will perform the inverse of the above sequence. Only the extended IRET instruction (operand size=32) can be used and must be executed at level 0 to change the VM bit to 1.

1. If the NT bit in the FLAGS register is on, an inter-task return is performed. The current state is stored in the current TSS, and the link field in the current TSS is used to locate the TSS for the interrupted task which is to be resumed. Otherwise, continue with the following sequence:
2. Read the FLAGS image from SS:8[ESP] into the FLAGS register. This will set VM to the value active in the interrupted routine.
3. Pop off the instruction pointer CS:EIP. EIP is popped first, then a 32-bit word is popped which contains the CS value in the lower 16 bits. If VM=0, this CS load is done as a protected mode segment load. If VM=1, this will be done as an 8086 segment load.
4. Increment the ESP register by 4 to bypass the FLAGS image which was 'popped' in step 1.
5. If VM=1, load segment registers ES, DS, FS, and GS from memory locations SS:[ESP+8], SS:[ESP+12], SS:[ESP+16], and SS:[ESP=20], respectively, where the new value of ESP stored in step 4 is used. Since VM=1, these are done as 8086 segment register loads.

   Else if VM=0, check that the selectors in ES, DS, FS, and GS are valid in the interrupted routine. Null out invalid selectors to trap if an attempt is made to access through them.
6. If RPL(CS)>CPL, pop the stack pointer SS:ESP from the stack. The ESP register is popped first, followed by 32-bits containing SS in the lower 16 bits. If VM=0, SS is loaded as a protected mode segment register load. If VM=1, an 8086 segment register load is used.
7. Resume execution of the interrupted routine. The VM bit in the FLAGS register (restored from the interrupt routine's stack image in step 1) determines whether the processor resumes the interrupted routine in Protected mode or Virtual 8086 Mode.

5.0 FUNCTIONAL DATA

The 386 SX Microprocessor features a straightforward functional interface to the external hardware. The 386 SX Microprocessor has separate parallel buses for data and address. The data bus is 16-bits in width, and bi-directional. The address bus outputs 24-bit address values using 23 address lines and two byte enable signals.

The 386 SX Microprocessor has two selectable address bus cycles: address pipelined and non-address pipelined. The address pipelining option allows as much time as possible for data access by starting the pending bus cycle before the present bus cycle is finished. A non-pipelined bus cycle gives the highest bus performance by executing every bus cycle in two processor CLK cycles. For maximum design flexibility, the address pipelining option is selectable on a cycle-by-cycle basis.

The processor's bus cycle is the basic mechanism for information transfer, either from system to processor, or from processor to system. 386 SX Microprocessor bus cycles perform data transfer in a minimum of only two clock periods. The maximum transfer bandwidth at 16 MHz is therefore 16 Mbytes/sec. However, any bus cycle will be extended for more than two clock periods if external hardware withholds acknowledgement of the cycle.

The 386 SX Microprocessor can relinquish control of its local buses to allow mastership by other devices, such as direct memory access (DMA) channels. When relinquished, HLDA is the only output pin driven by the 386 SX Microprocessor, providing near-complete isolation of the processor from its system (all other output pins are in a float condition).

5.1 Signal Description Overview

Ahead is a brief description of the 386 SX Microprocessor input and output signals arranged by functional groups. Note the # symbol at the end of a signal name indicates the active, or asserted, state occurs when the signal is at a LOW voltage. When no # is present after the signal name, the signal is asserted when at the HIGH voltage level.

Example signal: M/IO# — HIGH voltage indicates Memory selected

— LOW voltage indicates I/O selected

The signal descriptions sometimes refer to AC timing parameters, such as 't_{25} Reset Setup Time' and 't_{26} Reset Hold Time.' The values of these parameters can be found in Table 7.4.

CLOCK (CLK2)

CLK2 provides the fundamental timing for the 386 SX Microprocessor. It is divided by two internally to generate the internal processor clock used for instruction execution. The internal clock is comprised of two phases, 'phase one' and 'phase two'. Each CLK2 period is a phase of the internal clock. Figure 5.2 illustrates the relationship. If desired, the phase of the internal processor clock can be synchronized to a known phase by ensuring the falling edge of the RESET signal meets the applicable setup and hold times t_{25} and t_{26}.

DATA BUS (D_{15}–D_0)

These three-state bidirectional signals provide the general purpose data path between the 386 SX Microprocessor and other devices. The data bus outputs are active HIGH and will float during bus hold acknowledge. Data bus reads require that read-data setup and hold times t_{21} and t_{22} be met relative to CLK2 for correct operation.

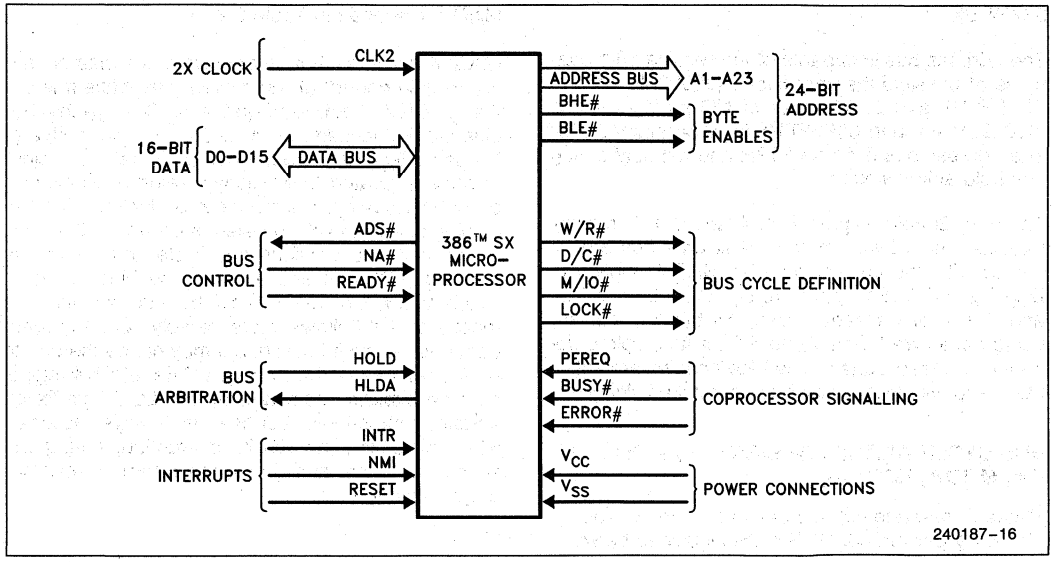


Figure 5.1. Functional Signal Groups

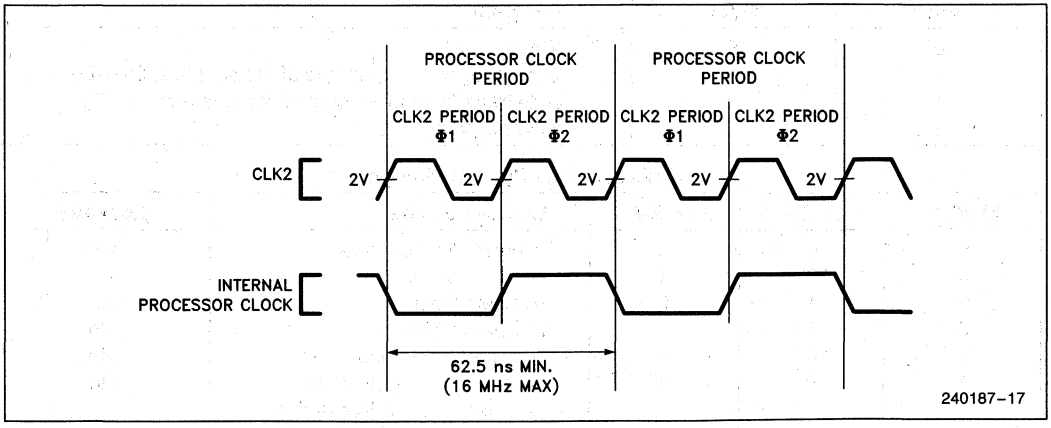


Figure 5.2. CLK2 Signal and Internal Processor Clock

ADDRESS BUS (A_{23}–A_1, BHE#, BLE#)

These three-state outputs provide physical memory addresses or I/O port addresses. A_{23}–A_{16} are LOW during I/O transfers except for I/O transfers automatically generated by coprocessor instructions. During coprocessor I/O transfers, A_{22}–A_{16} are driven LOW, and A_{23} is driven HIGH so that this address line can be used by external logic to generate the coprocessor select signal. Thus, the I/O address driven by the 386 SX Microprocessor for coprocessor commands is 8000F8H, the I/O addresses driven by the 386 SX Microprocessor for coprocessor data are 8000FCH or 8000FEH for cycles to the 387™ SX.

The address bus is capable of addressing 16 megabytes of physical memory space (000000H through FFFFFFH), and 64 kilobytes of I/O address space (000000H through 00FFFFH) for programmed I/O. The address bus is active HIGH and will float during bus hold acknowledge.

The Byte Enable outputs, BHE# and BLE#, directly indicate which bytes of the 16-bit data bus are involved with the current transfer. BHE# applies to D_{15}–D_8 and BLE# applies to D_7–D_0. If both BHE# and BLE# are asserted, then 16 bits of data are being transferred. See Table 5.1 for a complete decoding of these signals. The byte enables are active LOW and will float during bus hold acknowledge.

BUS CYCLE DEFINITION SIGNALS (W/R#, D/C#, M/IO#, LOCK#)

These three-state outputs define the type of bus cycle being performed: W/R# distinguishes between write and read cycles, D/C# distinguishes between data and control cycles, M/IO# distinguishes between memory and I/O cycles, and LOCK# distinguishes between locked and unlocked bus cycles. All of these signals are active LOW and will float during bus acknowledge.

The primary bus cycle definition signals are W/R#, D/C# and M/IO#, since these are the signals driven valid as ADS# (Address Status output) becomes active. The LOCK# is driven valid at the same time the bus cycle begins, which due to address pipelining, could be after ADS# becomes active. Exact bus cycle definitions, as a function of W/R#, D/C#, and M/IO# are given in Table 5.2.

LOCK# indicates that other system bus masters are not to gain control of the system bus while it is active. LOCK# is activated on the CLK2 edge that begins the first locked bus cycle (i.e., it is not active at the same time as the other bus cycle definition pins) and is deactivated when ready is returned at the end of the last bus cycle which is to be locked. The beginning of a bus cycle is determined when READY# is returned in a previous bus cycle and another is pending (ADS# is active) or by the clock edge in which ADS# is driven active if the bus was idle. This means that it follows more closely with the write data rules when it is valid, but may cause the bus to be locked longer than desired. The LOCK# signal may be explicitly activated by the LOCK prefix on certain instructions. LOCK# is always asserted when executing the XCHG instruction, during descriptor updates, and during the interrupt acknowledge sequence.

Table 5.1. Byte Enable Definitions

BHE#	BLE#	Function
0	0	Word Transfer
0	1	Byte transfer on upper byte of the data bus, D_{15}–D_8
1	0	Byte transfer on lower byte of the data bus, D_7–D_0
1	1	Never occurs

Table 5.2. Bus Cycle Definition

M/IO#	D/C#	W/R#	Bus Cycle Type	Locked?
0	0	0	Interrupt Acknowledge	Yes
0	0	1	does not occur	—
0	1	0	I/O Data Read	No
0	1	1	I/O Data Write	No
1	0	0	Memory Code Read	No
1	0	1	Halt: Address = 2, BHE# = 1, BLE# = 0 / Shutdown: Address = 0, BHE# = 1, BLE# = 0	No
1	1	0	Memory Data Read	Some Cycles
1	1	1	Memory Data Write	Some Cycles

BUS CONTROL SIGNALS (ADS#, READY#, NA#)

The following signals allow the processor to indicate when a bus cycle has begun, and allow other system hardware to control address pipelining and bus cycle termination.

Address Status (ADS#)

This three-state output indicates that a valid bus cycle definition and address (W/R#, D/C#, M/IO#, BHE#, BLE# and A_{23}–A_1) are being driven at the 386 SX Microprocessor pins. ADS# is an active LOW output. Once ADS# is driven active, valid address, byte enables, and definition signals will not change. In addition, ADS# will remain active until its associated bus cycle begins (when READY# is returned for the previous bus cycle when running pipelined bus cycles). When address pipelining is utilized, maximum throughput is achieved by initiating bus cycles when ADS# and READY# are active in the same clock cycle. ADS# will float during bus hold acknowledge. See sections **Non-Pipelined Address** and **Pipelined Address** for additional information on how ADS# is asserted for different bus states.

Transfer Acknowledge (READY#)

This input indicates the current bus cycle is complete, and the active bytes indicated by BHE# and BLE# are accepted or provided. When READY# is sampled active during a read cycle or interrupt acknowledge cycle, the 386 SX Microprocessor latches the input data and terminates the cycle. When READY# is sampled active during a write cycle, the processor terminates the bus cycle.

READY# is ignored on the first bus state of all bus cycles, and sampled each bus state thereafter until asserted. READY# must eventually be asserted to acknowledge every bus cycle, including Halt Indication and Shutdown Indication bus cycles. When being sampled, READY# must always meet setup and hold times t_{19} and t_{20} for correct operation.

Next Address Request (NA#)

This is used to request address pipelining. This input indicates the system is prepared to accept new values of BHE#, BLE#, A_{23}–A_1, W/R#, D/C# and M/IO# from the 386 SX Microprocessor even if the end of the current cycle is not being acknowledged on READY#. If this input is active when sampled, the next address is driven onto the bus, provided the next bus request is already pending internally. NA# is ignored in CLK cycles in which ADS# or READY# is activated. This signal is active LOW and must satisfy setup and hold times t_{15} and t_{16} for correct operation. See **Pipelined Address** and **Read and Write Cycles** for additional information.

BUS ARBITRATION SIGNALS (HOLD, HLDA)

This section describes the mechanism by which the processor relinquishes control of its local buses when requested by another bus master device. See **Entering and Exiting Hold Acknowledge** for additional information.

Bus Hold Request (HOLD)

This input indicates some device other than the 386 SX Microprocessor requires bus mastership. When control is granted, the 386 SX Microprocessor floats A_{23}–A_1, BHE#, BLE#, D_{15}–D_0, LOCK#, M/IO#, D/C#, W/R# and ADS#, and then activates HLDA, thus entering the bus hold acknowledge state. The local bus will remain granted to the requesting master until HOLD becomes inactive. When HOLD becomes inactive, the 386 SX Microprocessor will deactivate HLDA and drive the local bus (at the same time), thus terminating the hold acknowledge condition.

HOLD must remain asserted as long as any other device is a local bus master. External pull-up resistors may be required when in the hold acknowledge state since none of the 386 SX Microprocessor floated outputs have internal pull-up resistors. See **Resistor Recommendations** for additional information. HOLD is not recognized while RESET is active. If RESET is asserted while HOLD is asserted, RESET has priority and places the bus into an idle state, rather than the hold acknowledge (high-impedance) state.

HOLD is a level-sensitive, active HIGH, synchronous input. HOLD signals must always meet setup and hold times t_{23} and t_{24} for correct operation.

Bus Hold Acknowledge (HLDA)

When active (HIGH), this output indicates the 386 SX Microprocessor has relinquished control of its local bus in response to an asserted HOLD signal, and is in the bus Hold Acknowledge state.

The Bus Hold Acknowledge state offers near-complete signal isolation. In the Hold Acknowledge state, HLDA is the only signal being driven by the 386 SX Microprocessor. The other output signals or bidirectional signals (D_{15}–D_0, BHE#, BLE#, A_{23}–A_1, W/R#, D/C#, M/IO#, LOCK# and ADS#) are in a high-impedance state so the requesting bus

master may control them. These pins remain OFF throughout the time that HLDA remains active (see Table 5.3)). Pull-up resistors may be desired on several signals to avoid spurious activity when no bus master is driving them. See **Resistor Recommendations** for additional information.

When the HOLD signal is made inactive, the 386 SX Microprocessor will deactivate HLDA and drive the bus. One rising edge on the NMI input is remembered for processing after the HOLD input is negated.

Table 5.3. Output pin State During HOLD

Pin Value	Pin Names
1	HLDA
Float	LOCK#, M/IO#, D/C#, W/R#, ADS#, A_{23}–A_1, BHE#, BLE#, D_{15}–D_0

In addition to the normal usage of Hold Acknowledge with DMA controllers or master peripherals, the near-complete isolation has particular attractiveness during system test when test equipment drives the system, and in hardware fault-tolerant applications.

HOLD Latencies

The maximum possible HOLD latency depends on the software being executed. The actual HOLD latency at any time depends on the current bus activity, the state of the LOCK# signal (internal to the CPU) activated by the LOCK# prefix, and interrupts. The 386 SX Microprocessor will not honor a HOLD request until the current bus operation is complete.

The 386 SX Microprocessor breaks 32-bit data or I/O accesses into 2 internally locked 16-bit bus cycles; the LOCK# signal is not asserted. The 386 SX Microprocessor breaks unaligned 16-bit or 32-bit data or I/O accesses into 2 or 3 internally locked 16-bit bus cycles. Again, the LOCK# signal is not asserted but a HOLD request will not be recognized until the end of the entire transfer.

Wait states affect HOLD latency. The 386 SX Microprocessor will not honor a HOLD request until the end of the current bus operation, no matter how many wait states are required. Systems with DMA where data transfer is critical must insure that READY# returns sufficiently soon.

COPROCESSOR INTERFACE SIGNALS (PEREQ, BUSY#, ERROR#)

In the following sections are descriptions of signals dedicated to the numeric coprocessor interface. In addition to the data bus, address bus, and bus cycle definition signals, these following signals control communication between the 386 SX Microprocessor and its 387™ SX processor extension.

Coprocessor Request (PEREQ)

When asserted (HIGH), this input signal indicates a coprocessor request for a data operand to be transferred to/from memory by the 386 SX Microprocessor. In response, the 386 SX Microprocessor transfers information between the coprocessor and memory. Because the 386 SX Microprocessor has internally stored the coprocessor opcode being executed, it performs the requested data transfer with the correct direction and memory address.

PEREQ is a level-sensitive active HIGH asynchronous signal. Setup and hold times, t_{29} and t_{30}, relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. This signal is provided with a weak internal pull-down resistor of around 20 K-ohms to ground so that it will not float active when left unconnected.

Coprocessor Busy (BUSY#)

When asserted (LOW), this input indicates the coprocessor is still executing an instruction, and is not yet able to accept another. When the 386 SX Microprocessor encounters any coprocessor instruction which operates on the numerics stack (e.g. load, pop, or arithmetic operation), or the WAIT instruction, this input is first automatically sampled until it is seen to be inactive. This sampling of the BUSY# input prevents overrunning the execution of a previous coprocessor instruction.

The FNINIT, FNSTENV, FNSAVE, FNSTSW, FNSTCW and FNCLEX coprocessor instructions are allowed to execute even if BUSY# is active, since these instructions are used for coprocessor initialization and exception-clearing.

BUSY# is an active LOW, level-sensitive asynchronous signal. Setup and hold times, t_{29} and t_{30}, rela-

tive to the CLK2 signal must be met to guarantee recognition at a particular clock edge. This pin is provided with a weak internal pull-up resistor of around 20 K-ohms to Vcc so that it will not float active when left unconnected.

BUSY# serves an additional function. If BUSY# is sampled LOW at the falling edge of RESET, the 386 SX Microprocessor performs an internal self-test (see **Bus Activity During and Following Reset**. If BUSY# is sampled HIGH, no self-test is performed.

Coprocessor Error (ERROR#)

When asserted (LOW), this input signal indicates that the previous coprocessor instruction generated a coprocessor error of a type not masked by the coprocessor's control register. This input is automatically sampled by the 386 SX Microprocessor when a coprocessor instruction is encountered, and if active, the 386 SX Microprocessor generates exception 16 to access the error-handling software.

Several coprocessor instructions, generally those which clear the numeric error flags in the coprocessor or save coprocessor state, do execute without the 386 SX Microprocessor generating exception 16 even if ERROR# is active. These instructions are FNINIT, FNCLEX, FNSTSW, FNSTSWAX, FNSTCW, FNSTENV and FNSAVE.

ERROR# is an active LOW, level-sensitive asynchronous signal. Setup and hold times, t_{29} and t_{30}, relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. This pin is provided with a weak internal pull-up resistor of around 20 K-ohms to Vcc so that it will not float active when left unconnected.

INTERRUPT SIGNALS (INTR, NMI, RESET)

The following descriptions cover inputs that can interrupt or suspend execution of the processor's current instruction stream.

Maskable Interrupt Request (INTR)

When asserted, this input indicates a request for interrupt service, which can be masked by the 386 SX CPU Flag Register IF bit. When the 386 SX Microprocessor responds to the INTR input, it performs two interrupt acknowledge bus cycles and, at the end of the second, latches an 8-bit interrupt vector on D_7–D_0 to identify the source of the interrupt.

INTR is an active HIGH, level-sensitive asynchronous signal. Setup and hold times, t_{27} and t_{28}, relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. To assure recognition of an INTR request, INTR should remain active until the first interrupt acknowledge bus cycle begins. INTR is sampled at the beginning of every instruction in the 386 SX Microprocessor's Execution Unit. In order to be recognized at a particular instruction boundary, INTR must be active at least eight CLK2 clock periods before the beginning of the instruction. If recognized, the 386 SX Microprocessor will begin execution of the interrupt.

Non-Maskable Interrupt Request (NMI))

This input indicates a request for interrupt service which cannot be masked by software. The non-maskable interrupt request is always processed according to the pointer or gate in slot 2 of the interrupt table. Because of the fixed NMI slot assignment, no interrupt acknowledge cycles are performed when processing NMI.

NMI is an active HIGH, rising edge-sensitive asynchronous signal. Setup and hold times, t_{27} and t_{28}, relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. To assure recognition of NMI, it must be inactive for at least eight CLK2 periods, and then be active for at least eight CLK2 periods before the beginning of the instruction boundary in the 386 SX Microprocessor's Execution Unit.

Once NMI processing has begun, no additional NMI's are processed until after the next IRET instruction, which is typically the end of the NMI service routine. If NMI is re-asserted prior to that time, however, one rising edge on NMI will be remembered for processing after executing the next IRET instruction.

Interrupt Latency

The time that elapses before an interrupt request is serviced (interrupt latency) varies according to several factors. This delay must be taken into account by the interrupt source. Any of the following factors can affect interrupt latency:

1. If interrupts are masked, an INTR request will not be recognized until interrupts are reenabled.
2. If an NMI is currently being serviced, an incoming NMI request will not be recognized until the 386 SX Microprocessor encounters the IRET instruction.
3. An interrupt request is recognized only on an instruction boundary of the 386 SX Microprocessor's Execution Unit except for the following cases:
   — Repeat string instructions can be interrupted after each iteration.

— If the instruction loads the Stack Segment register, an interrupt is not processed until after the following instruction, which should be an ESP. This allows the entire stack pointer to be loaded without interruption.

— If an instruction sets the interrupt flag (enabling interrupts), an interrupt is not processed until after the next instruction.

The longest latency occurs when the interrupt request arrives while the 386 SX Microprocessor is executing a long instruction such as multiplication, division, or a task-switch in the protected mode.

4. Saving the Flags register and CS:EIP registers.
5. If interrupt service routine requires a task switch, time must be allowed for the task switch.
6. If the interrupt service routine saves registers that are not automatically saved by the 386 SX Microprocessor.

RESET

This input signal suspends any operation in progress and places the 386 SX Microprocessor in a known reset state. The 386 SX Microprocessor is reset by asserting RESET for 15 or more CLK2 periods (80 or more CLK2 periods before requesting self-test). When RESET is active, all other input pins, except FLT#, are ignored, and all other bus pins are driven to an idle bus state as shown in Table 5.5. If RESET and HOLD are both active at a point in time, RESET takes priority even if the 386 SX Microprocessor was in a Hold Acknowledge state prior to RESET active.

RESET is an active HIGH, level-sensitive synchronous signal. Setup and hold times, t_{25} and t_{26}, must be met in order to assure proper operation of the 386 SX Microprocessor.

Table 5.5. Pin State (Bus Idle) During Reset

Pin Name	Signal Level During Reset
ADS#	1
D_{15}–D_0	Float
BHE#, BLE#	0
A_{23}–A_1	1
W/R#	0
D/C#	1
M/IO#	0
LOCK#	1
HLDA	0

5.2 Bus Transfer Mechanism

All data transfers occur as a result of one or more bus cycles. Logical data operands of byte and word lengths may be transferred without restrictions on physical address alignment. Any byte boundary may be used, although two physical bus cycles are performed as required for unaligned operand transfers.

The 386 SX Microprocessor address signals are designed to simplify external system hardware. Higher-order address bits are provided by A_{23}–A_1. BHE# and BLE# provide linear selects for the two bytes of the 16-bit data bus.

Byte Enable outputs BHE# and BLE# are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 5.6.

Table 5.6. Byte Enables and Associated Data and Operand Bytes

Byte Enable Signal	Associated Data Bus Signals	
BLE#	D_7–D_0	(byte 0 — least significant)
BHE#	D_{15}–D_8	(byte 1 — most significant)

Each bus cycle is composed of at least two bus states. Each bus state requires one processor clock period. Additional bus states added to a single bus cycle are called wait states. See section **5.4 Bus Functional Description**.

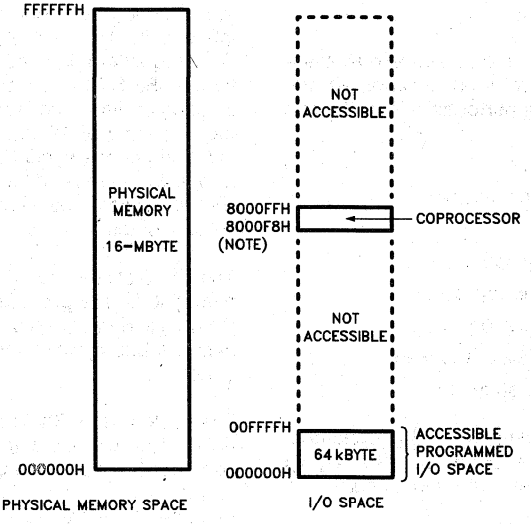
5.3 Memory and I/O Spaces

Bus cycles may access physical memory space or I/O space. Peripheral devices in the system may either be memory-mapped, or I/O-mapped, or both. As shown in Figure 5.3, physical memory addresses range from 000000H to 0FFFFFFH (16 megabytes) and I/O addresses from 000000H to 00FFFFH (64 kilobytes). Note the I/O addresses used by the automatic I/O cycles for coprocessor communication are 8000F8H to 8000FFH, beyond the address range of programmed I/O, to allow easy generation of a coprocessor chip select signal using the A_{23} and M/IO# signals.

5.4 Bus Functional Description

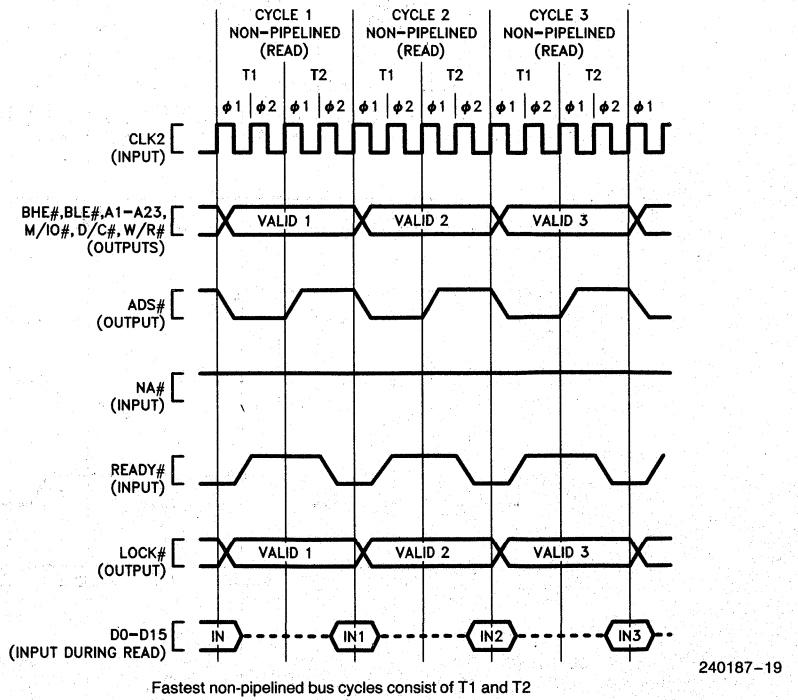
The 386 SX Microprocessor has separate, parallel buses for data and address. The data bus is 16-bits in width, and bidirectional. The address bus provides a 24-bit value using 23 signals for the 23 upper-order address bits and 2 Byte Enable signals to directly indicate the active bytes. These buses are interpreted and controlled by several definition signals.

The definition of each bus cycle is given by three signals: M/IO#, W/R# and D/C#. At the same time, a valid address is present on the byte enable signals, BHE# and BLE#, and the other address signals A_{23}–A_1. A status signal, ADS#, indicates



NOTE:
Since A23 is HIGH during automatic communication with coprocessor, A23 HIGH and M/IO# LOW can be used to easily generate a coprocessor select signal.

Figure 5.3. Physical Memory and I/O Spaces



Fastest non-pipelined bus cycles consist of T1 and T2

Figure 5.4. Fastest Read Cycles with Non-pipelined Address Timing

5

when the 386 SX Microprocessor issues a new bus cycle definition and address.

Collectively, the address bus, data bus and all associated control signals are referred to simply as 'the bus'. When active, the bus performs one of the bus cycles below:

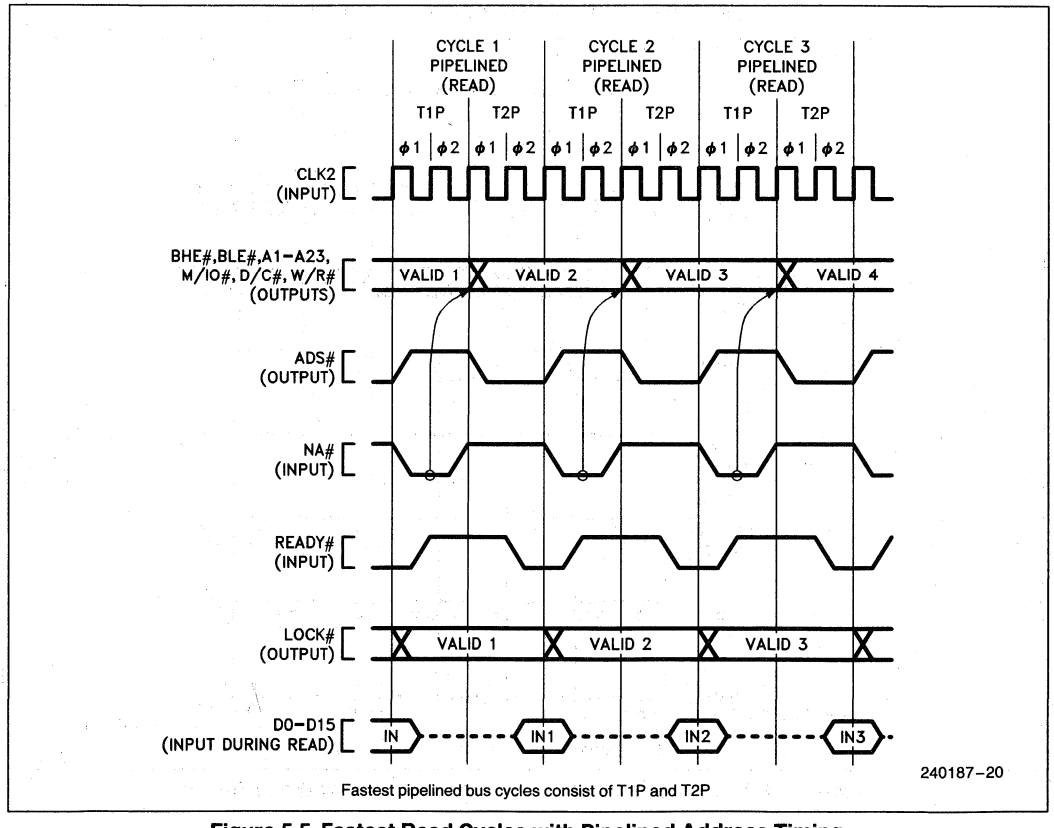
1. Read from memory space
2. Locked read from memory space
3. Write to memory space
4. Locked write to memory space
5. Read from I/O space (or coprocessor)
6. Write to I/O space (or coprocessor)
7. Interrupt acknowledge (always locked)
8. Indicate halt, or indicate shutdown

Table 5.2 shows the encoding of the bus cycle definition signals for each bus cycle. See **Bus Cycle Definition Signals** for additional information.

When the 386 SX Microprocessor bus is not performing one of the activities listed above, it is either Idle or in the Hold Acknowledge state, which may be detected externally. The idle state can be identified by the 386 SX Microprocessor giving no further assertions on its address strobe output (ADS#) since the beginning of its most recent bus cycle, and the most recent bus cycle having been terminated. The hold acknowledge state is identified by the 386 SX Microprocessor asserting its hold acknowledge (HLDA) output.

The shortest time unit of bus activity is a bus state. A bus state is one processor clock period (two CLK2 periods) in duration. A complete data transfer occurs during a bus cycle, composed of two or more bus states.

The fastest 386 SX Microprocessor bus cycle requires only two bus states. For example, three consecutive bus read cycles, each consisting of two bus states, are shown by Figure 5.4. The bus states in each cycle are named T1 and T2. Any memory or I/O address may be accessed by such a two-state bus cycle, if the external hardware is fast enough.



Fastest pipelined bus cycles consist of T1P and T2P

Figure 5.5. Fastest Read Cycles with Pipelined Address Timing

Every bus cycle continues until it is acknowledged by the external system hardware, using the 386 SX Microprocessor READY# input. Acknowledging the bus cycle at the end of the first T2 results in the shortest bus cycle, requiring only T1 and T2. If READY# is not immediately asserted however, T2 states are repeated indefinitely until the READY# input is sampled active.

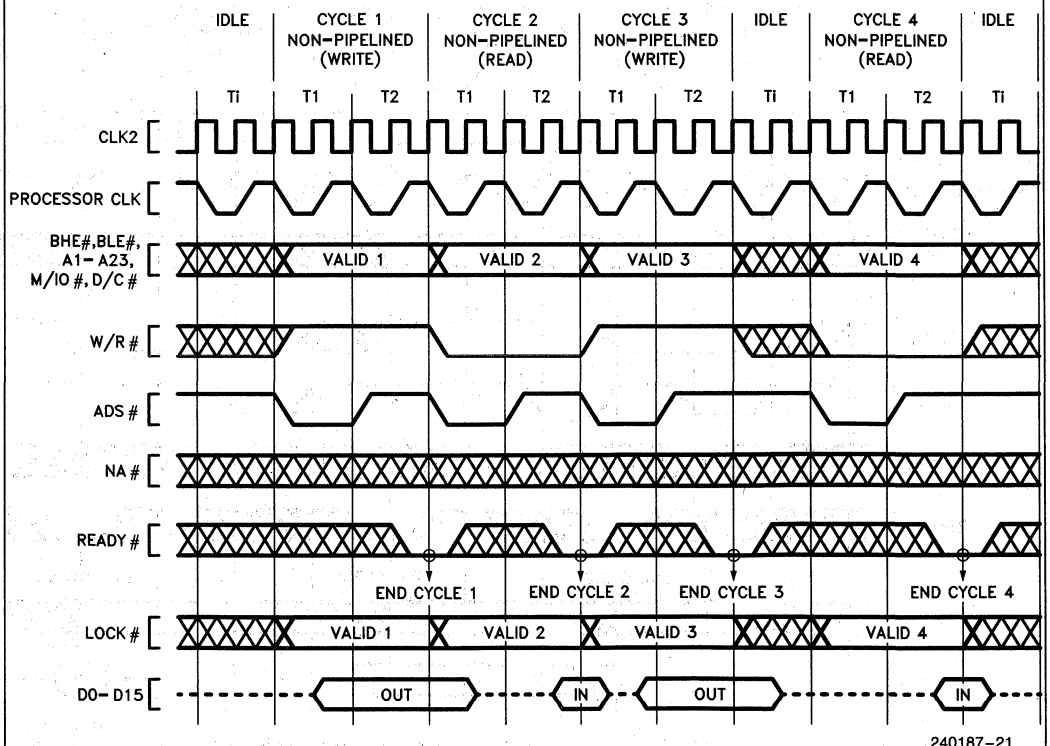
The address pipelining option provides a choice of bus cycle timings. Pipelined or non-pipelined address timing is selectable on a cycle-by-cycle basis with the Next Address (NA#) input.

When address pipelining is selected the address (BHE#, BLE# and A_{23}–A_1) and definition (W/R#, D/C#, M/IO# and LOCK#) of the next cycle are available before the end of the current cycle. To signal their availability, the 386 SX Microprocessor address status output (ADS#) is asserted. Figure 5.5 illustrates the fastest read cycles with pipelined address timing.

Note from Figure 5.5 the fastest bus cycles using pipelined address require only two bus states, named **T1P** and **T2P**. Therefore cycles with pipelined address timing allow the same data bandwidth as non-pipelined cycles, but address-to-data access time is increased by one T-state time compared to that of a non-pipelined cycle.

READ AND WRITE CYCLES

Data transfers occur as a result of bus cycles, classified as read or write cycles. During read cycles, data is transferred from an external device to the processor. During write cycles, data is transferred from the processor to an external device.



Idle states are shown here for diagram variety only. Write cycles are **not** always followed by an idle state. An active bus cycle can immediately follow the write cycle.

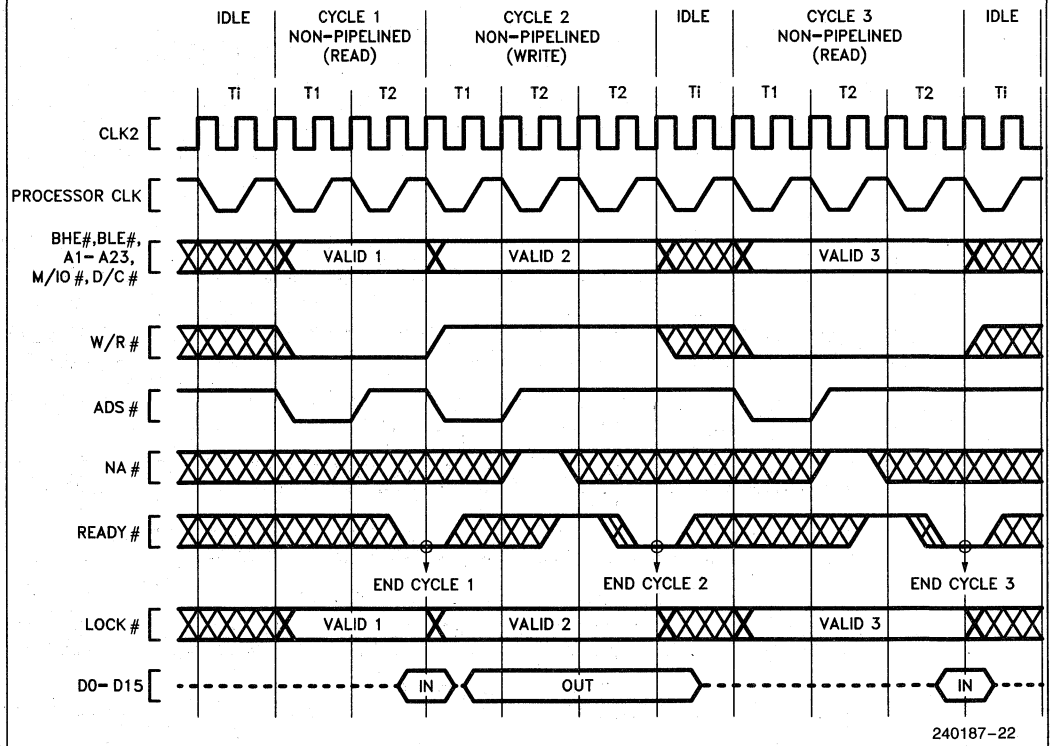
Figure 5.6. Various Bus Cycles with Non-Pipelined Address (zero wait states)

Two choices of address timing are dynamically selectable: non-pipelined or pipelined. After an idle bus state, the processor always uses non-pipelined address timing. However the NA# (Next Address) input may be asserted to select pipelined address timing for the next bus cycle. When pipelining is selected and the 386 SX Microprocessor has a bus request pending internally, the address and definition of the next cycle is made available even before the current bus cycle is acknowledged by READY#.

Terminating a read or write cycle, like any bus cycle, requires acknowledging the cycle by asserting the READY# input. Until acknowledged, the processor inserts wait states into the bus cycle, to allow adjustment for the speed of any external device. External hardware, which has decoded the address and bus cycle type, asserts the READY# input at the appropriate time.

At the end of the second bus state within the bus cycle, READY# is sampled. At that time, if external hardware acknowledges the bus cycle by asserting READY#, the bus cycle terminates as shown in Figure 5.6. If READY# is negated as in Figure 5.7, the 386 SX Microprocessor executes another bus state (a wait state) and READY# is sampled again at the end of that state. This continues indefinitely until the cycle is acknowledged by READY# asserted.

When the current cycle is acknowledged, the 386 SX Microprocessor terminates it. When a read cycle is acknowledged, the 386 SX Microprocessor latches the information present at its data pins. When a write cycle is acknowledged, the 386 SX CPU's write data remains valid throughout phase one of the next bus state, to provide write data hold time.



Idle states are shown here for diagram variety only. Write cycles are **not** always followed by an idle state. An active bus cycle can immediately follow the write cycle.

Figure 5.7. Various Bus Cycles with Non-Pipelined Address (various number of wait states)

Non-Pipelined Address

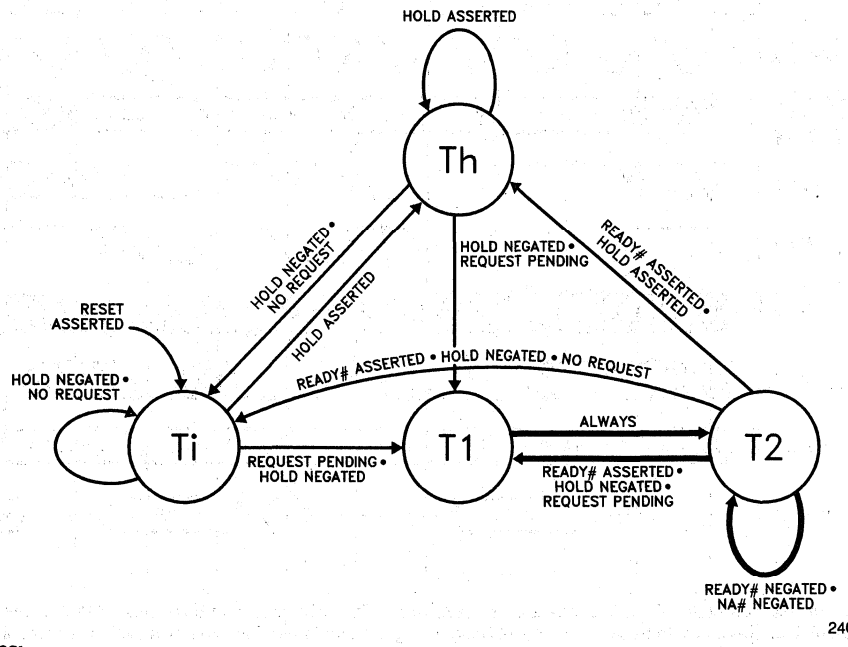
Any bus cycle may be performed with non-pipelined address timing. For example, Figure 5.6 shows a mixture of read and write cycles with non-pipelined address timing. Figure 5.6 shows that the fastest possible cycles with non-pipelined address have two bus states per bus cycle. The states are named T1 and T2. In phase one of T1, the address signals and bus cycle definition signals are driven valid and, to signal their availability, address strobe (ADS#) is simultaneously asserted.

During read or write cycles, the data bus behaves as follows. If the cycle is a read, the 386 SX Microprocessor floats its data signals to allow driving by the external device being addressed. **The 386 SX Microprocessor requires that all data bus pins be at a valid logic state (HIGH or LOW) at the end of each read cycle, when READY# is asserted. The system MUST be designed to meet this requirement.** If the cycle is a write, data signals are driven by the 386 SX Microprocessor beginning in phase two of T1 until phase one of the bus state following cycle acknowledgment.

Figure 5.7 illustrates non-pipelined bus cycles with one wait state added to Cycles 2 and 3. READY# is sampled inactive at the end of the first T2 in Cycles 2 and 3. Therefore Cycles 2 and 3 have T2 repeated again. At the end of the second T2, READY# is sampled active.

When address pipelining is not used, the address and bus cycle definition remain valid during all wait states. When wait states are added and it is desirable to maintain non-pipelined address timing, it is necessary to negate NA# during each T2 state except the last one, as shown in Figure 5.7 Cycles 2 and 3. If NA# is sampled active during a T2 other than the last one, the next state would be T2I or T2P instead of another T2.

When address pipelining is not used, the bus states and transitions are completely illustrated by Figure 5.8. The bus transitions between four possible states, T1, T2, T_i, and T_h. Bus cycles consist of T1 and T2, with T2 being repeated for wait states. Otherwise the bus may be idle, T_i, or in the hold acknowledge state T_h.



Bus States:
T1—first clock of a non-pipelined bus cycle (386™ SX CPU drives new address and asserts ADS#).
T2—subsequent clocks of a bus cycle when NA# has not been sampled asserted in the current bus cycle.
Ti—idle state.
Th—hold acknowledge state (386 SX CPU asserts HLDA).
The fastest bus cycle consists of two states T1 and T2.
Four basic bus states describe bus operation when not using pipelined address.

Figure 5.8. Bus States (not using pipelined address)

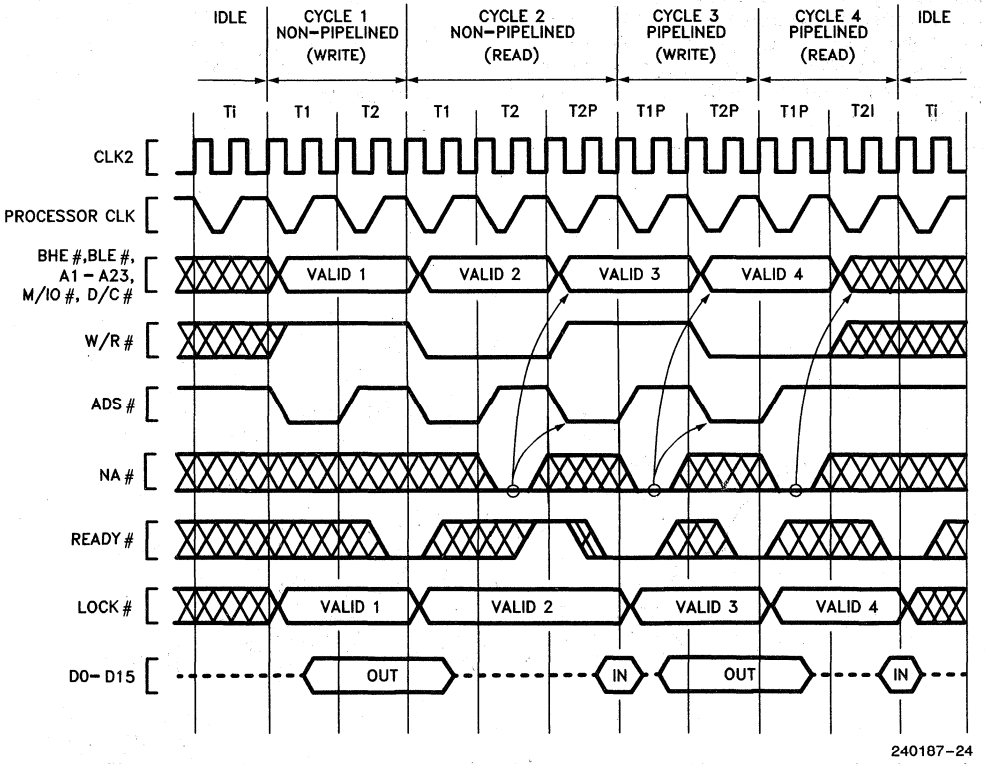
Bus cycles always begin with T1. T1 always leads to T2. If a bus cycle is not acknowledged during T2 and NA# is inactive, T2 is repeated. When a cycle is acknowledged during T2, the following state will be T1 of the next bus cycle if a bus request is pending internally, or T_i if there is no bus request pending, or T_h if the HOLD input is being asserted.

Use of pipelined address allows the 386 SX Microprocessor to enter three additional bus states not shown in Figure 5.8. Figure 5.12 is the complete bus state diagram, including pipelined address cycles.

Pipelined Address

Address pipelining is the option of requesting the address and the bus cycle definition of the next internally pending bus cycle before the current bus cycle is acknowledged with READY# asserted. ADS# is asserted by the 386 SX Microprocessor when the next address is issued. The address pipelining option is controlled on a cycle-by-cycle basis with the NA# input signal.

Once a bus cycle is in progress and the current address has been valid for at least one entire bus state, the NA# input is sampled at the end of every phase one until the bus cycle is acknowledged. During non-pipelined bus cycles NA# is sampled at the end of phase one in every T2. An example is Cycle 2 in Figure 5.9, during which NA# is sampled at the end of phase one of every T2 (it was asserted once during the first T2 and has no further effect during that bus cycle).



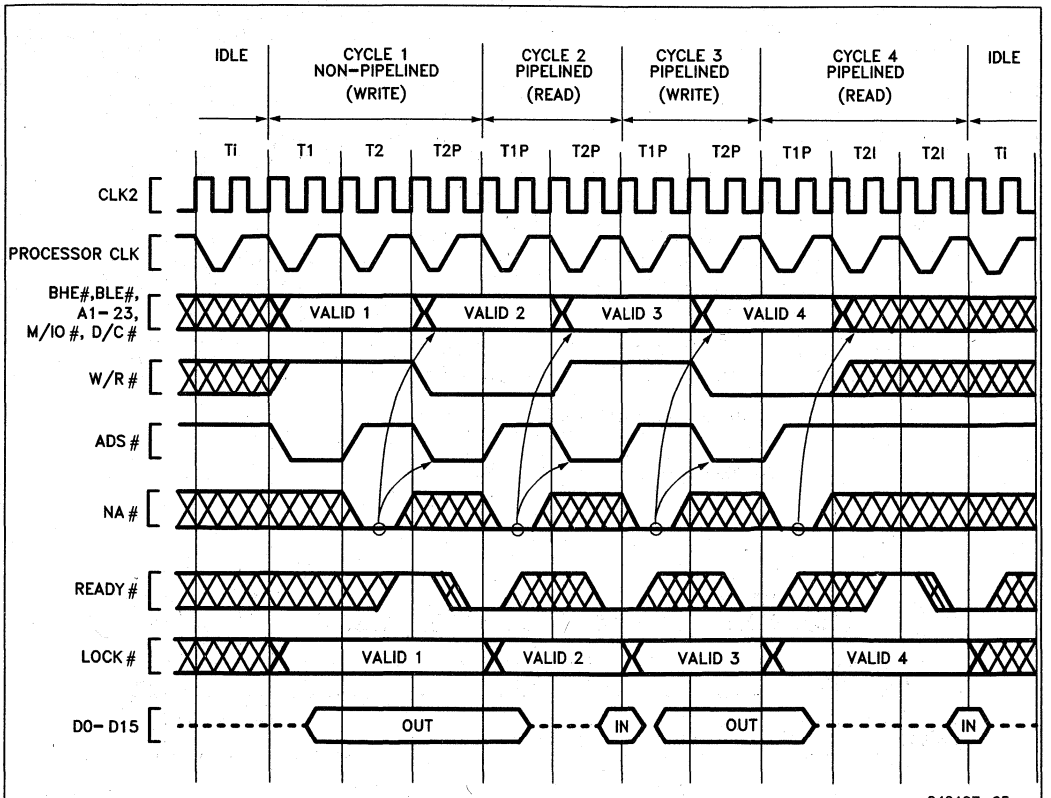
Following any idle bus state (Ti), addresses are non-pipelined. Within non-pipelined bus cycles, NA# is only sampled during wait states. Therefore, to begin address pipelining during a group of non-pipelined bus cycles requires a non-pipelined cycle with at least one wait state (Cycle 2 above).

Figure 5.9. Transitioning to Pipelined Address During Burst of Bus Cycles

If NA# is sampled active, the 386 SX Microprocessor is free to drive the address and bus cycle definition of the next bus cycle, and assert ADS#, as soon as it has a bus request internally pending. It may drive the next address as early as the next bus state, whether the current bus cycle is acknowledged at that time or not.

Regarding the details of address pipelining, the 386 SX Microprocessor has the following characteristics:

1. The next address may appear as early as the bus state after NA# was sampled active (see Figures 5.9 or 5.10). In that case, state T2P is entered immediately. However, when there is not an internal bus request already pending, the next address will not be available immediately after NA# is asserted and T2I is entered instead of T2P (see Figure 5.11 Cycle 3). Provided the current bus cycle isn't yet acknowledged by READY# asserted, T2P will be entered as soon as the 386 SX Microprocessor does drive the next address. External hardware should therefore observe the ADS# output as confirmation the next address is actually being driven on the bus.
2. Any address which is validated by a pulse on the ADS# output will remain stable on the address pins for at least two processor clock periods. The 386 SX Microprocessor cannot produce a new address more frequently than every two processor clock periods (see Figures 5.9, 5.10, and 5.11).
3. Only the address and bus cycle definition of the very next bus cycle is available. The pipelining capability cannot look further than one bus cycle ahead (see Figure 5.11 Cycle 1).



Following any bus state (Ti) the address is always non-pipelined and NA# is only sampled during wait states. To start address pipelining after an idle state requires a non-pipelined cycle with at least one wait state (cycle 1 above) The pipelined cycles (2, 3, 4 above) are shown with various numbers of wait states.

Figure 5.10. Fastest Transition to Pipelined Address Following Idle Bus State

The complete bus state transition diagram, including operation with pipelined address is given by Figure 5.12. Note it is a superset of the diagram for non-pipelined address only, and the three additional bus states for pipelined address are drawn in bold.

The fastest bus cycle with pipelined address consists of just two bus states, T1P and T2P (recall for non-pipelined address it is T1 and T2). T1P is the first bus state of a pipelined cycle.

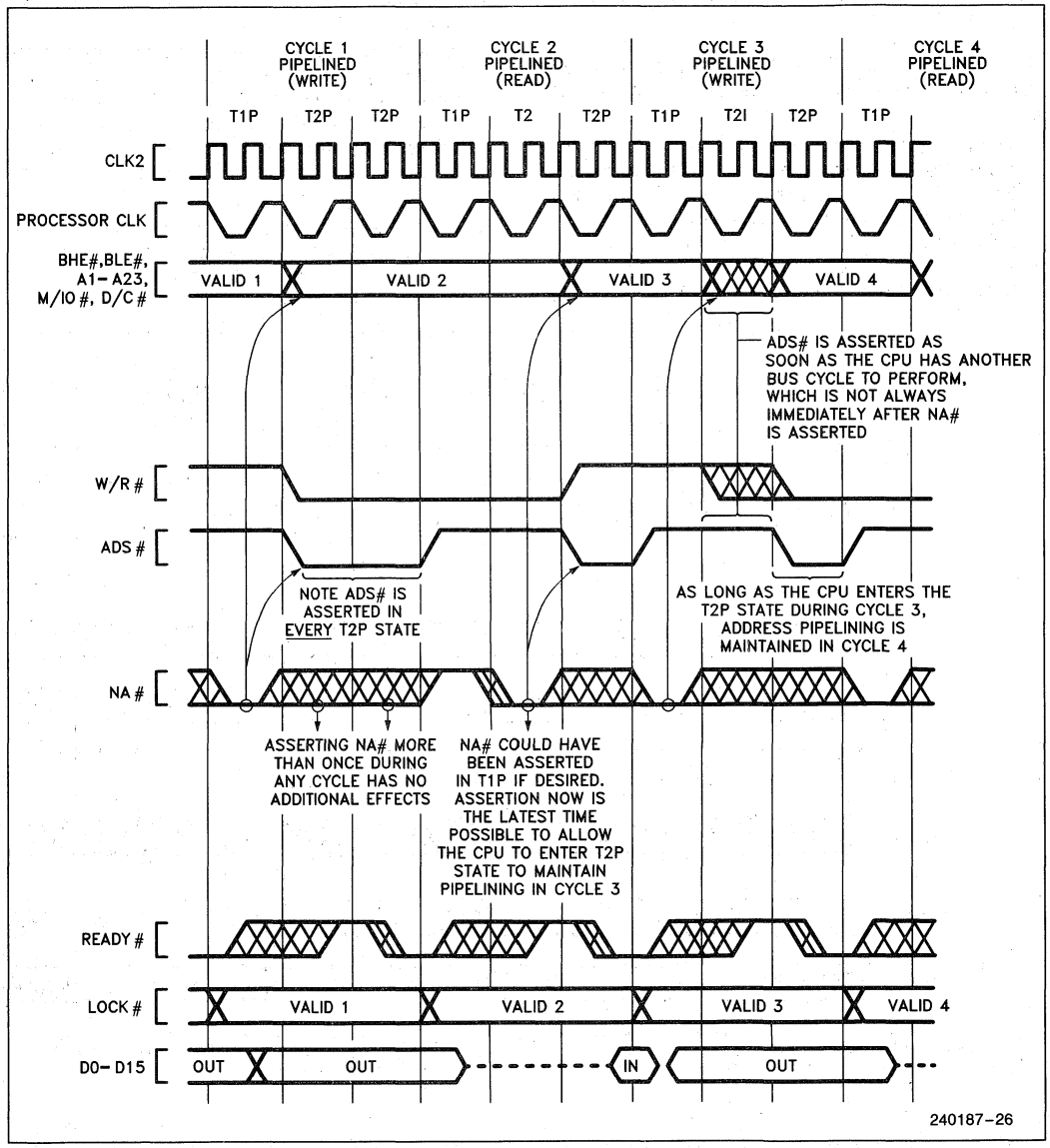
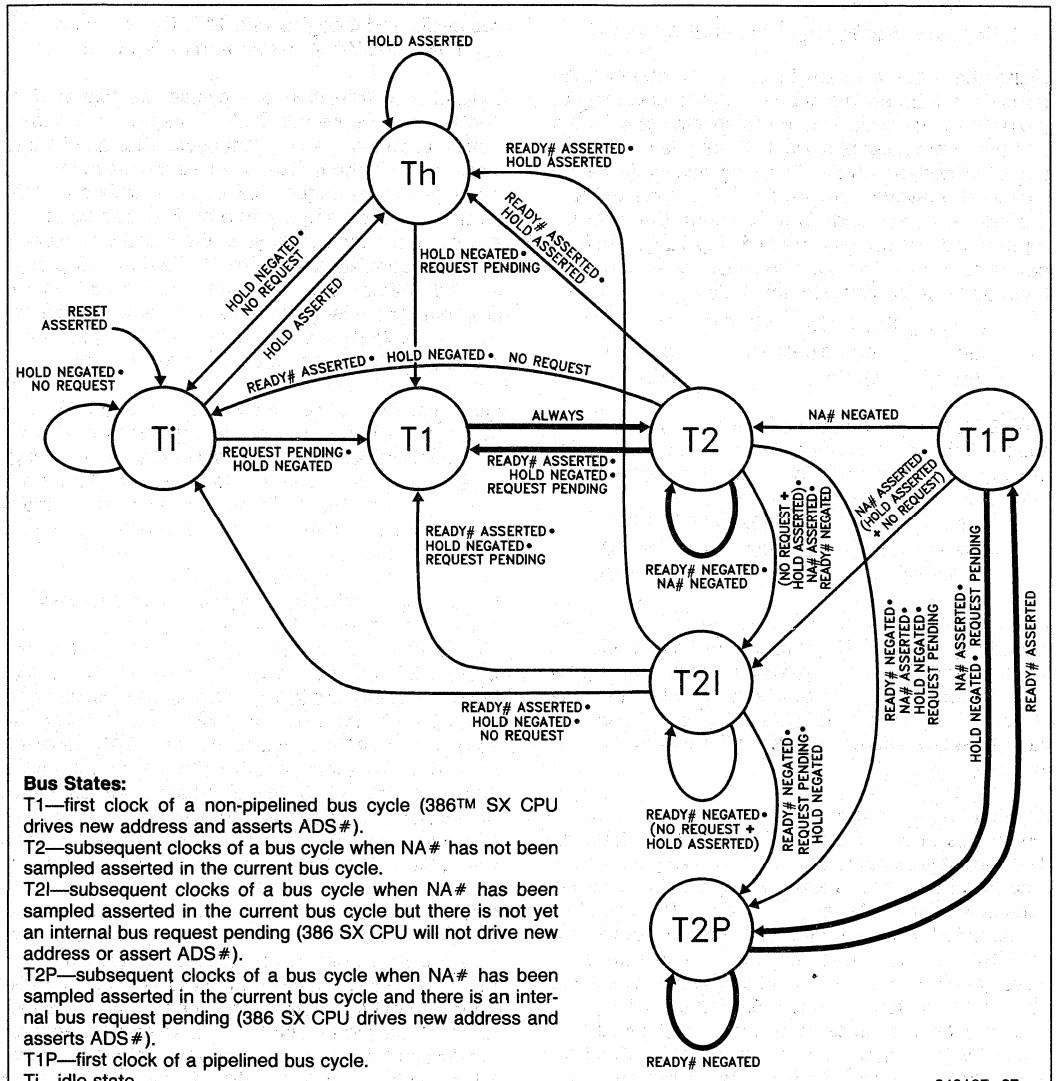


Figure 5.11. Details of Address Pipelining During Cycles with Wait States



Bus States:
T1—first clock of a non-pipelined bus cycle (386™ SX CPU drives new address and asserts ADS#).
T2—subsequent clocks of a bus cycle when NA# has not been sampled asserted in the current bus cycle.
T2I—subsequent clocks of a bus cycle when NA# has been sampled asserted in the current bus cycle but there is not yet an internal bus request pending (386 SX CPU will not drive new address or assert ADS#).
T2P—subsequent clocks of a bus cycle when NA# has been sampled asserted in the current bus cycle and there is an internal bus request pending (386 SX CPU drives new address and asserts ADS#).
T1P—first clock of a pipelined bus cycle.
Ti—idle state.
Th—hold acknowledge state (386 SX CPU asserts HLDA).
Asserting NA# for pipelined address gives access to three more bus states: T2I, T2P and T1P.
Using pipelined address, the fastest bus cycle consists of T1P and T2P.

Figure 5.12. Complete Bus States (including pipelined address)

Initiating and Maintaining Pipelined Address

Using the state diagram Figure 5.12, observe the transitions from an idle state, T_i, to the beginning of a pipelined bus cycle T1P. From an idle state, T_i, the first bus cycle must begin with T1, and is therefore a non-pipelined bus cycle. The next bus cycle will be pipelined, however, provided NA# is asserted and the first bus cycle ends in a T2P state (the address for the next bus cycle is driven during T2P). The fastest path from an idle state to a bus cycle with pipelined address is shown in bold below:

T_i, T_i, T_i, T1 - T2 - T2P, T1P - T2P,

idle	non-pipelined	pipelined
states	cycle	cycle

T1-T2-T2P are the states of the bus cycle that establish address pipelining for the next bus cycle, which begins with T1P. The same is true after a bus hold state, shown below:

T_h, T_h, T_h, T1 - T2 - T2P, T1P - T2P,

hold acknowledge	non-pipelined	pipelined
states	cycle	cycle

The transition to pipelined address is shown functionally by Figure 5.10 Cycle 1. Note that Cycle 1 is used to transition into pipelined address timing for the subsequent Cycles 2, 3 and 4, which are pipelined. The NA# input is asserted at the appropriate time to select address pipelining for Cycles 2, 3 and 4.

Once a bus cycle is in progress and the current address has been valid for one entire bus state, the NA# input is sampled at the end of every phase one until the bus cycle is acknowledged. Sampling begins in T2 during Cycle 1 in Figure 5.10. Once NA# is sampled active during the current cycle, the 386 SX Microprocessor is free to drive a new address and bus cycle definition on the bus as early as the next bus state. In Figure 5.10 Cycle 1 for example, the next address is driven during state T2P. Thus Cycle 1 makes the transition to pipelined address timing, since it begins with T1 but ends with T2P. Because the address for Cycle 2 is available before Cycle 2 begins, Cycle 2 is called a pipelined bus cycle, and it begins with T1P. Cycle 2 begins as soon as READY# asserted terminates Cycle 1.

Examples of transition bus cycles are Figure 5.10 Cycle 1 and Figure 5.9 Cycle 2. Figure 5.10 shows transition during the very first cycle after an idle bus state, which is the fastest possible transition into address pipelining. Figure 5.9 Cycle 2 shows a transition cycle occurring during a burst of bus cycles. In any case, a transition cycle is the same whenever it occurs: it consists at least of T1, T2 (NA# is asserted at that time), and T2P (provided the 386 SX Microprocessor has an internal bus request already pending, which it almost always has). T2P states are repeated if wait states are added to the cycle.

Note that only three states (T1, T2 and T2P) are required in a bus cycle performing a **transition** from non-pipelined address into pipelined address timing, for example Figure 5.10 Cycle 1. Figure 5.10 Cycles 2, 3 and 4 show that address pipelining can be maintained with two-state bus cycles consisting only of T1P and T2P.

Once a pipelined bus cycle is in progress, pipelined timing is maintained for the next cycle by asserting NA# and detecting that the 386 SX Microprocessor enters T2P during the current bus cycle. The current bus cycle must end in state T2P for pipelining to be maintained in the next cycle. T2P is identified by the assertion of ADS#. Figures 5.9 and 5.10 however, each show pipelining ending after Cycle 4 because Cycle 4 ends in T2I. This indicates the 386 SX Microprocessor didn't have an internal bus request prior to the acknowledgement of Cycle 4. If a cycle ends with a T2 or T2I, the next cycle will not be pipelined.

Realistically, address pipelining is almost always maintained as long as NA# is sampled asserted. This is so because in the absence of any other request, a code prefetch request is always internally pending until the instruction decoder and code prefetch queue are completely full. Therefore, address pipelining is maintained for long bursts of bus cycles, if the bus is available (i.e., HOLD inactive) and NA# is sampled active in each of the bus cycles.

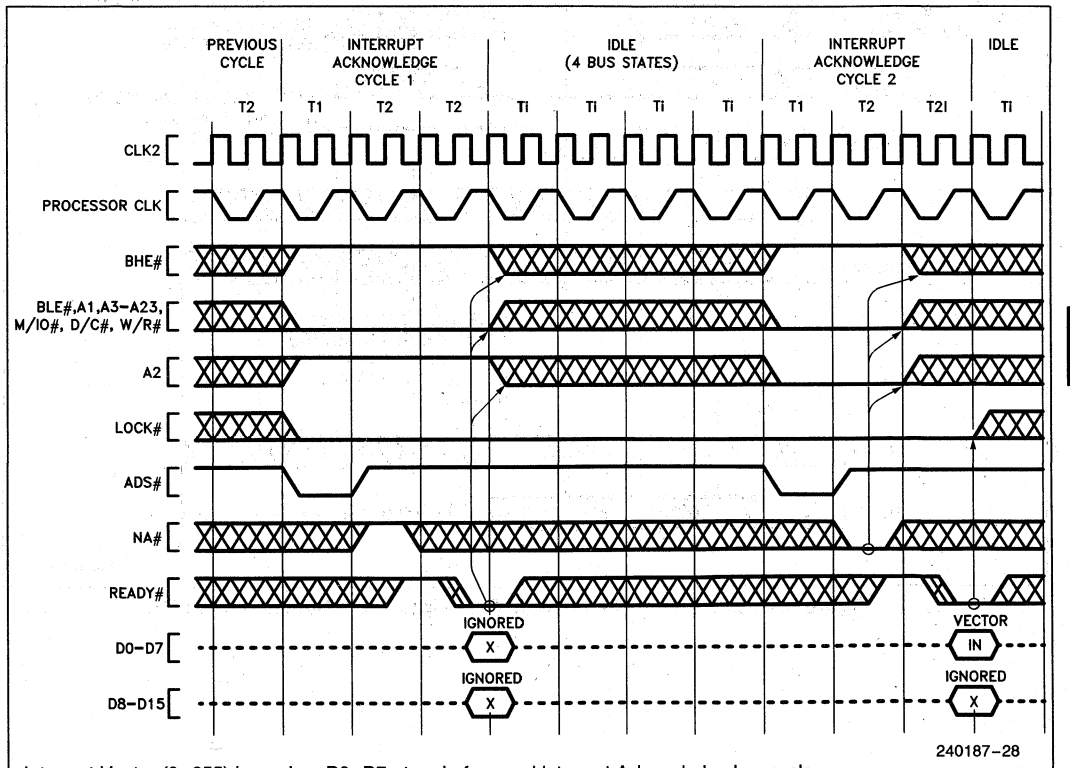
INTERRUPT ACKNOWLEDGE (INTA) CYCLES

In response to an interrupt request on the INTR input when interrupts are enabled, the 386 SX Microprocessor performs two interrupt acknowledge cycles. These bus cycles are similar to read cycles in that bus definition signals define the type of bus activity taking place, and each cycle continues until acknowledged by READY# sampled active.

The state of A_2 distinguishes the first and second interrupt acknowledge cycles. The byte address driven during the first interrupt acknowledge cycle is 4 (A_{23}–A_3, A_1, BLE# LOW, A_2 and BHE# HIGH). The byte address driven during the second interrupt acknowledge cycle is 0 (A_{23}–A_1, BLE# LOW, and BHE# HIGH).

The LOCK# output is asserted from the beginning of the first interrupt acknowledge cycle until the end of the second interrupt acknowledge cycle. Four idle bus states, T_i, are inserted by the 386 SX Microprocessor between the two interrupt acknowledge cycles for compatibility with spec TRHRL of the 8259A Interrupt Controller.

During both interrupt acknowledge cycles, D_{15}–D_0 float. No data is read at the end of the first interrupt acknowledge cycle. At the end of the second interrupt acknowledge cycle, the 386 SX Microprocessor will read an external interrupt vector from D_7–D_0 of the data bus. The vector indicates the specific interrupt number (from 0–255) requiring service.



Interrupt Vector (0–255) is read on D0–D7 at end of second interrupt Acknowledge bus cycle.
Because each Interrupt Acknowledge bus cycle is followed by idle bus states. asserting NA# has no practical effect. Choose the approach which is simplest for your system hardware design.

Figure 5.13. Interrupt Acknowledge Cycles

HALT INDICATION CYCLE

The execution unit halts as a result of executing a HLT instruction. Signaling its entrance into the halt state, a halt indication cycle is performed. The halt indication cycle is identified by the state of the bus definition signals shown on page 40, **Bus Cycle Definition Signals**, and an address of 2. The halt indication cycle must be acknowledged by READY# asserted. A halted 386 SX Microprocessor resumes execution when INTR (if interrupts are enabled), NMI or RESET is asserted.

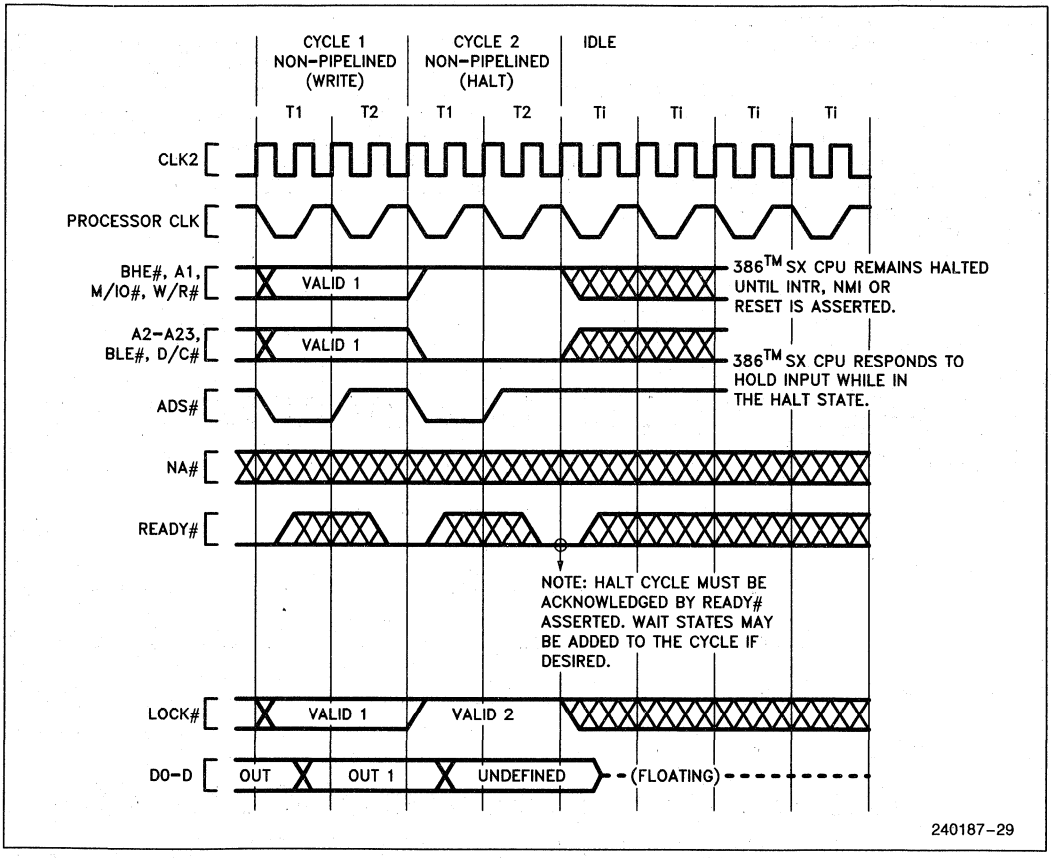


Figure 5.14. Example Halt Indication Cycle from Non-Pipelined Cycle

SHUTDOWN INDICATION CYCLE

The 386 SX Microprocessor shuts down as a result of a protection fault while attempting to process a double fault. Signaling its entrance into the shutdown state, a shutdown indication cycle is performed. The shutdown indication cycle is identified by the state of the bus definition signals shown in **Bus Cycle Definition Signals** and an address of 0. The shutdown indication cycle must be acknowledged by READY# asserted. A shutdown 386 SX Microprocessor resumes execution when NMI or RESET is asserted.

ENTERING AND EXITING HOLD ACKNOWLEDGE

The bus hold acknowledge state, T_h, is entered in response to the HOLD input being asserted. In the bus hold acknowledge state, the 386 SX Microprocessor floats all outputs or bidirectional signals, except for HLDA. HLDA is asserted as long as the 386 SX Microprocessor remains in the bus hold acknowledge state. In the bus hold acknowledge state, all inputs except HOLD, FLT# and RESET are ignored.

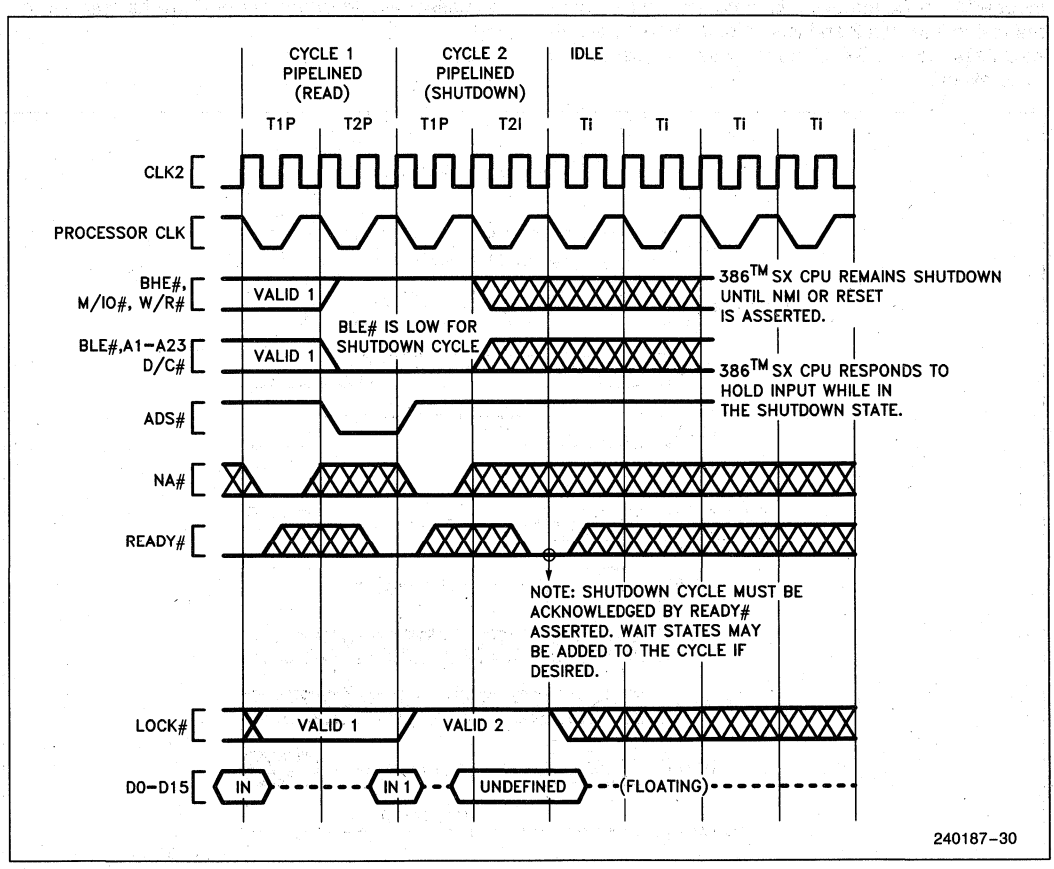


Figure 5.15. Example Shutdown Indication Cycle from Non-Pipelined Cycle

T_h may be entered from a bus idle state as in Figure 5.16 or after the acknowledgement of the current physical bus cycle if the LOCK# signal is not asserted, as in Figures 5.17 and 5.18.

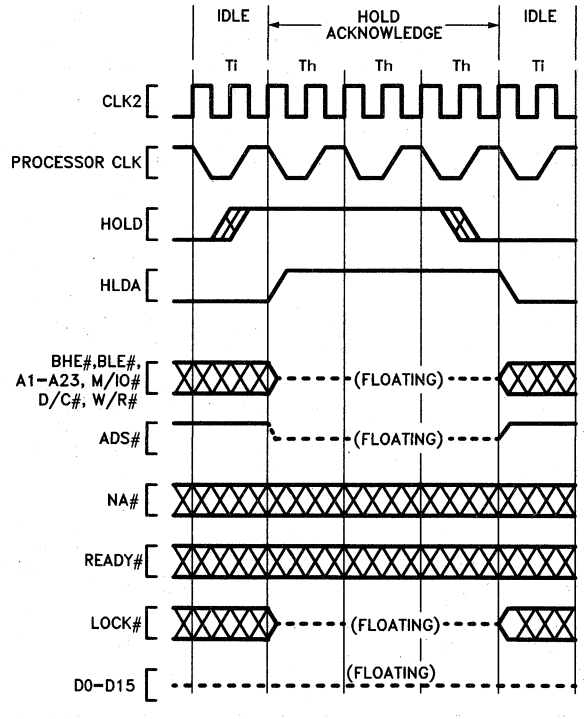
T_h is exited in response to the HOLD input being negated. The following state will be T_i as in Figure 5.16 if no bus request is pending. The following bus state will be T1 if a bus request is internally pending, as in Figures 5.17 and 5.18. T_h is exited in response to RESET being asserted.

If a rising edge occurs on the edge-triggered NMI input while in T_h, the event is remembered as a non-maskable interrupt 2 and is serviced when T_h is exited unless the 386 SX Microprocessor is reset before T_h is exited.

RESET DURING HOLD ACKNOWLEDGE

RESET being asserted takes priority over HOLD being asserted. If RESET is asserted while HOLD remains asserted, the 386 SX Microprocessor drives its pins to defined states during reset, as in **Table 5.5 Pin State During Reset**, and performs internal reset activity as usual.

If HOLD remains asserted when RESET is inactive, the 386 SX Microprocessor enters the hold acknowledge state before performing its first bus cycle, provided HOLD is still asserted when the 386 SX Microprocessor would otherwise perform its first bus cycle.



NOTE:
For maximum design flexibility the 386™ SX CPU has no internal pullup resistors on its outputs. Your design may require an external pullup on ADS# and other outputs to keep them negated during float periods.

Figure 5.16. Requesting Hold from Idle Bus

FLOAT

Activating the FLT# input floats all 386 SX bidirectional and output signals, including HLDA. Asserting FLT# isolates the 386 SX from the surrounding circuitry.

As the 386 SX is packaged in a surface mount PQFP, it cannot be removed from the motherboard when In-Circuit Emulation (ICE) is needed. The FLT# input allows the 386 SX to be electrically isolated from the surrounding circuitry. This allows connection of an emulator to the 386 SX PQFP without removing it from the PCB. This method of emulation is referred to as ON-Circuit Emulation (ONCE).

ENTERING AND EXITING FLOAT

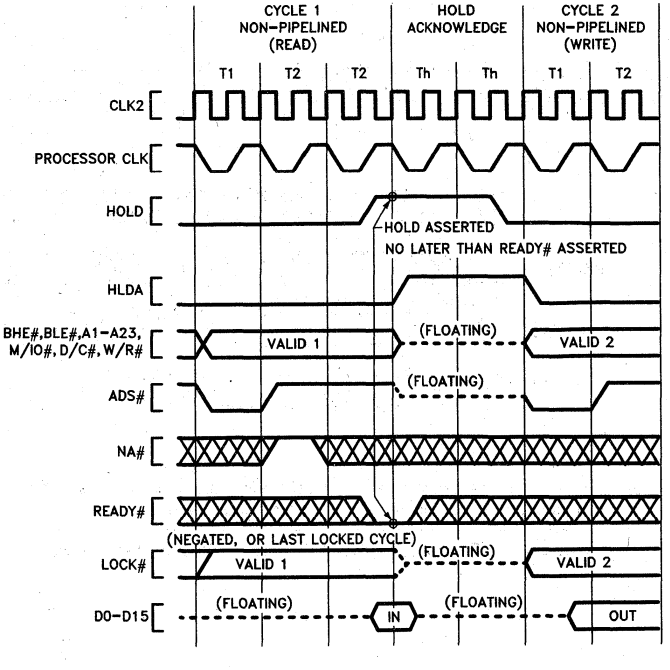
FLT# is an asynchronous, active-low input. It is recognized on the rising edge of CLK2. When recognized, it aborts the current bus cycle and floats the outputs of the 386 SX (Figure 5.20). FLT# must be held low for a minimum of 16 CLK2 cycles. Reset should be asserted and held asserted until after FLT# is deasserted. This will ensure that the 386 SX will exit float in a valid state.

Asserting the FLT# input unconditionally aborts the current bus cycle and forces the 386 SX into the FLOAT mode. Since activating FLT# unconditionally forces the 386 SX into FLOAT mode, the 386 SX is not guaranteed to enter FLOAT in a valid state. After deactivating FLT#, the 386 SX is not guaranteed to exit FLOAT mode in a valid state. This is not a problem as the FLT# pin is meant to be used only during ONCE. After exiting FLOAT, the 386 SX must be reset to return it to a valid state. Reset should be asserted before FLT# is deasserted. This will ensure that the 386 SX will exit float in a valid state.

FLT# has an internal pull-up resistor, and if it is not used it should be unconnected.

BUS ACTIVITY DURING AND FOLLOWING RESET

RESET is the highest priority input signal, capable of interrupting any processor activity when it is asserted. A bus cycle in progress can be aborted at any stage, or idle states or bus hold acknowledge states discontinued so that the reset state is established.



NOTE:
HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold (t_{23} and t_{24}) requirements are met. This waveform is useful for determining Hold Acknowledge latency.

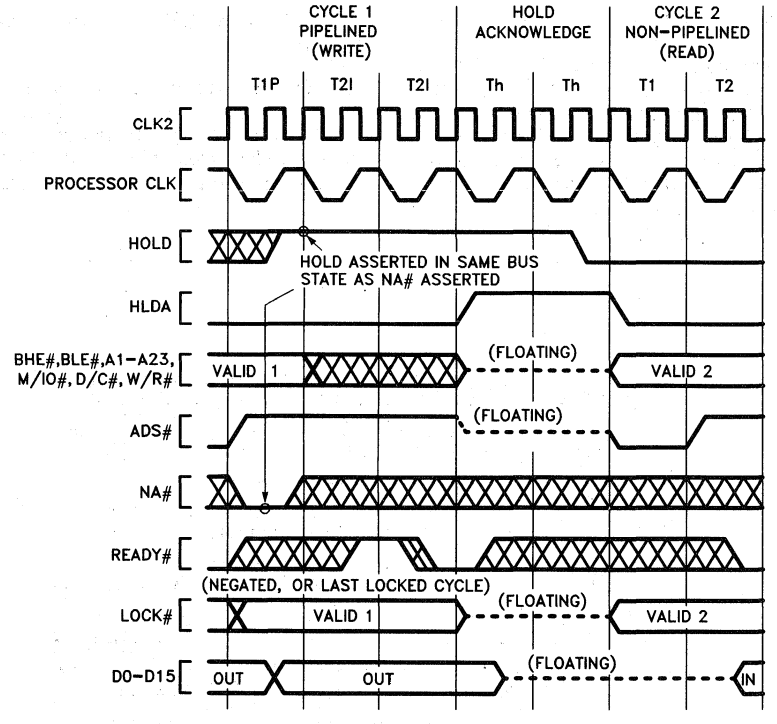
Figure 5.17. Requesting Hold from Active Bus (NA# inactive)

RESET should remain asserted for at least 15 CLK2 periods to ensure it is recognized throughout the 386 SX Microprocessor, and at least 80 CLK2 periods if self-test is going to be requested at the falling edge. RESET asserted pulses less than 15 CLK2 periods may not be recognized. RESET pulses less than 80 CLK2 periods followed by a self-test may cause the self-test to report a failure when no true failure exists.

Provided the RESET falling edge meets setup and hold times t_{25} and t_{26}, the internal processor clock phase is defined at that time as illustrated by Figure 5.19 and Figure 7.7.

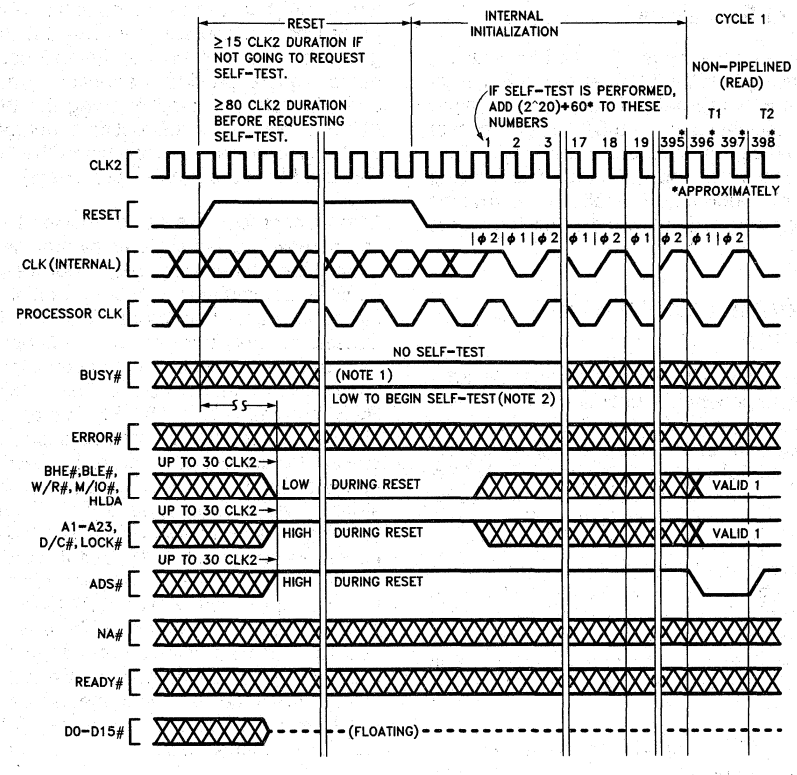
A self-test may be requested at the time RESET goes inactive by having the BUSY# input at a LOW level as shown in Figure 5.19. The self-test requires approximately (2^{20} + 60) CLK2 periods to complete. The self-test duration is not affected by the test results. Even if the self-test indicates a problem, the 386 SX Microprocessor attempts to proceed with the reset sequence afterwards.

After the RESET falling edge (and after the self-test if it was requested) the 386 SX Microprocessor performs an internal initialization sequence for approximately 350 to 450 CLK2 periods.



NOTE:
HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold (t23 and t24) requirements are met. This waveform is useful for determining Hold Acknowledge latency.

Figure 5.18. Requesting Hold from Idle Bus (NA# active)



NOTES:
1. BUSY# should be held stable for 8 CLK2 periods before and after the CLK2 period in which RESET falling edge occurs.
2. If self-test is requested the outputs remain in their reset state as shown here.

Figure 5.19. Bus Activity from Reset Until First Code Fetch

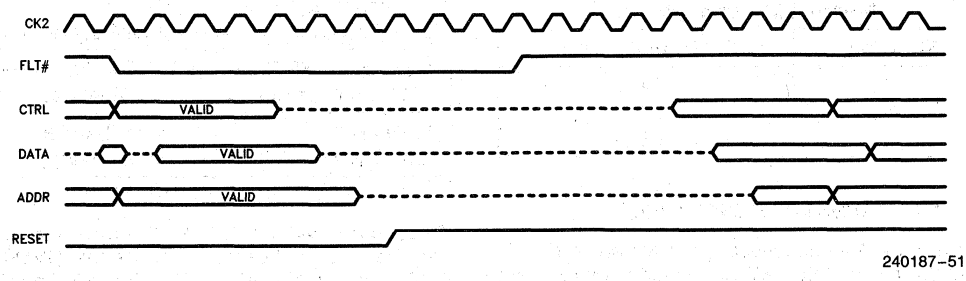


Figure 5.20. Entering and Exiting, FLT#

5.5 Self-test Signature

Upon completion of self-test (if self-test was requested by driving BUSY# LOW at the falling edge of RESET) the EAX register will contain a signature of 00000000H indicating the 386 SX Microprocessor passed its self-test of microcode and major PLA contents with no problems detected. The passing signature in EAX, 00000000H, applies to all revision levels. Any non-zero signature indicates the unit is faulty.

5.6 Component and Revision Identifiers

To assist users, the 386 SX Microprocessor after reset holds a component identifier and revision identifier in its DX register. The upper 8 bits of DX hold 23H as identification of the 386 SX Microprocessor (the lower nibble, 03H, refers to the Intel386 DX Architecture. The upper nibble, 02H, refers to the second member of the Intel386 DX Family). The lower 8 bits of DX hold an 8-bit unsigned binary number related to the component revision level. The revision identifier will, in general, chronologically track those component steppings which are intended to have certain improvements or distinction from previous steppings. The 386 SX Microprocessor revision identifier will track that of the 386 DX CPU where possible.

The revision identifier is intended to assist users to a practical extent. However, the revision identifier value is not guaranteed to change with every stepping revision, or to follow a completely uniform numerical sequence, depending on the type or intention of revision, or manufacturing materials required to be changed. Intel has sole discretion over these characteristics of the component.

Table 5.7. Component and Revision Identifier History

Stepping	Revision Identifier
A0	04H
B	05H
C	08H

5.7 Coprocessor Interfacing

The 386 SX Microprocessor provides an automatic interface for the Intel 387 SX numeric floating-point coprocessor. The 387 SX coprocessor uses an I/O mapped interface driven automatically by the 386 SX Microprocessor and assisted by three dedicated signals: BUSY#, ERROR# and PEREQ.

As the 386 SX Microprocessor begins supporting a coprocessor instruction, it tests the BUSY# and ERROR# signals to determine if the coprocessor can accept its next instruction. Thus, the BUSY# and ERROR# inputs eliminate the need for any 'preamble' bus cycles for communication between processor and coprocessor. The 387™ SX can be given its command opcode immediately. The dedicated signals provide instruction synchronization, and eliminate the need of using the WAIT opcode (9BH) for 387™ SX instruction synchronization (the WAIT opcode was required when the 8086 or 8088 was used with the 8087 coprocessor).

Custom coprocessors can be included in 386 SX Microprocessor based systems by memory-mapped or I/O-mapped interfaces. Such coprocessor interfaces allow a completely custom protocol, and are not limited to a set of coprocessor protocol 'primitives'. Instead, memory-mapped or I/O-mapped interfaces may use all applicable instructions for high-speed coprocessor communication. The BUSY# and ERROR# inputs of the 386 SX Microprocessor may also be used for the custom coprocessor interface, if such hardware assist is desired. These signals can be tested by the WAIT opcode (9BH). The WAIT instruction will wait until the BUSY# input is inactive (interruptable by an NMI or enabled INTR input), but generates an exception 16 fault if the ERROR# pin is active when the BUSY# goes (or is) inactive. If the custom coprocessor interface is memory-mapped, protection of the addresses used for the interface can be provided with the 386 SX CPU's on-chip paging or segmentation mechanisms. If the custom interface is I/O-mapped, protection of the interface can be provided with the IOPL (I/O Privilege Level) mechanism.

The 387™ SX numeric coprocessor interface is I/O mapped as shown in Table 5.8. Note that the 387™ SX coprocessor interface addresses are beyond the 0H-0FFFFH range for programmed I/O. When the 386 SX Microprocessor supports the 387™ SX coprocessor, the 386 SX Microprocessor automatically generates bus cycles to the coprocessor interface addresses.

Table 5.8. Numeric Coprocessor Port Addresses

Address in 386™ SX CPU I/O Space	387™ SX Coprocessor Register
8000F8H	Opcode Register
8000FCH/8000FEH*	Operand Register

*Generated as 2nd bus cycle during Dword transfer.

To correctly map the 387™ SX registers to the appropriate I/O addresses, connect the CMD0 and CMD1 lines of the 387™ SX as listed in Table 5.9.

Table 5.9. Connections for CMD0 and CMD1 Inputs for the 387™ SX

Signal	Connection
CMD0	Connect directly to 386™ SX CPU A2 signal
CMD1	Connect to ground.

Software Testing for Coprocessor Presence

When software is used to test for coprocessor (387 SX) presence, it should use only the following coprocessor opcodes: FINIT, FNINIT, FSTCW mem, FSTSW mem and FSTSW AX. To use other coprocessor opcodes when a coprocessor is known to be not present, first set EM = 1 in the 386 SX CPU's CR0 register.

6.0 PACKAGE THERMAL SPECIFICATIONS

The 386 SX Microprocessor is specified for operation when case temperature is within the range of 0°C–100°C. The case temperature may be measured in any environment, to determine whether the 386 SX Microprocessor is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

The ambient temperature is guaranteed as long as T_c is not violated. The ambient temperature can be calculated from the θ_{jc} and θ_{ja} from the following equations:

$$T_j = T_c + P*\theta_{jc}$$

$$T_a = T_j - P*\theta_{ja}$$

$$T_c = T_a + P*[\theta_{ja} - \theta_{jc}]$$

Values for θ_{ja} and θ_{jc} are given in table 6.1 for the 100 lead fine pitch. θ_{ja} is given at various airflows. Table 6.2 shows the maximum T_a allowable (without exceeding T_c) at various airflows. Note that T_a can be improved further by attaching 'fins' or a 'heat sink' to the package.

7.0 ELECTRICAL SPECIFICATIONS

The following sections describe recommended electrical connections for the 386 SX Microprocessor, and its electrical specifications.

7.1 Power and Grounding

The 386 SX Microprocessor is implemented in CHMOS IV technology and has modest power requirements. However, its high clock frequency and 47 output buffers (address, data, control, and HLDA) can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 14 Vcc and 18 Vss pins separately feed functional units of the 386 SX Microprocessor.

Power and ground connections must be made to all external Vcc and V_{SS} pins of the 386 SX Microprocessor. On the circuit board, all Vcc pins should be connected on a Vcc plane and all Vss pins should be connected on a GND plane.

POWER DECOUPLING RECOMMENDATIONS

Liberal decoupling capacitors should be placed near the 386 SX Microprocessor. The 386 SX Microprocessor driving its 24-bit address bus and 16-bit data bus at high frequencies can cause transient power surges, particularly when driving large capacitive loads. Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the 386 SX Microprocessor and decoupling capacitors as much as possible.

5

Table 6.1. Thermal Resistances (°C/Watt) θ_{jc} and θ_{ja}.

Package	θjc	θ_{ja} versus Airflow - ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
100 Lead Fine Pitch	7.5	34.5	29.5	25.5	22.5	21.5	21

Table 6.2. Maximum T_a at various airflows.

Package	Frequency	T_A(°C) versus Airflow - ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
100L PQFP Fine Pitch	16 MHz	74	80	83	86	89	90
	20 MHz	70	77	80	84	87	88

NOTE:
The numbers in Table 6.2 were calculated using an I_{CC} of 200 mA at 16 MHz and 230 mA at 20 MHz, which is representative of the worst case I_{CC} at T_C = 100°C with the outputs unloaded.

Table 7.1. Recommended Resistor Pull-ups to Vcc

Pin	Signal	Pull-up Value	Purpose
16	ADS#	20 K-Ohm ± 10%	Lightly pull ADS# inactive during 386™ SX CPU hold acknowledge states
26	LOCK#	20 K-Ohm ± 10%	Lightly pull LOCK# inactive during 386™ SX CPU hold acknowledge states

RESISTOR RECOMMENDATIONS

The ERROR#, FLT# and BUSY# inputs have internal pull-up resistors of approximately 20 K-Ohms and the PEREQ input has an internal pull-down resistor of approximately 20 K-Ohms built into the 386 SX Microprocessor to keep these signals inactive when the 387 SX is not present in the system (or temporarily removed from its socket).

In typical designs, the external pull-up resistors shown in Table 7.1 are recommended. However, a particular design may have reason to adjust the resistor values recommended here, or alter the use of pull-up resistors in other ways.

OTHER CONNECTION RECOMMENDATIONS

For reliable operation, always connect unused inputs to an appropriate signal level. N/C pins should always remain **unconnected. Connection of N/C pins to Vcc or Vss will result in component malfunction or incompatibility with future steppings of the 386 SX Microprocessor.**

Particularly when not using interrupts or bus hold (as when first prototyping), prevent any chance of spurious activity by connecting these associated inputs to GND:

Pin	Signal
40	INTR
38	NMI
4	HOLD

If not using address pipelining, connect pin 6, NA#, through a pull-up in the range of 20 K-Ohms to Vcc.

7.2 Maximum Ratings

Table 7.2. Maximum Ratings

Parameter	Maximum Rating
Storage temperature	−65 °C to 150 °C
Case temperature under bias	−65 °C to 110 °C
Supply voltage with respect to Vss	−.5V to 6.5V
Voltage on other pins	−.5V to (Vcc+.5)V

Table 7.2 gives stress ratings only, and functional operation at the maximums is not guaranteed. Functional operating conditions are given in section **7.3, D.C. Specifications**, and section **7.4, A.C. Specifications**.

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the 386 SX Microprocessor contains protective circuitry to resist damage from static electric discharge, always take precautions to avoid high static voltages or electric fields.

7.3 D.C. Specifications

Functional operating range: Vcc = 5V ± 10%; T_{CASE} = 0°C to 100°C

Table 7.3. 386™ SX D.C. Characteristics

Symbol	Parameter	386™SX 20 MHz, 16 MHz, 12 MHz (LP Only)		Unit	Test Condition
V_{IL}	Input LOW Voltage	−0.3	+0.8	V	
V_{IH}	Input HIGH Voltage	2.0	Vcc+0.3	V	
V_{ILC}	CLK2 Input LOW Voltage	−0.3	+0.8	V	
V_{IHC}	CLK2 Input HIGH Voltage	Vcc−0.8	Vcc+0.3	V	
V_{OL}	Output LOW Voltage I_{OL} = 4mA: A_{23}–A_1,D_{15}–D_0 I_{OL} = 5mA: BHE#,BLE#,W/R#, D/C#,M/IO#,LOCK#, ADS#,HLDA		0.45 0.45	V V	
V_{OH}	Output high voltage I_{OH} = −1mA: A_{23}–A_1,D_{15}–D_0 I_{OH} = −0.2 mA: A_{23}–A_1,D_{15}–D_0 I_{OH} = −0.9mA: BHE#,BLE#,W/R#, D/C#,M/IO#,LOCK#, ADS#,HLDA I_{OH} = −0.18 mA: BHE#,BLE#,W/R#, D/C#,M/IO#,LOCK#, ADS#,HLDA	2.4 Vcc−0.5 2.4 Vcc−0.5		V V V V	
I_{LI}	Input leakage current (for all pins except PEREQ, BUSY#, FLT# and ERROR#)		±15	μA	0V ≤ V_{IN} ≤ Vcc
I_{IH}	Input Leakage Current (PEREQ pin)		200	μA	V_{IH} = 2.4V, Note 1
I_{IL}	Input Leakage Current (BUSY#, ERROR# and FLT# Pins)		−400	μA	V_{IL} = 0.45V, Note 2
I_{LO}	Output leakage current		±15	μA	0.45V ≤ V_{OUT} ≤ Vcc
I_{CC}	Supply Current CLK2 = 4 MHz: with 20, 16, or 12 MHz 386 SX (LP) CLK2 = 24 MHz: with 12 MHz 386 SX CLK2 = 32 MHz: with 16 MHz 386 SX CLK2 = 40 MHz: with 20 MHz 386 SX		 140 245 275 305	 mA mA mA mA	 I_{CC} typ = 70 mA, Note 3 I_{CC} typ = 140 mA, Note 3 I_{CC} typ = 175 mA, Note 3 I_{CC} Typ = 20 mA, Note 3
C_{IN}	Input capacitance		10	pF	Fc = 1 MHz, Note 4
C_{OUT}	Output or I/O capacitance		12	pF	Fc = 1 MHz, Note 4
C_{CLK}	CLK2 Capacitance		20	pF	Fc = 1 MHz, Note 4

Tested at the minimum operating frequency of the part.

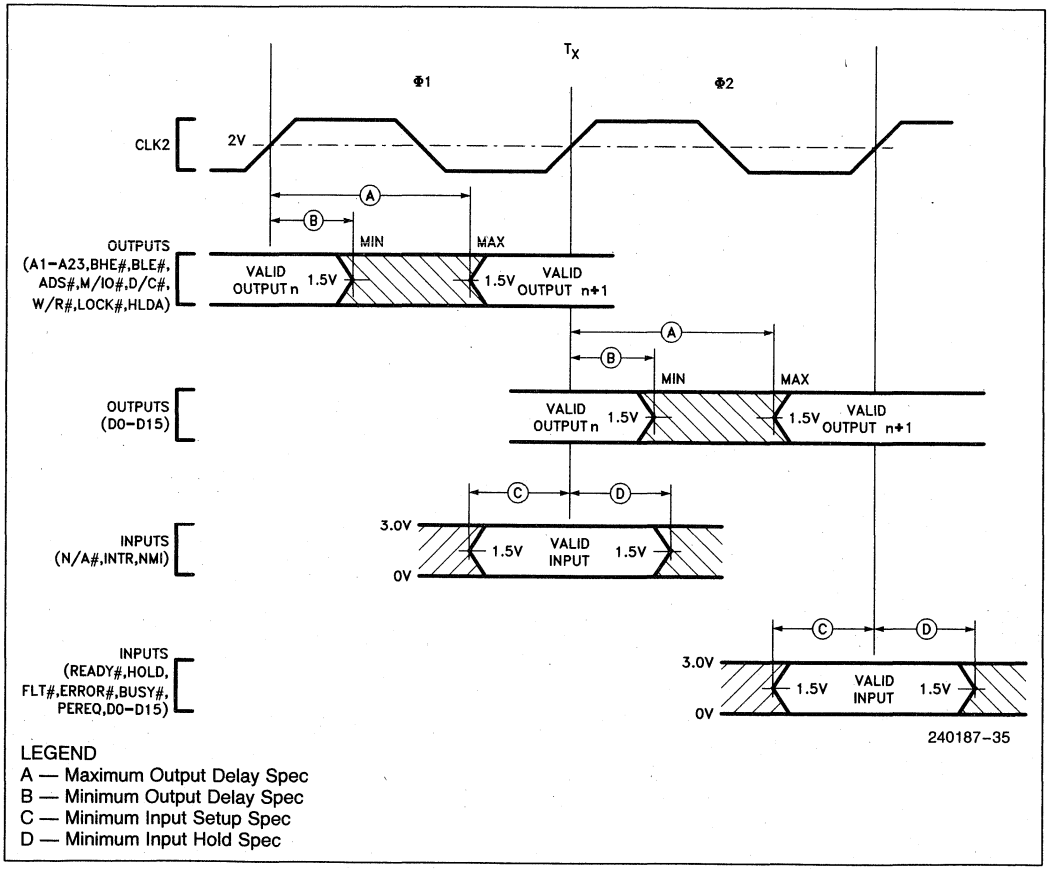
NOTES:
1. PEREQ input has an internal pull-down resistor.
2. BUSY#, FLT# and ERROR# inputs each have an internal pull-up resistor.
3. Icc max measurement at worst case load, frequency, Vcc and temperature.
4. Not 100% tested.

7.4 A.C. Specifications

The A.C. specifications given in Table 7.4 consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the CLK2 rising edge crossing the 2.0V level.

A.C. spec measurement is defined by Figure 7.1. Inputs must be driven to the voltage levels indicated by Figure 7.1 when A.C. specifications are measured. Output delays are specified with minimum and maximum limits measured as shown. The minimum delay times are hold times provided to external circuitry. Input setup and hold times are specified as minimums, defining the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct operation.

Outputs NA#, W/R#, D/C#, M/IO#, LOCK#, BHE#, BLE#, A_{23}–A_1 and HLDA only change at the beginning of phase one. D_{15}–D_0 (write cycles) only change at the beginning of phase two. The READY#, HOLD, BUSY#, ERROR#, PEREQ, FLT# and D_{15}–D_0 (read cycles) inputs are sampled at the beginning of phase one. The NA#, INTR and NMI inputs are sampled at the beginning of phase two.



LEGEND
A — Maximum Output Delay Spec
B — Minimum Output Delay Spec
C — Minimum Input Setup Spec
D — Minimum Input Hold Spec

Figure 7.1. Drive Levels and Measurement Points for A.C. Specifications

A.C. SPECIFICATONS TABLES

Functional operating range: V_{CC} = 5V ±10%; T_{CASE} = 0°C to 100°C

Table 7.4. 386™ SX A.C. Characteristics

Symbol	Parameter	20 MHz 386 SX		16 MHz 386 SX		Unit	Figure	Notes
		Min	Max	Min	Max			
	Operating Frequency	4	20	4	16	MHz		Half CLK2 Frequency
t_1	CLK2 Period	25	125	31	125	ns	7.3	
t_{2a}	CLK2 HIGH Time	8		9		ns	7.3	at 2V(3)
t_{2b}	CLK2 HIGH Time	5		5		ns	7.3	at (V_{CC}−0.8)V(3)
t_{3a}	CLK2 LOW Time	8		9		ns	7.3	at 2V(3)
t_{3b}	CLK2 LOW Time	6		7		ns	7.3	at 0.8V(3)
t_4	CLK2 Fall Time		8		8	ns	7.3	(V_{CC}−0.8)V to 0.8V(3)
t_5	CLK2 Rise Time		8		8	ns	7.3	0.8V to (V_{CC}−0.8)V(3)
t_6	A_{23}–A_1 Valid Delay	4	30	4	36	ns	7.5	C_L = 120 pF(4)
t_7	A_{23}–A_1 Float Delay	4	32	4	40	ns	7.6	(Note 1)
t_8	BHE#, BLE#, LOCK# Valid Delay	4	30	4	36	ns	7.5	C_L = 75 pF(4)
t_9	BHE#, BLE#, LOCK# Float Delay	4	32	4	40	ns	7.6	(Note 1)
t_{10a}	M/IO# D/C# Valid Delay	6	28	6	33	ns	7.5	C_L = 75 pF(4)
t_{10b}	W/R#, ADS# Valid Delay		26					
t_{11}	W/R#, M/IO#, D/C#, ADS# Float Delay	6	30	6	35	ns	7.6	(Note 1)
t_{12}	D_{15}–D_0 Write Data Valid Delay	4	38	4	40	ns	7.5	C_L = 120 pF(4)
t_{13}	D_{15}–D_0 Write Data Float Delay	4	27	4	35	ns	7.6	(Note 1)
t_{14}	HLDA Valid Delay	4	28	4	33	ns	7.5	C_L = 75 pF(4)
t_{15}	NA# Setup Time	5		5		ns	7.4	
t_{16}	NA# Hold Time	12		21		ns	7.4	
t_{19}	READY# Setup Time	12		19		ns	7.4	
t_{20}	READY# Hold Time	4		4		ns	7.4	
t_{21}	D_{15}–D_0 Read Data Setup Time	9		9		ns	7.4	
t_{22}	D_{15}–D_0 Read Data Hold Time	6		6		ns	7.4	
t_{23}	HOLD Setup Time	17		26		ns	7.4	
t_{24}	HOLD Hold Time	5		5		ns	7.4	
t_{25}	RESET Setup Time	12		13		ns	7.7	
t_{26}	RESET Hold Time	4		4		ns	7.7	

Functional operating range: V_{CC} = 5V ±10%; T_{CASE} = 0°C to 100°C

Table 7.4. 386™ SX A.C. Characteristics (Continued)

Symbol	Parameter	20 MHz 386 SX		16 MHz 386 SX		Unit	Figure	Notes
		Min	Max	Min	Max			
t_{27}	NMI, INTR Setup Time	16		16		ns	7.4	(Note 2)
t_{28}	NMI, INTR Hold Time	16		16		ns	7.4	(Note 2)
t_{29}	PEREQ, ERROR#, BUSY#, FLT# Setup Time	14		16		ns	7.4	(Note 2)
t_{30}	PEREQ, ERROR#, BUSY#, FLT# Hold Time	5		5		ns	7.4	(Note 2)

Table 7.5. Low Power (LP) 386™ SX A.C. Characteristics

Symbol	Parameter	20 MHz 386 SX		16 MHz 386 SX		12 MHz 386 SX		Unit	Figure	Notes
		Min	Max	Min	Max	Min	Max			
	Operating Frequency	2	20	2	16	2	12.5	MHz		Half CLK2 Frequency
t_1	CLK2 Period	25	250	31	250	40	250	ns	7.3	
t_{2a}	CLK2 HIGH Time	8		9		11		ns	7.3	at 2V (Note 3)
t_{2b}	CLK2 HIGH Time	5		5		7		ns	7.3	at (V_{CC} − 0.8V)(3)
t_{3a}	CLK2 LOW Time	8		9		11		ns	7.3	at 2V(3)
t_{3b}	CLK2 LOW Time	6		7		9		ns	7.3	at 0.8V(3)
t_4	CLK2 Fall Time	8		8		8		ns	7.3	(V_{CC} − 0.8V) to 0.8V(3)
t_5	CLK2 Rise Time	8		8		8		ns	7.3	0.8V to (V_{CC} − 0.8V)(3)
t_6	A_{23}–A_1 Valid Delay	4	30	4	36	4	42	ns	7.5	C_L = 120 pF(4)
t_7	A_{23}–A_1 Float Delay	4	32	4	40	4	45	ns	7.6	(Note 1)
t_8	BHE#, BLE#, LOCK# Valid Delay	4	30	4	36	4	36	ns	7.5	C_L = 75 pF
t_9	BHE#, BLE#, LOCK# Float Delay	4	32	4	40	4	40	ns	7.6	(Note 1)
t_{10}	M/IO#, D/C#, W/R#, ADS# Valid Delay	6	28	6	33	4	33	ns	7.5	C_L = 75 pF
t_{11}	M/IO#, D/C#, W/R#, ADS# Float Delay	6	30	6	35	4	35	ns	7.6	(Note 1)
t_{12}	D15–D0 Write Data Valid Delay	4	38	4	40	4	50	ns	7.5	C_L = 120 pF(4)
t_{13}	D15–D0 Write Data Float Delay	4	27	4	35	4	40	ns	7.6	(Note 1)
t_{14}	HLDA Valid Delay	4	28	6	33	4	33	ns	7.5	C_L = 75 pF(4)
t_{15}	NA# Setup Time	5		5		7		ns	7.4	
t_{16}	NA# Hold Time	12		21		21		ns	7.4	

Functional operating range: $V_{CC} = 5V \pm 10\%$; $T_{CASE} = 0°C$ to $100°C$

Table 7.5. Low Power (LP) 386™ SX A.C. Characteristics (Continued)

Symbol	Parameter	20 MHz 386 SX		16 MHz 386 SX		12 MHz 386 SX		Unit	Figure	Notes
		Min	Max	Min	Max	Min	Max			
t_{19}	READY# Setup Time	12		19		19		ns	7.4	
t_{20}	READY# Hold Time	4		4		4		ns	7.4	
t_{21}	D15–D0 Read Data Setup Time	9		9		9		ns	7.4	
t_{22}	D15–D0 Read Data Hold Time	6		6		6		ns	7.4	
t_{23}	HOLD Setup Time	17		26		26		ns	7.4	
t_{24}	HOLD Hold Time	5		5		7		ns	7.4	
t_{25}	RESET Setup Time	12		13		15		ns	7.7	
t_{26}	RESET Hold Time	4		4		6		ns	7.7	
t_{27}	NMI, INTR Setup Time	16		16		16		ns	7.4	(Note 2)
t_{28}	NMI, INTR Hold Time	16		16		16		ns	7.4	(Note 2)
t_{29}	PEREQ, ERROR#, BUSY#, FLT# Setup Time	14		16		16		ns	7.4	(Note 2)
t_{30}	PEREQ, ERROR#, BUSY#, FLT# Hold Time	5		5		5		ns	7.4	(Note 2)

NOTES:
1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.
2: These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.
3: These are not tested. They are guaranteed by design characterization.
4: Tested with C_L set at 50 pf and derated to support the indicated distributed capacitive load. See Figures 7.8 though 7.10 for the capacitive derating curve.

5

A.C. TEST LOADS

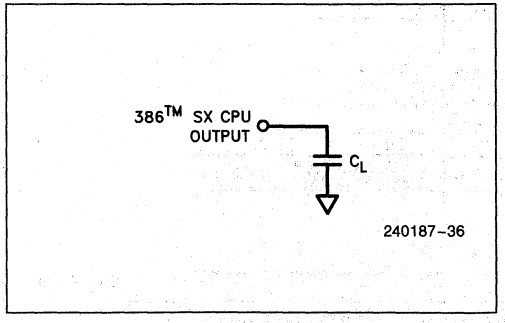


Figure 7.2. A.C. Test Loads

A.C. TIMING WAVEFORMS

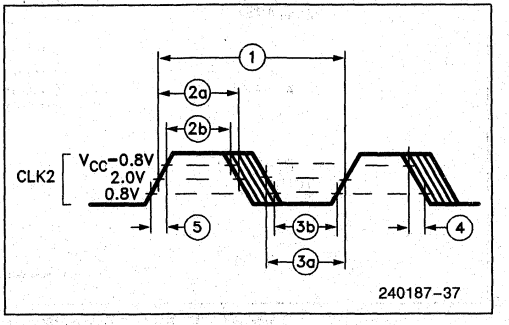


Figure 7.3. CLK2 Waveform

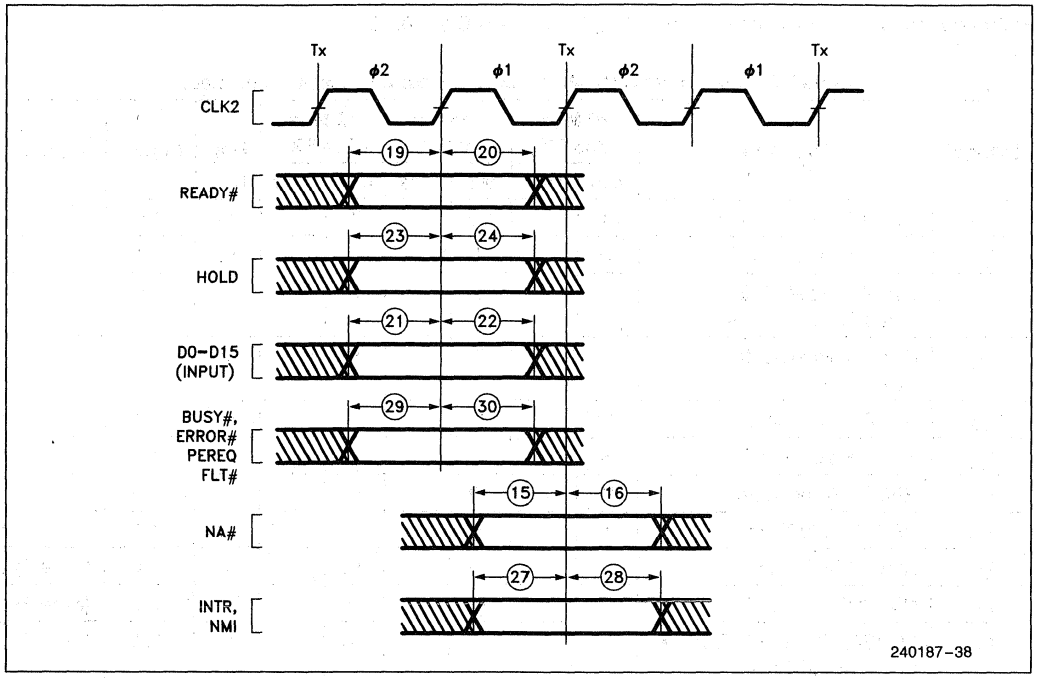


Figure 7.4. A.C. Timing Waveforms—Input Setup and Hold Timing

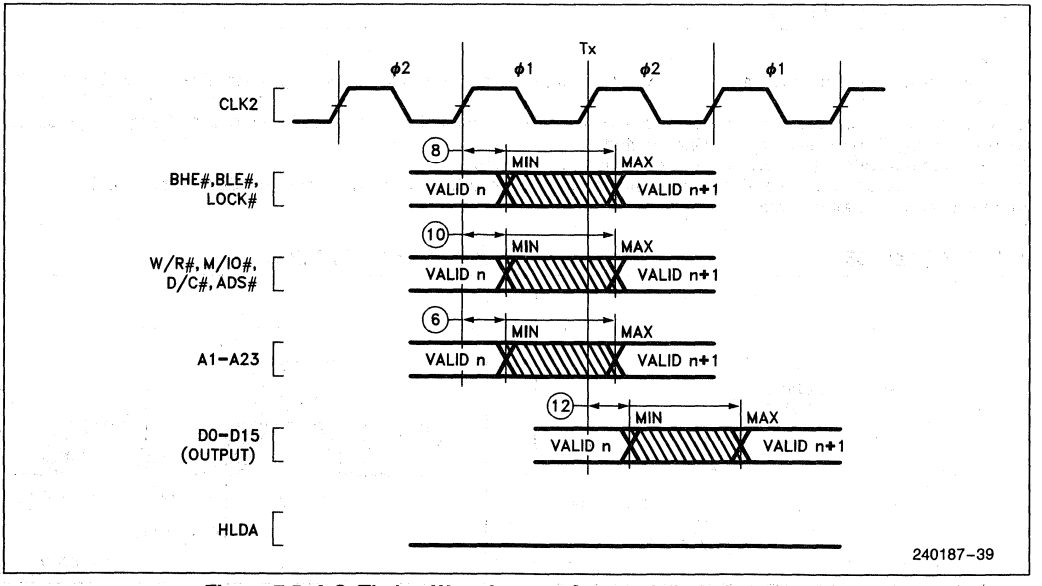


Figure 7.5. A.C. Timing Waveforms—Output Valid Delay Timing

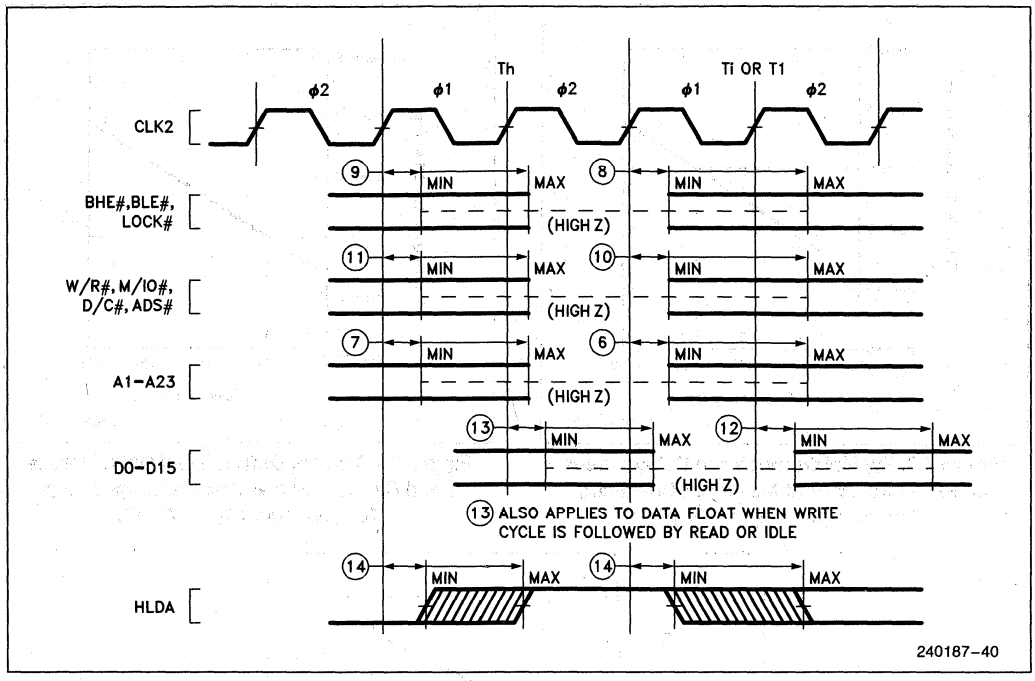


Figure 7.6. A.C. Timing Waveforms—Output Float Delay and HLDA Valid Delay Timing

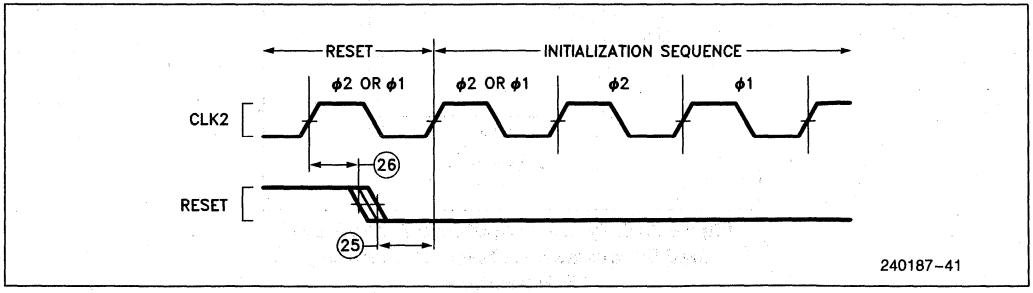


Figure 7.7. A.C. Timing Waveforms—RESET Setup and Hold Timing and Internal Phase

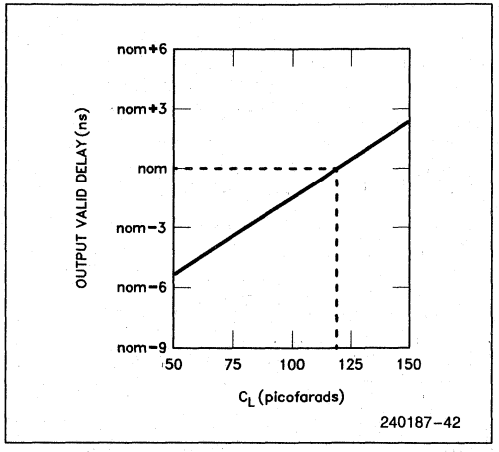


Figure 7.8. Typical Output Valid Delay versus Load Capacitance at Maximum Operating Temperature (C_L = 120 pF)

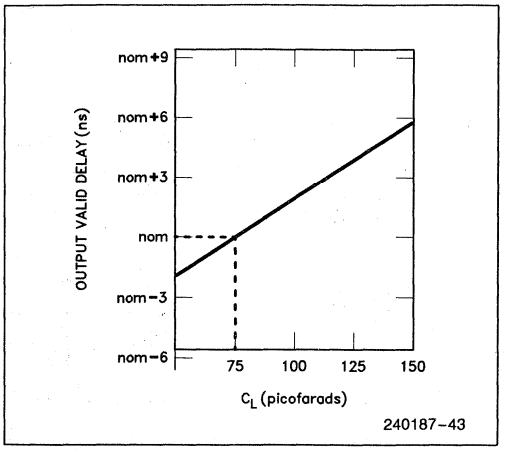


Figure 7.9. Typical Output Valid Delay versus Load Capacitance at Maximum Operating Temperature (C_L = 75 pF)

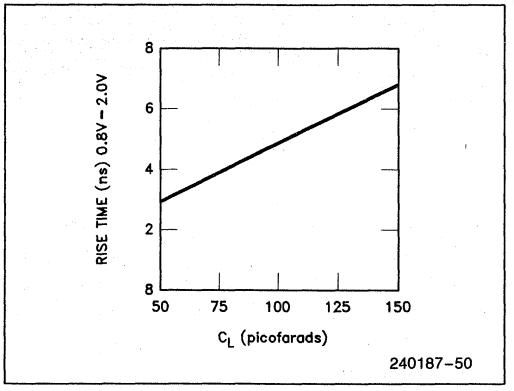


Figure 7.10. Typical Output Rise Time versus Load Capacitance at Maximum Operating Temperature

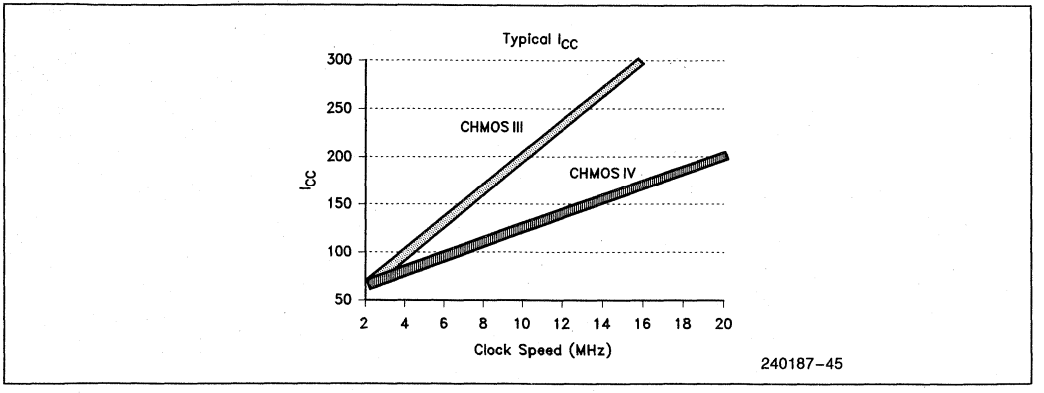


Figure 7.11. Typical I_{CC} vs Frequency

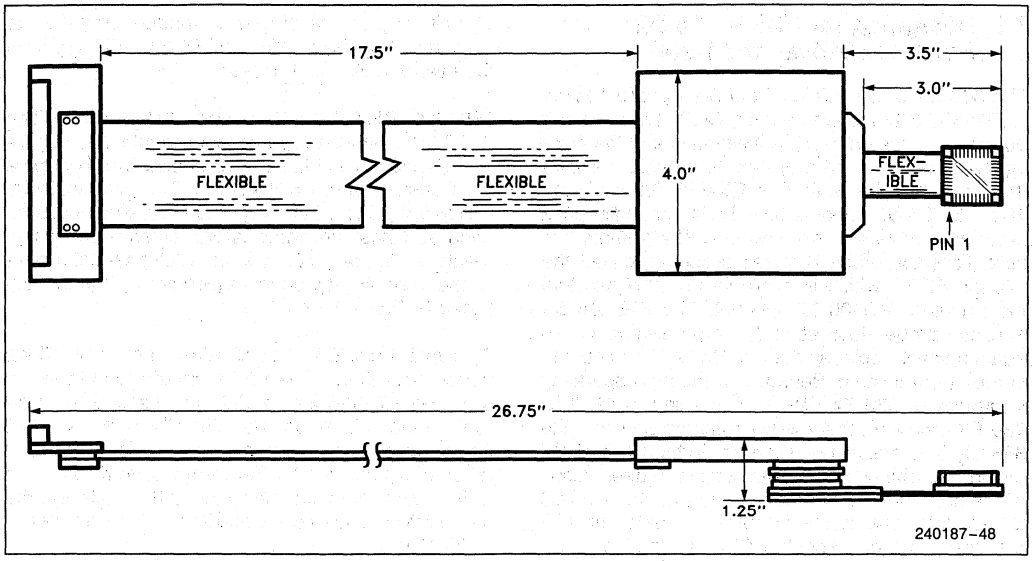


Figure 7.12. Preliminary ICE™-386 SX Emulator User Cable with PQFP Adapter

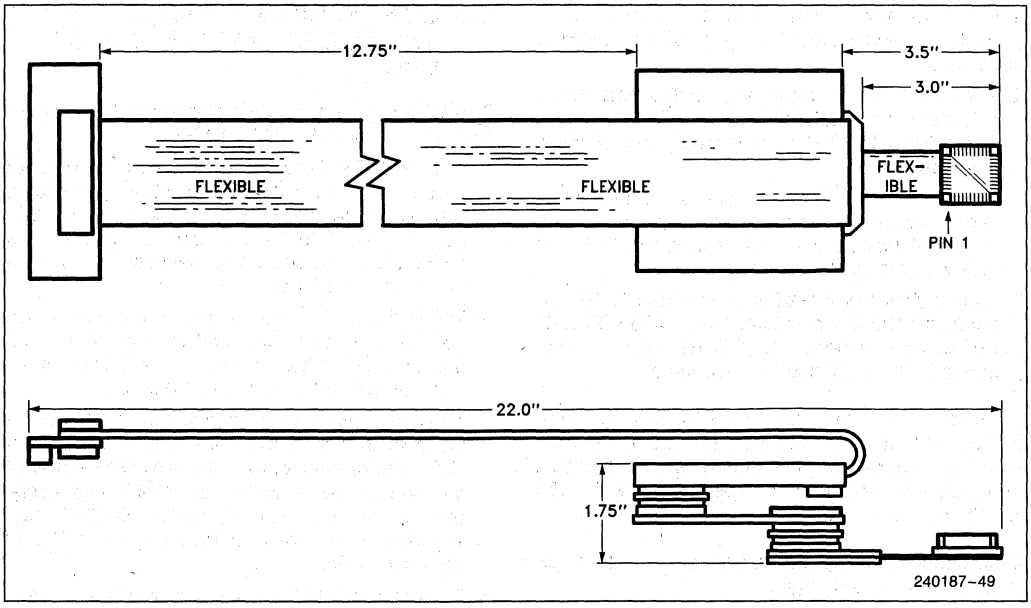


Figure 7.13. Preliminary ICE™-386 SX Emulator User Cable with OIB and PQFP Adapter

5

7.5 Designing for ICE™-386 SX Emulator (Advanced Data)

The 386 SX CPU's in-circuit emulator product is the ICE™-386 SX emulator. If your ICE™ system is not equipped to use on circuit emulation use of the emulator requires the target system to provide a socket that is compatible with the ICE-386 SX emulator. The ICE-386 SX offers a 100-pin fine pitch flat-pack probe for emulating user systems. The 100-pin fine pitch flat-pack probe requires a socket, called the 100-pin PQFP, which is available from 3M text-tool (part number 2-0100-07243-000). The ICE-386 SX emulator probe attaches to the target system via an adapter which replaces the 386 SX CPU component in the target system. Because of the high operating frequency of 386 SX CPU systems and of the ICE-386 SX emulator, there is no buffering between the 386 SX CPU emulation processor in the ICE-386 SX emulator probe and the target system. A direct result of the non-buffered interconnect is that the ICE-386 SX emulator shares the address and data bus with the user's system, and the RESET signal is intercepted by the ICE emulator hardware. In order for the ICE-386 SX emulator to be functional in the user's system without the Optional Isolation Board (OIB) the designer must be aware of the following conditions:

1. The bus controller must only enable data transceivers onto the data bus during valid read cycles of the 386 SX CPU, other local devices or other bus masters.
2. Before another bus master drives the local processor address bus, the other master must gain control of the address bus by asserting HOLD and receiving the HLDA response.
3. The emulation processor receives the RESET signal 2 or 4 CLK2 cycles later than an 386 SX CPU would, and responds to RESET later. Correct phase of the response is guaranteed.

In addition to the above considerations, the ICE-386 SX emulator processor module has several electrical and mechanical characteristics that should be taken into consideration when designing the 386 SX CPU system.

Capacitive Loading: ICE-386 SX adds up to 27 pF to each 386 SX CPU signal.

Drive Requirements: ICE-386 SX adds one FAST TTL load on the CLK2, control, address, and data lines. These loads are within the processor module and are driven by the 386 SX CPU emulation processor, which has standard drive and loading capability listed in Tables 7.3 and 7.4.

Power Requirements: For noise immunity and CMOS latch-up protection the ICE-386 SX emulator processor module is powered by the user system. The circuitry on the processor module draws up to 1.4A including the maximum 386 SX CPU I_{CC} from the user 386 SX CPU socket.

386 SX CPU Location and Orientation: The ICE-386 SX emulator processor module may require lateral clearance. Figure 7.12 shows the clearance requirements of the iMP adapter. The optional isolation board (OIB), which provides extra electrical buffering and has the same lateral clearance requirements as Figure 7.12, adds an additional 0.5 inches to the vertical clearance requirement. This is illustrated in Figure 7.13.

Optional Isolation Board (OIB) and the CLK2 speed reduction: Due to the unbuffered probe design, the ICE-386 SX emulator is susceptible to errors on the user's bus. The OIB allows the ICE-386 SX emulator to function in user systems with faults (shorted signals, etc.). After electrical verification the OIB may be removed. When the OIB is installed, the user system must have a maximum CLK2 frequency of 20 MHz.

8.0 DIFFERENCES BETWEEN THE 386 SX CPU AND THE 386 DX CPU

The following are the major differences between the 386 SX CPU and the 386 DX CPU:

1. The 386 SX CPU generates byte selects on BHE# and BLE# (like the 8086 and 80286) to distinguish the upper and lower bytes on its 16-bit data bus. The 386 DX CPU uses four byte selects, BE0#-BE3#, to distinguish between the different bytes on its 32-bit bus.
2. The 386 SX CPU has no bus sizing option. The 386 DX CPU can select between either a 32-bit bus or a 16-bit bus by use of the BS16# input. The 386 SX CPU has a 16-bit bus size.
3. The NA# pin operation in the 386 SX CPU is identical to that of the NA# pin on the 386 DX CPU with one exception: the 386 DX CPU NA# pin cannot be activated on 16-bit bus cycles (where BS16# is LOW in the 386 DX CPU case), whereas NA# can be activated on any 386 SX CPU bus cycle.
4. The contents of all 386 SX CPU registers at reset are identical to the contents of the 386 DX CPU registers at reset, except the DX register. The DX register contains a component-stepping identifier at reset, i.e.

   in 386 DX CPU, after reset: DH = 3 indicates 386 DX CPU; DL = revision number;

   in 386 SX CPU, after reset: DH = 23H indicates 386 SX CPU; DL = revision number.

5. The 386 DX CPU uses A_{31} and M/IO# as selects for the numerics coprocessor. The 386 SX CPU uses A_{23} and M/IO# as selects.
6. The 386 DX CPU prefetch unit fetches code in four-byte units. The 386 SX CPU prefetch unit reads two bytes as one unit (like the 80286). In BS16 mode, the 386 DX CPU takes two consecutive bus cycles to complete a prefetch request. If there is a data read or write request after the prefetch starts, the 386 DX CPU will fetch all four bytes before addressing the new request.
7. Both 386 DX CPU and 386 SX CPU have the same logical address space. The only difference is that the 386 DX CPU has a 32-bit physical address space and the 386 SX CPU has a 24-bit physical address space. The 386 SX CPU has a physical memory address space of up to 16 megabytes instead of the 4 gigabytes available to the 386 DX CPU. Therefore, in 386 SX CPU systems, the operating system must be aware of this physical memory limit and should allocate memory for applications programs within this limit. If a 386 DX CPU system uses only the lower 16 megabytes of physical address, then there will be no extra effort required to migrate 386 DX CPU software to the 386 SX CPU. Any application which uses more than 16 megabytes of memory can run on the 386 SX CPU if the operating system utilizes the 386 SX CPU's paging mechanism. In spite of this difference in physical address space, the 386 SX CPU and 386 DX CPU can run the same operating systems and applications within their respective physical memory constraints.
8. The 386 SX has an input called FLT# which tristates all bidirectional and output pins, including HLDA#, when asserted. It is used with ON Circuit Emulation (ONCE).

9.0 INSTRUCTION SET

This section describes the instruction set. Table 9.1 lists all instructions along with instruction encoding diagrams and clock counts. Further details of the instruction encoding are then provided in the following sections, which completely describe the encoding structure and the definition of all fields occurring within instructions.

9.1 386 SX CPU Instruction Encoding and Clock Count Summary

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in Table 9.1 below, by the processor clock period (e.g. 62.5 ns for an 386 SX Microprocessor operating at 16 MHz). The actual clock count of an 386 SX Microprocessor program will average 5% more than the calculated clock count due to instruction sequences which execute faster than they can be fetched from memory.

Instruction Clock Count Assumptions

1. The instruction has been prefetched, decoded, and is ready for execution.
2. Bus cycles do not require wait states.
3. There are no local bus HOLD requests delaying processor access to the bus.
4. No exceptions are detected during instruction execution.
5. If an effective address is calculated, it does not use two general register components. One register, scaling and displacement can be used within the clock counts shown. However, if the effective address calculation uses two general register components, add 1 clock to the clock count shown.

Instruction Clock Count Notation

1. If two clock counts are given, the smaller refers to a register operand and the larger refers to a memory operand.
2. n = number of times repeated.
3. m = number of components in the next instruction executed, where the entire displacement (if any) counts as one component, the entire immediate data (if any) counts as one component, and all other bytes of the instruction and prefix(es) each count as one component.

Misaligned or 32-Bit Operand Accesses

— If instructions accesses a misaligned 16-bit operand or 32-bit operand on even address add:
  2* clocks for read or write
  4** clocks for read and write
— If instructions accesses a 32-bit operand on odd address add:
  4* clocks for read or write
  8** clocks for read and write

Wait States

Wait states add 1 clock per wait state to instruction execution for each data access.

5

Table 9-1. Instruction Set Clock Count Summary

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
GENERAL DATA TRANSFER					
MOV = Move:					
Register to Register/Memory	1000100w mod reg r/m	2/2	2/2*	b	h
Register/Memory to Register	1000101w mod reg r/m	2/4	2/4*	b	h
Immediate to Register/Memory	1100011w mod 000 r/m immediate data	2/2	2/2*	b	h
Immediate to Register (short form)	1011 w reg immediate data	2	2		
Memory to Accumulator (short form)	1010000w full displacement	4*	4*	b	h
Accumulator to Memory (short form)	1010001w full displacement	2*	2*	b	h
Register Memory to Segment Register	10001110 mod sreg3 r/m	2/5	22/23	b	h, i, j
Segment Register to Register/Memory	10001100 mod sreg3 r/m	2/2	2/2	b	h
MOVSX = Move With Sign Extension					
Register From Register/Memory	00001111 1011111w mod reg r/m	3/6*	3/6*	b	h
MOVZX = Move With Zero Extension					
Register From Register/Memory	00001111 1011011w mod reg r/m	3/6*	3/6*	b	h
PUSH = Push:					
Register/Memory	11111111 mod 110 r/m	5/7*	7/9*	b	h
Register (short form)	01010 reg	2	4	b	h
Segment Register (ES, CS, SS or DS) (short form)	000sreg2 110	2	4	b	h
Segment Register (ES, CS, SS, DS, FS or GS)	00001111 10 sreg3 000	2	4	b	h
Immediate	011010s0 immediate data	2	4	b	h
PUSHA = Push All	01100000	18	34	b	h
POP = Pop					
Register/Memory	10001111 mod 000 r/m	5/7	7/9	b	h
Register (short form)	01011 reg	6	6	b	h
Segment Register (ES, CS, SS or DS) (short form)	000sreg 2111	7	25	b	h, i, j
Segment Register (ES, CS, SS or DS), FS or GS	00001111 10 sreg 3001	7	25	b	h, i, j
POPA = Pop All	01100001	24	40	b	h
XCHG = Exchange					
Register/Memory With Register	1000011w mod reg r/m	3/5**	3/5**	b, f	f, h
Register With Accumulator (short form)	10010 reg	3	3		
IN = Input from:		**Clk Count Virtual 8086 Mode**			
Fixed Port	1110010w port number	†26	12*	6*/26*	s/t,m
Variable Port	1110110w	†27	13*	7*/27*	s/t,m
OUT = Output to:					
Fixed Port	1110011w port number	†24	10*	4*/24*	s/t,m
Variable Port	1110111w	†25	11*	5*/25*	s/t,m
LEA = Load EA to Register	10001101 mod reg r/m		2	2	

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
SEGMENT CONTROL					
LDS = Load Pointer to DS	11000101 | mod reg r/m	7*	26*/28*	b	h, i, j
LES = Load Pointer to ES	11000100 | mod reg r/m	7*	26*/28*	b	h, i, j
LFS = Load Pointer to FS	00001111 | 10110100 | mod reg r/m	7*	29*/31*	b	h, i, j
LGS = Load Pointer to GS	00001111 | 10110101 | mod reg r/m	7*	26*/28*	b	h, i, j
LSS = Load Pointer to SS	00001111 | 10110010 | mod reg r/m	7*	26*/28*	b	h, i, j
FLAG CONTROL					
CLC = Clear Carry Flag	11111000	2	2		
CLD = Clear Direction Flag	11111100	2	2		
CLI = Clear Interrupt Enable Flag	11111010	8	8		m
CLTS = Clear Task Switched Flag	00001111 | 00000110	5	5	c	l
CMC = Complement Carry Flag	11110101	2	2		
LAHF = Load AH into Flag	10011111	2	2		
POPF = Pop Flags	10011101	5	5	b	h, n
PUSHF = Push Flags	10011100	4	4	b	h
SAHF = Store AH into Flags	10011110	3	3		
STC = Set Carry Flag	11111001	2	2		
STD = Set Direction Flag	11111101				
STI = Set Interrupt Enable Flag	11111011	8	8		m
ARITHMETIC					
ADD = Add					
Register to Register	000000dw | mod reg r/m	2	2		
Register to Memory	0000000w | mod reg r/m	7**	7**	b	h
Memory to Register	0000001w | mod reg r/m	6*	6*	b	h
Immediate to Register/Memory	100000sw | mod 000 r/m | immediate data	2/7**	2/7**	b	h
Immediate to Accumulator (short form)	0000010w | immediate data	2	2		
ADC = Add With Carry					
Register to Register	000100dw | mod reg r/m	2	2		
Register to Memory	0001000w | mod reg r/m	7**	7**	b	h
Memory to Register	0001001w | mod reg r/m	6*	6*	b	h
Immediate to Register/Memory	100000sw | mod 010 r/m | immediate data	2/7**	2/7**	b	h
Immediate to Accumulator (short form)	0001010w | immediate data	2	2		
INC = Increment					
Register/Memory	1111111w | mod 000 r/m	2/6**	2/6**	b	h
Register (short form)	01000 reg	2	2		
SUB = Subtract					
Register from Register	001010dw | mod reg r/m	2	2		

5

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
ARITHMETIC (Continued)					
Register from Memory	0010100w | mod reg r/m	7**	7**	b	h
Memory from Register	0010101w | mod reg r/m	6*	6*	b	h
Immediate from Register/Memory	100000sw | mod101 r/m | immediate data	2/7**	2/7**	b	h
Immediate from Accumulator (short form)	0010110w | immediate data	2	2		
SBB = Subtract with Borrow					
Register from Register	000110dw | mod reg r/m	2	2		
Register from Memory	0001100w | mod reg r/m	7**	7**	b	h
Memory from Register	0001101w | mod reg r/m	6*	6*	b	h
Immediate from Register/Memory	100000sw | mod011 r/m | immediate data	2/7**	2/7**	b	h
Immediate from Accumulator (short form)	0001110w | immediate data	2	2		
DEC = Decrement					
Register/Memory	1111111w | reg001 r/m	2/6	2/6	b	h
Register (short form)	01001 reg	2	2		
CMP = Compare					
Register with Register	001110dw | mod reg r/m	2	2		
Memory with Register	0011100w | mod reg r/m	5*	5*	b	h
Register with Memory	0011101w | mod reg r/m	6*	6*	b	h
Immediate with Register/Memory	100000sw | mod111 r/m | immediate data	2/5*	2/5*	b	h
Immediate with Accumulator (short form)	0011110w | immediate data	2	2		
NEG = Change Sign	1111011w | mod011 r/m	2/6*	2/6*	b	h
AAA = ASCII Adjust for Add	00110111	4	4		
AAS = ASCII Adjust for Subtract	00111111	4	4		
DAA = Decimal Adjust for Add	00100111	4	4		
DAS = Decimal Adjust for Subtract	00101111	4	4		
MUL = Multiply (unsigned)					
Accumulator with Register/Memory	1111011w | mod100 r/m				
Multiplier-Byte		12–17/15–20*	12–17/15–20*	b, d	d, h
-Word		12–25/15–28*	12–25/15–28*	b, d	d, h
-Doubleword		12–41/17–46*	12–41/17–46*	b, d	d, h
IMUL = Integer Multiply (signed)					
Accumulator with Register/Memory	1111011w | mod101 r/m				
Multiplier-Byte		12–17/15–20*	12–17/15–20*	b, d	d, h
-Word		12–25/15–28*	12–25/15–28*	b, d	d, h
-Doubleword		12–41/17–46*	12–41/17–46*	b, d	d, h
Register with Register/Memory	00001111 | 10101111 | mod reg r/m				
Multiplier-Byte		12–17/15–20*	12–17/15–20*	b, d	d, h
-Word		12–25/15–28*	12–25/15–28*	b, d	d, h
-Doubleword		12–41/17–46*	12–41/17–46*	b, d	d, h
Register/Memory with Immediate to Register	011010s1 | mod reg r/m | immediate data				
-Word		13–26	13–26/14–27	b, d	d, h
-Doubleword		13–42	13–42/16–45	b, d	d, h

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
ARITHMETIC (Continued)					
DIV = Divide (Unsigned)					
Accumulator by Register/Memory	1111011w mod110 r/m				
Divisor—Byte		14/17	14/17	b,e	e,h
—Word		22/25	22/25	b,e	e,h
—Doubleword		38/43	38/43	b,e	e,h
IDIV = Integer Divide (Signed)					
Accumulator By Register/Memory	1111011w mod111 r/m				
Divisor—Byte		19/22	19/22	b,e	e,h
—Word		27/30	27/30	b,e	e,h
—Doubleword		43/48	43/48	b,e	e,h
AAD = ASCII Adjust for Divide	11010101 00001010	19	19		
AAM = ASCII Adjust for Multiply	11010100 00001010	17	17		
CBW = Convert Byte to Word	10011000	3	3		
CWD = Convert Word to Double Word	10011001	2	2		
LOGIC					
Shift Rotate Instructions					
Not Through Carry (**ROL, ROR, SAL, SAR, SHL,** and **SHR**)					
Register/Memory by 1	1101000w mod TTT r/m	3/7**	3/7**	b	h
Register/Memory by CL	1101001w mod TTT r/m	3/7*	3/7*	b	h
Register/Memory by Immediate Count	1100000w mod TTT r/m immed 8-bit data	3/7*	3/7*	b	h
Through Carry (**RCL** and **RCR**)					
Register/Memory by 1	1101000w mod TTT r/m	9/10*	9/10*	b	h
Register/Memory by CL	1101001w mod TTT r/m	9/10*	9/10*	b	h
Register/Memory by Immediate Count	1100000w mod TTT r/m immed 8-bit data	9/10*	9/10*	b	h
	TTT Instruction: 000 ROL; 001 ROR; 010 RCL; 011 RCR; 100 SHL/SAL; 101 SHR; 111 SAR				
SHLD = Shift Left Double					
Register/Memory by Immediate	00001111 10100100 mod reg r/m immed 8-bit data	3/7**	3/7**		
Register/Memory by CL	00001111 10100101 mod reg r/m	3/7**	3/7**		
SHRD = Shift Right Double					
Register/Memory by Immediate	00001111 10101100 mod reg r/m immed 8-bit data	3/7**	3/7**		
Register/Memory by CL	00001111 10101101 mod reg r/m	3/7**	3/7**		
AND = And					
Register to Register	001000dw mod reg r/m	2	2		

5

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Clk Count Virtual 8086 Mode	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
LOGIC (Continued)						
Register to Memory	0010000w mod reg r/m		7**	7**	b	h
Memory to Register	0010001w mod reg r/m		6*	6*	b	h
Immediate to Register/Memory	1000000w mod 100 r/m immediate data		2/7*	2/7**	b	h
Immediate to Accumulator (Short Form)	0010010w immediate data		2	2		
TEST = And Function to Flags, No Result						
Register/Memory and Register	1000010w mod reg r/m		2/5*	2/5*	b	h
Immediate Data and Register/Memory	1111011w mod 000 r/m immediate data		2/5*	2/5*	b	h
Immediate Data and Accumulator (Short Form)	1010100w immediate data		2	2		
OR = Or						
Register to Register	000010dw mod reg r/m		2	2		
Register to Memory	0000100w mod reg r/m		7**	7**	b	h
Memory to Register	0000101w mod reg r/m		6*	6*	b	h
Immediate to Register/Memory	1000000w mod 001 r/m immediate data		2/7**	2/7**	b	h
Immediate to Accumulator (Short Form)	0000110w immediate data		2	2		
XOR = Exclusive Or						
Register to Register	001100dw mod reg r/m		2	2		
Register to Memory	0011000w mod reg r/m		7**	7**	b	h
Memory to Register	0011001w mod reg r/m		6*	6*	b	h
Immediate to Register/Memory	1000000w mod 110 r/m immediate data		2/7**	2/7**	b	h
Immediate to Accumulator (Short Form)	0011010w immediate data		2	2		
NOT = Invert Register/Memory	1111011w mod 010 r/m		2/6**	2/6**	b	h
STRING MANIPULATION						
CMPS = Compare Byte Word	1010011w		10*	10*	b	h
INS = Input Byte/Word from DX Port	0110110w	†29	15	9*/29**	b	s/t, h, m
LODS = Load Byte/Word to AL/AX/EAX	1010110w		5	5*	b	h
MOVS = Move Byte Word	1010010w		7	7**	b	h
OUTS = Output Byte/Word to DX Port	0110111w	†28	14	8*/28*	b	s/t, h, m
SCAS = Scan Byte Word	1010111w		7*	7*	b	h
STOS = Store Byte/Word from AL/AX/EX	1010101w		4*	4*	b	h
XLAT = Translate String	11010111		5*	5*		h
REPEATED STRING MANIPULATION Repeated by Count in CX or ECX						
REPE CMPS = Compare String (Find Non-Match)	11110011 1010011w		5 + 9n**	5 + 9n**	b	h

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	Clk Count Virtual 8086 Mode	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
REPEATED STRING MANIPULATION (Continued)						
REPNE CMPS = Compare String						
(Find Match)	11110010 1010011w		5+9n**	5+9n**	b	h
REP INS = Input String	11110010 0110110w	†	13+6n*	7+6n*/ 27+6n*	b	s/t, h, m
REP LODS = Load String	11110010 1010110w		5+6n*	5+6n*	b	h
REP MOVS = Move String	11110010 1010010w		7+4n*	7+4n**	b	h
REP OUTS = Output String	11110010 0110111w	†	12+5n*	6+5n*/ 26+5n*	b	s/t, h, m
REPE SCAS = Scan String						
(Find Non-AL/AX/EAX)	11110011 1010111w		5+8n*	5+8n*	b	h
REPNE SCAS = Scan String						
(Find AL/AX/EAX)	11110010 1010111w		5+8n*	5+8n*	b	h
REP STOS = Store String	11110010 1010101w		5+5n*	5+5n*	b	h
BIT MANIPULATION						
BSF = Scan Bit Forward	00001111 10111100 mod reg r/m		10+3n*	10+3n**	b	h
BSR = Scan Bit Reverse	00001111 10111101 mod reg r/m		10+3n*	10+3n**	b	h
BT = Test Bit						
Register/Memory, Immediate	00001111 10111010 mod 100 r/m immed 8-bit data		3/6*	3/6*	b	h
Register/Memory, Register	00001111 10100011 mod reg r/m		3/12*	3/12*	b	h
BTC = Test Bit and Complement						
Register/Memory, Immediate	00001111 10111010 mod 111 r/m immed 8-bit data		6/8*	6/8*	b	h
Register/Memory, Register	00001111 10111011 mod reg r/m		6/13*	6/13*	b	h
BTR = Test Bit and Reset						
Register/Memory, Immediate	00001111 10111010 mod 110 r/m immed 8-bit data		6/8*	6/8*	b	h
Register/Memory, Register	00001111 10110011 mod reg r/m		6/13*	6/13*	b	h
BTS = Test Bit and Set						
Register/Memory, Immediate	00001111 10111010 mod 101 r/m immed 8-bit data		6/8*	6/8*	b	h
Register/Memory, Register	00001111 10101011 mod reg r/m		6/13*	6/13*	b	h
CONTROL TRANSFER						
CALL = Call						
Direct Within Segment	11101000 full displacement		7+m*	9+m*	b	r
Register/Memory Indirect Within Segment	11111111 mod 010 r/m		7+m*/10+m*	9+m/ 12+m*	b	h, r
Direct Intersegment	10011010 unsigned full offset, selector		17+m*	42+m*	b	j,k,r

NOTE:
† Clock count shown applies if I/O permission allows I/O to the port in virtual 8086 mode. If I/O bit map denies permission exception 13 fault occurs; refer to clock counts for INT 3 instruction.

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION FORMAT	CLOCK COUNT		NOTES	
	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode	Real Address Mode or Virtual 8086 Mode	Protected Virtual Address Mode
CONTROL TRANSFER (Continued)				
Protected Mode Only (Direct Intersegment)				
Via Call Gate to Same Privilege Level		64+m		h,j,k,r
Via Call Gate to Different Privilege Level, (No Parameters)		98+m		h,j,k,r
Via Call Gate to Different Privilege Level, (x Parameters)		106+8x+m		h,j,k,r
From 286 Task to 286 TSS		285		h,j,k,r
From 286 Task to 386™ SX CPU TSS		310		h,j,k,r
From 286 Task to Virtual 8086 Task (386 SX CPU TSS)		229		h,j,k,r
From 386 SX CPU Task to 286 TSS		285		h,j,k,r
From 386 SX CPU Task to 386 SX CPU TSS		392		h,j,k,r
From 386 SX CPU Task to Virtual 8086 Task (386 SX CPU TSS)		309		h,j,k,r
Indirect Intersegment [11111111 mod011 r/m]	30+m	46+m	b	h,j,k,r
Protected Mode Only (Indirect Intersegment)				
Via Call Gate to Same Privilege Level		68+m		h,j,k,r
Via Call Gate to Different Privilege Level, (No Parameters)		102+m		h,j,k,r
Via Call Gate to Different Privilege Level, (x Parameters)		110+8x+m		h,j,k,r
From 286 Task to 286 TSS				h,j,k,r
From 286 Task to 386 SX CPU TSS				h,j,k,r
From 286 Task to Virtual 8086 Task (386 SX CPU TSS)				h,j,k,r
From 386 SX CPU Task to 286 TSS				h,j,k,r
From 386 SX CPU Task to 386 SX CPU TSS		399		h,j,k,r
From 386 SX CPU Task to Virtual 8086 Task (386 SX CPU TSS)				h,j,k,r
JMP = Unconditional Jump				
Short [11101011 8-bit displacement]	7+m	7+m		r
Direct within Segment [11101001 full displacement]	7+m	7+m		r
Register/Memory Indirect within Segment [11111111 mod100 r/m]	9+m/14+m	9+m/14+m	b	h,r
Direct Intersegment [11101010 unsigned full offset, selector]	16+m	31+m		j,k,r
Protected Mode Only (Direct Intersegment)				
Via Call Gate to Same Privilege Level		53+m		h,j,k,r
From 286 Task to 286 TSS				h,j,k,r
From 286 Task to 386 SX CPU TSS				h,j,k,r
From 286 Task to Virtual 8086 Task (386 SX CPU TSS)				h,j,k,r
From 386 SX CPU Task to 286 TSS				h,j,k,r
From 386 SX CPU Task to 386 SX CPU TSS				h,j,k,r
From 386 SX CPU Task to Virtual 8086 Task (386 SX CPU TSS)		395		h,j,k,r
Indirect Intersegment [11111111 mod101 r/m]	17+m	31+m	b	h,j,k,r
Protected Mode Only (Indirect Intersegment)				
Via Call Gate to Same Privilege Level		49+m		h,j,k,r
From 286 Task to 286 TSS				h,j,k,r
From 286 Task to 386 SX CPU TSS				h,j,k,r
From 286 Task to Virtual 8086 Task (386 SX CPU TSS)				h,j,k,r
From 386 SX CPU Task to 286 TSS				h,j,k,r
From 386 SX CPU Task to 386 SX CPU TSS		328		h,j,k,r
From 386 SX CPU Task to Virtual 8086 Task (386 SX CPU TSS)				h,j,k,r

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
CONTROL TRANSFER (Continued)					
RET = Return from CALL:					
Within Segment	11000011		12+m	b	g, h, r
Within Segment Adding Immediate to SP	11000010 | 16-bit displ		12+m	b	g, h, r
Intersegment	11001011		36+m	b	g, h, j, k, r
Intersegment Adding Immediate to SP	11001010 | 16-bit displ		36+m	b	g, h, j, k, r
Protected Mode Only (RET):					
to Different Privilege Level					
Intersegment			72		h, j, k, r
Intersegment Adding Immediate to SP			72		h, j, k, r
CONDITIONAL JUMPS					
NOTE: Times Are Jump "Taken or Not Taken"					
JO = Jump on Overflow					
8-Bit Displacement	01110000 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10000000 | full displacement	7+m or 3	7+m or 3		r
JNO = Jump on Not Overflow					
8-Bit Displacement	01110001 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10000001 | full displacement	7+m or 3	7+m or 3		r
JB/JNAE = Jump on Below/Not Above or Equal					
8-Bit Displacement	01110010 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10000010 | full displacement	7+m or 3	7+m or 3		r
JNB/JAE = Jump on Not Below/Above or Equal					
8-Bit Displacement	01110011 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10000011 | full displacement	7+m or 3	7+m or 3		r
JE/JZ = Jump on Equal/Zero					
8-Bit Displacement	01110100 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10000100 | full displacement	7+m or 3	7+m or 3		r
JNE/JNZ = Jump on Not Equal/Not Zero					
8-Bit Displacement	01110101 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10000101 | full displacement	7+m or 3	7+m or 3		r
JBE/JNA = Jump on Below or Equal/Not Above					
8-Bit Displacement	01110110 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10000110 | full displacement	7+m or 3	7+m or 3		r
JNBE/JA = Jump on Not Below or Equal/Above					
8-Bit Displacement	01110111 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10000111 | full displacement	7+m or 3	7+m or 3		r
JS = Jump on Sign					
8-Bit Displacement	01111000 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10001000 | full displacement	7+m or 3	7+m or 3		r

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
CONDITIONAL JUMPS (Continued)					
JNS = Jump on Not Sign					
8-Bit Displacement	01111001 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10001001 | full displacement	7+m or 3	7+m or 3		r
JP/JPE = Jump on Parity/Parity Even					
8-Bit Displacement	01111010 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10001010 | full displacement	7+m or 3	7+m or 3		r
JNP/JPO = Jump on Not Parity/Parity Odd					
8-Bit Displacement	01111011 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10001011 | full displacement	7+m or 3	7+m or 3		r
JL/JNGE = Jump on Less/Not Greater or Equal					
8-Bit Displacement	01111100 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10001100 | full displacement	7+m or 3	7+m or 3		r
JNL/JGE = Jump on Not Less/Greater or Equal					
8-Bit Displacement	01111101 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10001101 | full displacement	7+m or 3	7+m or 3		r
JLE/JNG = Jump on Less or Equal/Not Greater					
8-Bit Displacement	01111110 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10001110 | full displacement	7+m or 3	7+m or 3		r
JNLE/JG = Jump on Not Less or Equal/Greater					
8-Bit Displacement	01111111 | 8-bit displ	7+m or 3	7+m or 3		r
Full Displacement	00001111 | 10001111 | full displacement	7+m or 3	7+m or 3		r
JCXZ = Jump on CX Zero	11100011 | 8-bit displ	9+m or 5	9+m or 5		r
JECXZ = Jump on ECX Zero	11100011 | 8-bit displ	9+m or 5	9+m or 5		r
(Address Size Prefix Differentiates JCXZ from JECXZ)					
LOOP = Loop CX Times	11100010 | 8-bit displ	11+m	11+m		r
LOOPZ/LOOPE = Loop with Zero/Equal	11100001 | 8-bit displ	11+m	11+m		r
LOOPNZ/LOOPNE = Loop While Not Zero	11100000 | 8-bit displ	11+m	11+m		r
CONDITIONAL BYTE SET					
NOTE: Times Are Register/Memory					
SETO = Set Byte on Overflow					
To Register/Memory	00001111 | 10010000 | mod 000 r/m	4/5*	4/5*		h
SETNO = Set Byte on Not Overflow					
To Register/Memory	00001111 | 10010001 | mod 000 r/m	4/5*	4/5*		h
SETB/SETNAE = Set Byte on Below/Not Above or Equal					
To Register/Memory	00001111 | 10010010 | mod 000 r/m	4/5*	4/5*		h

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
CONDITIONAL BYTE SET (Continued)					
SETNB = Set Byte on Not Below/Above or Equal					
To Register/Memory	00001111 | 10010011 | mod000 r/m	4/5*	4/5*		h
SETE/SETZ = Set Byte on Equal/Zero					
To Register/Memory	00001111 | 10010100 | mod000 r/m	4/5*	4/5*		h
SETNE/SETNZ = Set Byte on Not Equal/Not Zero					
To Register/Memory	00001111 | 10010101 | mod000 r/m	4/5*	4/5*		h
SETBE/SETNA = Set Byte on Below or Equal/Not Above					
To Register/Memory	00001111 | 10010110 | mod000 r/m	4/5*	4/5*		h
SETNBE/SETA = Set Byte on Not Below or Equal/Above					
To Register/Memory	00001111 | 10010111 | mod000 r/m	4/5*	4/5*		h
SETS = Set Byte on Sign					
To Register/Memory	00001111 | 10011000 | mod000 r/m	4/5*	4/5*		h
SETNS = Set Byte on Not Sign					
To Register/Memory	00001111 | 10011001 | mod000 r/m	4/5*	4/5*		h
SETP/SETPE = Set Byte on Parity/Parity Even					
To Register/Memory	00001111 | 10011010 | mod000 r/m	4/5*	4/5*		h
SETNP/SETPO = Set Byte on Not Parity/Parity Odd					
To Register/Memory	00001111 | 10011011 | mod000 r/m	4/5*	4/5*		h
SETL/SETNGE = Set Byte on Less/Not Greater or Equal					
To Register/Memory	00001111 | 10011100 | mod000 r/m	4/5*	4/5*		h
SETNL/SETGE = Set Byte on Not Less/Greater or Equal					
To Register/Memory	00001111 | 01111101 | mod000 r/m	4/5*	4/5*		h
SETLE/SETNG = Set Byte on Less or Equal/Not Greater					
To Register/Memory	00001111 | 10011110 | mod000 r/m	4/5*	4/5*		h
SETNLE/SETG = Set Byte on Not Less or Equal/Greater					
To Register/Memory	00001111 | 10011111 | mod000 r/m	4/5*	4/5*		h
ENTER = Enter Procedure	11001000 | 16-bit displacement, 8-bit level				
L = 0		10	10	b	h
L = 1		14	14	b	h
L > 1		17 + 8(n − 1)	17 + 8(n − 1)	b	h
LEAVE = Leave Procedure	11001001	4	4	b	h

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
INTERRUPT INSTRUCTIONS					
INT = Interrupt:					
Type Specified	11001101 | type	37		b	
Type 3	11001100	33		b	
INTO = Interrupt 4 if Overflow Flag Set	11001110				
If OF = 1		35		b, e	
If OF = 0		3	3	b, e	
Bound = Interrupt 5 if Detect Value Out of Range	01100010 | mod reg r/m				
If Out of Range		44		b, e	e, g, h, j, k, r
If In Range		10	10	b, e	e, g, h, j, k, r
Protected Mode Only (INT)					
INT: Type Specified					
Via Interrupt or Trap Gate					
Via Interrupt or Trap Gate					
to Same Privilege Level			71		g, j, k, r
to Different Privilege Level			111		g, j, k, r
From 286 Task to 286 TSS via Task Gate			438		g, j, k, r
From 286 Task to 386™ SX CPU TSS via Task Gate			465		g, j, k, r
From 286 Task to virt 8086 md via Task Gate			382		g, j, k, r
From 386™ SX CPU Task to 286 TSS via Task Gate			440		g, j, k, r
From 386™ SX CPU Task to 386™ SX CPU TSS via Task Gate			467		g, j, k, r
From 386™ SX CPU Task to virt 8086 md via Task Gate			384		g, j, k, r
From virt 8086 md to 286 TSS via Task Gate			445		g, j, k, r
From virt 8086 md to 386™ SX CPU TSS via Task Gate			472		g, j, k, r
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate			275		
INT: TYPE 3					
Via Interrupt or Trap Gate					
to Same Privilege Level			71		g, j, k, r
Via Interrupt or Trap Gate					
to Different Privilege Level			111		g, j, k, r
From 286 Task to 286 TSS via Task Gate			382		g, j, k, r
From 286 Task to 386™ SX CPU TSS via Task Gate			409		g, j, k, r
From 286 Task to Virt 8086 md via Task Gate			326		g, j, k, r
From 386™ SX CPU Task to 286 TSS via Task Gate			384		g, j, k, r
From 386™ SX CPU Task to 386™ SX CPU TSS via Task Gate			411		g, j, k, r
From 386™ SX CPU Task to Virt 8086 md via Task Gate			328		g, j, k, r
From virt 8086 md to 286 TSS via Task Gate			389		g, j, k, r
From virt 8086 md to 386™ SX CPU TSS via Task Gate			416		g, j, k, r
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate			223		
INTO:					
Via Interrupt or Trap Grate					
to Same Privilege Level			71		g, j, k, r
Via Interrupt or Trap Gate					
to Different Privilege Level			111		g, j, k, r
From 286 Task to 286 TSS via Task Gate			384		g, j, k, r
From 286 Task to 386™ SX CPU TSS via Task Gate			411		g, j, k, r
From 286 Task to virt 8086 md via Task Gate			328		g, j, k, r
From 386™ SX CPU Task to 286 TSS via Task Gate			386 DX		g, j, k, r
From 386™ SX CPU Task to 386™ SX CPU TSS via Task Gate			413		g, j, k, r
From 386™ SX CPU Task to virt 8086 md via Task Gate			329		g, j, k, r
From virt 8086 md to 286 TSS via Task Gate			391		g, j, k, r
From virt 8086 md to 386™ SX CPU TSS via Task Gate			418		g, j, k, r
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate			223		

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
INTERRUPT INSTRUCTIONS (Continued)					
BOUND:					
Via Interrupt or Trap Gate to Same Privilege Level			71		g, j, k, r
Via Interrupt or Trap Gate to Different Privilege Level			111		g, j, k, r
From 286 Task to 286 TSS via Task Gate			358		g, j, k, r
From 286 Task to 386™ SX CPU TSS via Task Gate			388		g, j, k, r
From 268 Task to virt 8086 Mode via Task Gate			335		g, j, k, r
From 386 SX CPU Task to 286 TSS via Task Gate			368		g, j, k, r
From 386 SX CPU Task to 386 SX CPU TSS via Task Gate			398		g, j, k, r
From 386 SX CPU Task to virt 8086 Mode via Task Gate			347		g, j, k, r,
From virt 8086 Mode to 286 TSS via Task Gate			368		g, j, k, r
From virt 8086 Mode to 386 SX CPU TSS via Task Gate			398		g, j, k, r
From virt 8086 md to priv level 0 via Trap Gate or Interrupt Gate			223		
INTERRUPT RETURN					
IRET = Interrupt Return	11001111	24			g, h, j, k, r
Protected Mode Only (IRET)					
To the Same Privilege Level (within task)			42		g, h, j, k, r
To Different Privilege Level (within task)			86		g, h, j, k, r
From 286 Task to 286 TSS			285		h, j, k, r
From 286 Task to 386 SX CPU TSS			318		h, j, k, r
From 286 Task to Virtual 8086 Task			267		h, j, k, r
From 286 Task to Virtual 8086 Mode (within task)			113		
From 386 SX CPU Task to 286 TSS			324		h, j, k, r
From 386 SX CPU Task to 386 SX CPU TSS			328		h, j, k, r
From 386 SX CPU Task to Virtual 8086 Task			377		h, j, k, r
From 386 SX CPU Task to Virtual 8086 Mode (within task)			113		
PROCESSOR CONTROL					
HLT = HALT	11110100	5	5		l
MOV = Move to and From Control/Debug/Test Registers					
CR0/CR2/CR3 from register	00001111 00100010 11 eee reg	10/4/5	10/4/5		l
Register From CR0–3	00001111 00100000 11 eee reg	6	6		l
DR0–3 From Register	00001111 00100011 11 eee reg	22	22		l
DR6–7 From Register	00001111 00100011 11 eee reg	16	16		l
Register from DR6–7	00001111 00100001 11 eee reg	14	14		l
Register from DR0–3	00001111 00100001 11 eee reg	22	22		l
TR6–7 from Register	00001111 00100110 11 eee reg	12	12		l
Register from TR6–7	00001111 00100100 11 eee reg	12	12		l
NOP = No Operation	10010000	3	3		
WAIT = Wait until BUSY# pin is negated	10011011	6	6		

5

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
PROCESSOR EXTENSION INSTRUCTIONS					
Processor Extension Escape	11011TTT | mod LLL r/m TTT and LLL bits are opcode information for coprocessor.	See 387SX data sheet for clock counts			h
PREFIX BYTES					
Address Size Prefix	01100111	0	0		
LOCK = Bus Lock Prefix	11110000	0	0		m
Operand Size Prefix	01100110	0	0		
Segment Override Prefix					
CS:	00101110	0	0		
DS:	00111110	0	0		
ES:	00100110	0	0		
FS:	01100100	0	0		
GS:	01100101	0	0		
SS:	00110110	0	0		
PROTECTION CONTROL					
ARPL = Adjust Requested Privilege Level					
From Register/Memory	01100011 | mod reg r/m	N/A	20/21**	a	h
LAR = Load Access Rights					
From Register/Memory	00001111 | 00000010 | mod reg r/m	N/A	15/16*	a	g, h, j, p
LGDT = Load Global Descriptor					
Table Register	00001111 | 00000001 | mod 010 r/m	11*	11*	b, c	h, l
LIDT = Load Interrupt Descriptor					
Table Register	00001111 | 00000001 | mod 011 r/m	11*	11*	b, c	h, l
LLDT = Load Local Descriptor					
Table Register to Register/Memory	00001111 | 00000000 | mod 010 r/m	N/A	20/24*	a	g, h, j, l
LMSW = Load Machine Status Word					
From Register/Memory	00001111 | 00000001 | mod 110 r/m	10/13	10/13*	b, c	h, l
LSL = Load Segment Limit					
From Register/Memory	00001111 | 00000011 | mod reg r/m				
Byte-Granular Limit		N/A	20/21*	a	g, h, j, p
Page-Granular Limit		N/A	25/26*	a	g, h, j, p
LTR = Load Task Register					
From Register/Memory	00001111 | 00000000 | mod 001 r/m	N/A	23/27*	a	g, h, j, l
SGDT = Store Global Descriptor					
Table Register	00001111 | 00000001 | mod 000 r/m	9*	9*	b, c	h
SIDT = Store Interrupt Descriptor					
Table Register	00001111 | 00000001 | mod 001 r/m	9*	9*	b, c	h
SLDT = Store Local Descriptor Table Register					
To Register/Memory	00001111 | 00000000 | mod 000 r/m	N/A	2/2*	a	h

Table 9-1. Instruction Set Clock Count Summary (Continued)

INSTRUCTION	FORMAT	CLOCK COUNT: Real Address Mode or Virtual 8086 Mode	CLOCK COUNT: Protected Virtual Address Mode	NOTES: Real Address Mode or Virtual 8086 Mode	NOTES: Protected Virtual Address Mode
PROTECTION CONTROL (Continued)					
SMSW = **Store Machine Status Word**	00001111 | 00000001 | mod 1 0 0 r/m	2/2*	2/2*	b, c	h, l
STR = **Store Task Register** To Register/Memory	00001111 | 00000000 | mod 0 0 1 r/m	N/A	2/2*	a	h
VERR = **Verify Read Access** Register/Memory	00001111 | 00000000 | mod 1 0 0 r/m	N/A	10/11*	a	g, h, j, p
VERW = **Verify Write Access**	00001111 | 00000000 | mod 1 0 1 r/m	N/A	15/16*	a	g, h, j, p

INSTRUCTION NOTES FOR TABLE 9-1

Notes a through c apply to Real Address Mode only:

a. This is a Protected Mode instruction. Attempted execution in Real Mode will result in exception 6 (invalid opcode).

b. Exception 13 fault (general protection) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum CS, DS, ES, FS or GS limit, FFFFH. Exception 12 fault (stack segment limit violation or not present) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.

c. This instruction may be executed in Real Mode. In Real Mode, its purpose is primarily to initialize the CPU for Protected Mode.

Notes d through g apply to Real Address Mode and Protected Virtual Address Mode:

d. The 386 SX CPU uses an early-out multiply algorithm. The actual number of clocks depends on the position of the most significant bit in the operand (multiplier).

Clock counts given are minimum to maximum. To calculate actual clocks use the following formula:

Actual Clock = if m < > 0 then max ([$\log_2$ |m|], 3) + b clocks:
if m = 0 then 3+b clocks

In this formula, m is the multiplier, and

b = 9 for register to register,
b = 12 for memory to register,
b = 10 for register with immediate to register,
b = 11 for memory with immediate to register.

e. An exception may occur, depending on the value of the operand.

f. LOCK# is automatically asserted, regardless of the presence or absence of the LOCK# prefix.

g. LOCK# is asserted during descriptor table accesses.

Notes h through r apply to Protected Virtual Address Mode only:

h. Exception 13 fault (general protection violation) will occur if the memory operand in CS, DS, ES, FS or GS cannot be used due to either a segment limit violation or access rights violation. If a stack limit is violated, an exception 12 (stack segment limit violation or not present) occurs.

i. For segment load operations, the CPL, RPL, and DPL must agree with the privilege rules to avoid an exception 13 fault (general protection violation). The segment's descriptor must indicate "present" or exception 11 (CS, DS, ES, FS, GS not present). If the SS register is loaded and a stack segment not present is detected, an exception 12 (stack segment limit violation or not present) occurs.

j. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert LOCK# to maintain descriptor integrity in multiprocessor systems.

k. JMP, CALL, INT, RET and IRET instructions referring to another code segment will cause an exception 13 (general protection violation) if an applicable privilege rule is violated.

l. An exception 13 fault occurs if CPL is greater than 0 (0 is the most privileged level).

m. An exception 13 fault occurs if CPL is greater than IOPL.

n. The IF bit of the flag register is not updated if CPL is greater than IOPL. The IOPL and VM fields of the flag register are updated only if CPL = 0.

o. The PE bit of the MSW (CR0) cannot be reset by this instruction. Use MOV into CR0 if desiring to reset the PE bit.

p. Any violation of privilege rules as applied to the selector operand does not cause a protection exception; rather, the zero flag is cleared.

q. If the coprocessor's memory operand violates a segment limit or segment access rights, an exception 13 fault (general protection exception) will occur before the ESC instruction is executed. An exception 12 fault (stack segment limit violation or not present) will occur if the stack limit is violated by the operand's starting address.

r. The destination of a JMP, CALL, INT, RET or IRET must be in the defined limit of a code segment or an exception 13 fault (general protection violation) will occur.

s/t. The instruction will execute in s clocks if CPL ≤ IOPL. If CPL > IOPL, the instruction will take t clocks.

9.2 INSTRUCTION ENCODING

9.2.1 Overview

All instruction encodings are subsets of the general instruction format shown in Figure 8-1. Instructions consist of one or two primary opcode bytes, possibly an address specifier consisting of the "mod r/m" byte and "scaled index" byte, a displacement if required, and an immediate data field if required.

Within the primary opcode or opcodes, smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary opcode byte(s). This byte, the mod r/m byte, specifies the address mode to be used. Certain encodings of the mod r/m byte indicate a second addressing byte, the scale-index-base byte, follows the mod r/m byte to fully specify the addressing mode.

Addressing modes can include a displacement immediately following the mod r/m byte, or scaled index byte. If a displacement is present, the possible sizes are 8, 16 or 32 bits.

If the instruction specifies an immediate operand, the immediate operand follows any displacement bytes. The immediate operand, if specified, is always the last field of the instruction.

Figure 9-1 illustrates several of the fields that can appear in an instruction, such as the mod field and the r/m field, but the Figure does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the opcode bytes themselves. Table 9-2 is a complete list of all fields appearing in the instruction set. Further ahead, following Table 9-2, are detailed tables for each field.

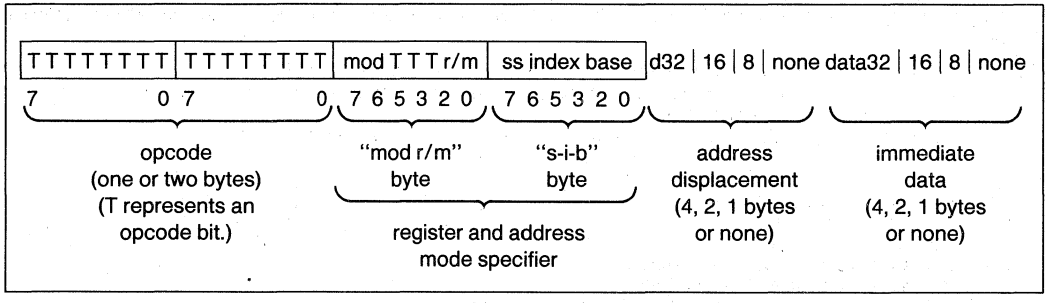


Figure 9-1. General Instruction Format

Table 9-2. Fields within Instructions

Field Name	Description	Number of Bits
w	Specifies if Data is Byte or Full Size (Full Size is either 16 or 32 Bits	1
d	Specifies Direction of Data Operation	1
s	Specifies if an Immediate Data Field Must be Sign-Extended	1
reg	General Register Specifier	3
mod r/m	Address Mode Specifier (Effective Address can be a General Register)	2 for mod; 3 for r/m
ss	Scale Factor for Scaled Index Address Mode	2
index	General Register to be used as Index Register	3
base	General Register to be used as Base Register	3
sreg2	Segment Register Specifier for CS, SS, DS, ES	2
sreg3	Segment Register Specifier for CS, SS, DS, ES, FS, GS	3
tttn	For Conditional Instructions, Specifies a Condition Asserted or a Condition Negated	4

Note: Table 9-1 shows encoding of individual instructions.

9.2.2 32-Bit Extensions of the Instruction Set

With the 386 SX CPU, the 8086/80186/80286 instruction set is extended in two orthogonal directions: 32-bit forms of all 16-bit instructions are added to support the 32-bit data types, and 32-bit addressing modes are made available for all instructions referencing memory. This orthogonal instruction set extension is accomplished having a Default (D) bit in the code segment descriptor, and by having 2 prefixes to the instruction set.

Whether the instruction defaults to operations of 16 bits or 32 bits depends on the setting of the D bit in the code segment descriptor, which gives the default length (either 32 bits or 16 bits) for both operands and effective addresses when executing that code segment. In the Real Address Mode or Virtual 8086 Mode, no code segment descriptors are used, but a D value of 0 is assumed internally by the 386 SX CPU when operating in those modes (for 16-bit default sizes compatible with the 8086/80186/80286).

Two prefixes, the Operand Size Prefix and the Effective Address Size Prefix, allow overriding individually the Default selection of operand size and effective address size. These prefixes may precede any opcode bytes and affect only the instruction they precede. If necessary, one or both of the prefixes may be placed before the opcode bytes. The presence of the Operand Size Prefix and the Effective Address Prefix will toggle the operand size or the effective address size, respectively, to the value "opposite" from the Default setting. For example, if the default operand size is for 32-bit data operations, then presence of the Operand Size Prefix toggles the instruction to 16-bit data operation. As another example, if the default effective address size is 16 bits, presence of the Effective Address Size prefix toggles the instruction to use 32-bit effective address computations.

These 32-bit extensions are available in all modes, including the Real Address Mode or the Virtual 8086 Mode. In these modes the default is always 16 bits, so prefixes are needed to specify 32-bit operands or addresses. For instructions with more than one prefix, the order of prefixes is unimportant.

Unless specified otherwise, instructions with 8-bit and 16-bit operands do not affect the contents of the high-order bits of the extended registers.

9.2.3 Encoding of Instruction Fields

Within the instruction are several fields indicating register selection, addressing mode and so on. The exact encodings of these fields are defined immediately ahead.

9.2.3.1 ENCODING OF OPERAND LENGTH (w) FIELD

For any given instruction performing a data operation, the instruction is executing as a 32-bit operation or a 16-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or the full operation size, as shown in the table below.

w Field	Operand Size During 16-Bit Data Operations	Operand Size During 32-Bit Data Operations
0	8 Bits	8 Bits
1	16 Bits	32 Bits

9.2.3.2 ENCODING OF THE GENERAL REGISTER (reg) FIELD

The general register is specified by the reg field, which may appear in the primary opcode bytes, or as the reg field of the "mod r/m" byte, or as the r/m field of the "mod r/m" byte.

Encoding of reg Field When w Field is not Present in Instruction

reg Field	Register Selected During 16-Bit Data Operations	Register Selected During 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
101	SI	ESI
101	DI	EDI

Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field During 16-Bit Data Operations:		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Register Specified by reg Field During 32-Bit Data Operations		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

9.2.3.3 ENCODING OF THE SEGMENT REGISTER (sreg) FIELD

The sreg field in certain instructions is a 2-bit field allowing one of the four 80286 segment registers to be specified. The sreg field in other instructions is a 3-bit field, allowing the 386 SX CPU FS and GS segment registers to be specified.

2-Bit sreg2 Field

2-Bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

3-Bit sreg3 Field

3-Bit sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	do not use
111	do not use

9.2.3.4 ENCODING OF ADDRESS MODE

Except for special instructions, such as PUSH or POP, where the addressing mode is pre-determined, the addressing mode for the current instruction is specified by addressing bytes following the primary opcode. The primary addressing byte is the "mod r/m" byte, and a second byte of addressing information, the "s-i-b" (scale-index-base) byte, can be specified.

The s-i-b byte (scale-index-base byte) is specified when using 32-bit addressing mode and the "mod r/m" byte has r/m = 100 and mod = 00, 01 or 10. When the sib byte is present, the 32-bit addressing mode is a function of the mod, ss, index, and base fields.

The primary addressing byte, the "mod r/m" byte, also contains three bits (shown as TTT in Figure 8-1) sometimes used as an extension of the primary opcode. The three bits, however, may also be used as a register field (reg).

When calculating an effective address, either 16-bit addressing or 32-bit addressing is used. 16-bit addressing uses 16-bit address components to calculate the effective address while 32-bit addressing uses 32-bit address components to calculate the effective address. When 16-bit addressing is used, the "mod r/m" byte is interpreted as a 16-bit addressing mode specifier. When 32-bit addressing is used, the "mod r/m" byte is interpreted as a 32-bit addressing mode specifier.

Tables on the following three pages define all encodings of all 16-bit addressing modes and 32-bit addressing modes.

Encoding of 16-bit Address Mode with "mod r/m" Byte

mod r/m	Effective Address
00 000	DS:[BX + SI]
00 001	DS:[BX + DI]
00 010	SS:[BP + SI]
00 011	SS:[BP + DI]
00 100	DS:[SI]
00 101	DS:[DI]
00 110	DS:d16
00 111	DS:[BX]
01 000	DS:[BX + SI + d8]
01 001	DS:[BX + DI + d8]
01 010	SS:[BP + SI + d8]
01 011	SS:[BP + DI + d8]
01 100	DS:[SI + d8]
01 101	DS:[DI + d8]
01 110	SS:[BP + d8]
01 111	DS:[BX + d8]

mod r/m	Effective Address
10 000	DS:[BX + SI + d16]
10 001	DS:[BX + DI + d16]
10 010	SS:[BP + SI + d16]
10 011	SS:[BP + DI + d16]
10 100	DS:[SI + d16]
10 101	DS:[DI + d16]
10 110	SS:[BP + d16]
10 111	DS:[BX + d16]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by r/m During 16-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by r/m During 32-Bit Data Operations		
mod r/m	Function of w Field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode with "mod r/m" byte (no "s-i-b" byte present):

mod r/m	Effective Address
00 000	DS:[EAX]
00 001	DS:[ECX]
00 010	DS:[EDX]
00 011	DS:[EBX]
00 100	s-i-b is present
00 101	DS:d32
00 110	DS:[ESI]
00 111	DS:[EDI]
01 000	DS:[EAX + d8]
01 001	DS:[ECX + d8]
01 010	DS:[EDX + d8]
01 011	DS:[EBX + d8]
01 100	s-i-b is present
01 101	SS:[EBP + d8]
01 110	DS:[ESI + d8]
01 111	DS:[EDI + d8]

mod r/m	Effective Address
10 000	DS:[EAX + d32]
10 001	DS:[ECX + d32]
10 010	DS:[EDX + d32]
10 011	DS:[EBX + d32]
10 100	s-i-b is present
10 101	SS:[EBP + d32]
10 110	DS:[ESI + d32]
10 111	DS:[EDI + d32]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by reg or r/m during 16-Bit Data Operations:		
mod r/m	function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Register Specified by reg or r/m during 32-Bit Data Operations:		
mod r/m	function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Encoding of 32-bit Address Mode ("mod r/m" byte and "s-i-b" byte present):

mod base	Effective Address
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

NOTE:
Mod field in "mod r/m" byte; ss, index, base fields in "s-i-b" byte.

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8

index	Index Register
000	EAX
001	ECX
010	EDX
011	EBX
100	no index reg**
101	EBP
110	ESI
111	EDI

****IMPORTANT NOTE:**
When index field is 100, indicating "no index register," then ss field MUST equal 00. If index is 100 and ss does not equal 00, the effective address is undefined.

5

9.2.3.5 ENCODING OF OPERATION DIRECTION (d) FIELD

In many two-operand instructions the d field is present to indicate which operand is considered the source and which is the destination.

d	Direction of Operation
0	Register/Memory <- - Register "reg" Field Indicates Source Operand; "mod r/m" or "mod ss index base" Indicates Destination Operand
1	Register <- - Register/Memory "reg" Field Indicates Destination Operand; "mod r/m" or "mod ss index base" Indicates Source Operand

9.2.3.6 ENCODING OF SIGN-EXTEND (s) FIELD

The s field occurs primarily to instructions with immediate data fields. The s field has an effect only if the size of the immediate data is 8 bits and is being placed in a 16-bit or 32-bit destination.

s	Effect on Immediate Data8	Effect on Immediate Data 16|32
0	None	None
1	Sign-Extend Data8 to Fill 16-Bit or 32-Bit Destination	None

9.2.3.7 ENCODING OF CONDITIONAL TEST (tttn) FIELD

For the conditional instructions (conditional jumps and set on condition), tttn is encoded with n indicating to use the condition (n = 0) or its negation (n = 1), and tt giving the condition to test.

Mnemonic	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less or Equal/Greater Than	1111

9.2.3.8 ENCODING OF CONTROL OR DEBUG OR TEST REGISTER (eee) FIELD

For the loading and storing of the Control, Debug and Test registers.

When Interpreted as Control Register Field

eee Code	Reg Name
000	CR0
010	CR2
011	CR3
Do not use any other encoding	

When Interpreted as Debug Register Field

eee Code	Reg Name
000	DR0
001	DR1
010	DR2
011	DR3
110	DR6
111	DR7
Do not use any other encoding	

When Interpreted as Test Register Field

eee Code	Reg Name
110	TR6
111	TR7
Do not use any other encoding	

DATA SHEET REVISION REVIEW

The following list represents key differences between this and the -001 version of the 386™ SX microprocessor data sheet. Please review this summary carefully.

The section significantly revised since version -002 is:

Section 1.0 — Figure 1.1 was modified to also give pin names. Table 1.1 was modified to list pin names in alphabetical order.

The sections significantly revised since version -003 are:

Section 7.3 — Table 7.3 modified to show new I_{CC} values at 16 MHz and 20 MHz.

Section 7.4 — Add 20 MHz A.C. Specifications in Table 7.5. Modified capacitive derating information in Tables 7.8 through 7.11. Modified typical I_{CC} vs. frequency in Table 7.12.

The sections significantly revised since version -004 are:

Section 5.4 — Added Section on FLT#.

Section 7.3 — Table 7.3 modified to show the FLT# function and T_{CASE} at 100°C.

Section 7.4 — Changed T_{14} to 4 ns. Deleted Figure 7.10.

The section significantly revised since version -005 are:

Section 1.0 — Pin Description was modified to add ONCE description in Symbol FLT# section.

Section 7.3 — Table 7.3 modified to show Low Power 386 SX I_{CC} Max value and typical value at different frequency.

Section 7.4 — Merge 20 MHz and 16 MHz standard 386 SX A.C. specification in Table 7.4.

Add Low Power 386 SX 20 MHz, 16 MHz and 12 MHz A.C. Specification as Table 7.5.

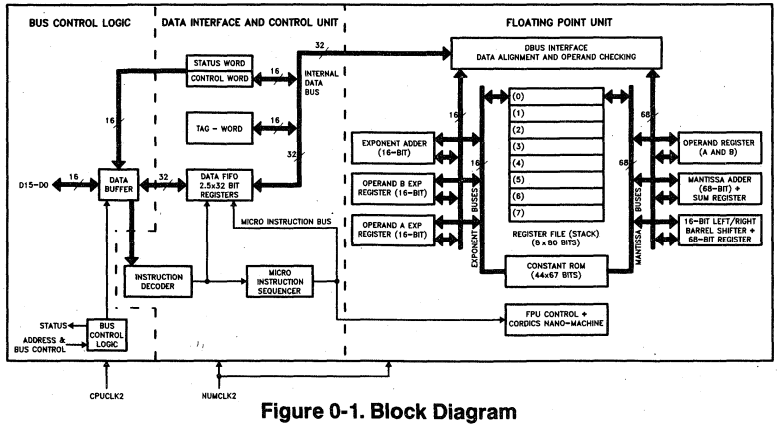
387™ SX MATH COPROCESSOR

- **Interfaces with 386™ SX Microprocessor**
- **Expands 386 SX CPU Data Types to Include 32-, 64-, 80-Bit Floating Point, 32-, 64-Bit Integers and 18-Digit BCD Operands**
- **High Performance 80-Bit Internal Architecture**
- **Two to Three Times 8087/80287 Performance at Equivalent Clock Speed**
- **Implements ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic**
- **Fully compatible with the 387™ Math Coprocessor. Implements all 387 NPX architectural enhancements over 8087 and 80287.**
- **Upward Object-Code Compatible from 8087 and 80287**
- **Directly Extends 386 SX CPU Instruction Set to Trigonometric, Logarithmic, Exponential, and Arithmetic Instructions for All Data Types**
- **Full-Range Transcendental Operations for SINE, COSINE, TANGENT, ARCTANGENT, and LOGARITHM.**
- **Operates Independently of Real, Protected, and Virtual-8086 Modes of the 386 SX Microprocessor**
- **Eight 80-Bit Numeric Registers, Usable as Individually Addressable General Registers or as a Register Stack**
- **Available in a 68-pin PLCC Package (see Packaging Specs: Order #231369)**

The Intel 387™ SX Math CoProcessor is an extension to the Intel 386™ microprocessor architecture. The combination of the 387 SX with the 386™ SX Microprocessor dramatically increases the processing speed of computer application software which utilizes mathematical operations. This makes an ideal computer workstation platform for applications such as financial modeling and spreadsheets, CAD/CAM, or graphics.

The 387 SX Math CoProcessor adds over seventy mnemonics to the 386 SX Microprocessor instruction set. Specific 387 SX math operations include logarithmic, arithmetic, exponentional, and triginometric functions. The 387 SX supports integer, extended integer, floating point and BCD data formats, and fully conforms to the ANSI/IEEE floating point standard.

The 387 SX Math CoProcessor is object code compatible with the 387™ DX and upward object code compatible from the 80287 and 8087 Math CoProcessors. The 387 SX is manufactured with Intel's CHMOS III technology and packaged in a 68-pin PLCC package. A low power consumption option allows use in laptop or portable applications.



240225-1

Figure 0-1. Block Diagram

September 1990
Order Number: 240225-005

TABLE OF CONTENTS

CONTENTS

PAGE

1.0 Functional Description 5-965
2.0 Programming Interface 5-966
2.1 Data Types 5-966
2.2 Numeric Operands 5-966
2.3 Register Set 5-968
2.3.1 Data Registers 5-968
2.3.2 Tag Word 5-968
2.3.3 Status Word 5-968
2.3.4 Control Word 5-972
2.3.5 Instruction and Data Pointers 5-973
2.4 Interrupt Description 5-975
2.5 Exception Handling 5-975
2.6 Initialization 5-975
2.7 8087 and 80287 Compatibility 5-976
2.7.1 General Differences 5-976
2.7.2 Exceptions 5-977
3.0 Hardware Interface 5-977
3.1 Signal Description 5-977
3.1.1 386™ SX CPU Clock 2 (CPUCLK2) 5-978
3.1.2 387™ SX NPX Clock 2 (NUMCLK2) 5-978
3.1.3 Clocking Mode (CKM) 5-978
3.1.4 System Reset (RESETIN) 5-978
3.1.5 Processor Extension Request (PEREQ) 5-979
3.1.6 Busy Status (BUSY#) 5-979
3.1.7 Error Status (ERROR#) 5-979
3.1.8 Data Pins (D15–D0) 5-979
3.1.9 Write/Read Bus Cycle (W/R#) 5-979
3.1.10 Address Strobe (ADS#) 5-979
3.1.11 Bus Ready Input (READY#) 5-979
3.1.12 Ready Output (READYO#) 5-979
3.1.13 Status Enable (STEN) 5-979
3.1.14 NPX Select 1 (NPS1#) 5-979
3.1.15 NPX Select 2 (NPS2) 5-980
3.1.16 Command (CMD0#) 5-980
3.1.17 System Power (V_{CC}) 5-980
3.1.18 System Ground (V_{SS}) 5-980
3.2 System Configuration 5-980
3.3 Processor Architecture 5-981
3.3.1 Bus Control Logic 5-981
3.3.2 Data Interface and Control Unit 5-981
3.3.3 Floating-Point Unit 5-981
3.4 Bus Cycles 5-981
3.4.1 387™ SX NPX Addressing 5-981
3.4.2 CPU/NPX Synchronization 5-982
3.4.3 Synchronous or Asynchronous Modes 5-982
3.4.4 Automatic Bus Cycle Termination 5-982
4.0 Bus Operation 5-982
4.1 Nonpipelined Busy Cycles 5-983
4.1.1 Write Cycle 5-983
4.1.2 Read Cycle 5-984
4.2 Pipelined Bus Cycles 5-984
4.3 Bus Cycles of Mixed Type 5-985
4.4 BUSY# and PEREQ Timing Relationship 5-986
5.0 Package Thermal Specifications 5-987

5

CONTENTS

PAGE

6.0 Electrical Data 5-989
6.1 Absolute Maximum Ratings 5-989
6.2 D.C. Characteristics 5-989
6.3 A.C. Characteristics 5-989
7.0 387™ SX NPX Extensions to the CPU's Instruction Set 5-995
Appendix A—Compatibility between the 80287 and the 8087 NPX 5-1000

FIGURES

Figure 0-1. Block Diagram 5-962
Figure 1-1. 386™ SX Microprocessor and 387™ SX Math Coprocessor Register Set 5-965
Figure 2-1. Tag Word 5-968
Figure 2-2. Status Word 5-969
Figure 2-3. Control Word 5-972
Figure 2-4. Instruction and Data Pointer Image in Memory, 32-bit Protected-Mode Format ... 5-973
Figure 2-5. Instruction and Data Pointer Image in Memory, 16-bit Protected-Mode Format ... 5-974
Figure 2-6. Instruction and Data Pointer Image in Memory, 32-bit Real-Mode Format 5-974
Figure 2-7. Instruction and Data Pointer Image in Memory, 16-bit Real-Mode Format 5-974
Figure 3-1. Asynchronous Operation 5-977
Figure 3-2. 386™ SX Microprocessor and 387™ SX Math Coprocessor System Configuration 5-980
Figure 4-1. Bus State Diagram 5-983
Figure 4-2. Nonpipelined Read and Write Cycles 5-984
Figure 4-3. Fastest Transitions to and from Pipelined Cycles 5-985
Figure 4-4. Pipelined Cycles with Wait States 5-986
Figure 4-5. STEN, BUSY# and PEREQ Timing Relationships 5-987
Figure 5-1. PLCC Pin Configuration 5-988
Figure 6-1a. Typical Output Valid Delay vs. Load Capacitance at Max Operating Temperature 5-991
Figure 6-1b. Typical Output Slew Times vs. Load Capacitance at Max Operating Temperature 5-991
Figure 6-1c. Maximum I_{CC} vs. Frequency 5-992
Figure 6-2. CPUCLK2/NUMCLK2 Waveform and Measurement Points for Input/Output 5-992
Figure 6-3. Output Signals 5-993
Figure 6-4. Input and I/O Signals 5-993
Figure 6-5. RESET Signal 5-994
Figure 6-6. Float from STEN 5-994
Figure 6-7. Other Parameters 5-995

TABLES

Table 2-1. 387™ SX NPX Data Type Representation in Memory 5-967
Table 2-2. Condition Code Interpretation 5-970
Table 2-3. Condition Code Interpretation after FPREM and FPREM1 Instructions 5-971
Table 2-4. Condition Code Resulting from Comparison 5-971
Table 2-5. Condition Code Defining Operand Class 5-971
Table 2-6. CPU Interrupt Vectors Reserved for NPX 5-975
Table 2-7. Exceptions 5-976
Table 3-1. Pin Summary 5-978
Table 3-2. Output Pin Status during Reset 5-979
Table 3-3. Bus Cycles Definition 5-981
Table 5-1. Thermal Resistances (°C/Watt) θ_{jc} and θ_{ja} 5-987
Table 5.2. Maximum T_A at Various Airflows 5-988
Table 5-3. Pin Cross-Reference 5-988
Table 6-1. D.C. Specifications 5-989
Table 6-2a. Combinations of Bus Interface and Execution Speeds 5-989
Table 6-2b. Timing Requirements of Execution Unit 5-990
Table 6-2c. Timing Requirements of Bus Interface Unit 5-990
Table 6-3. Other Parameters 5-994
Table 7-1. Instruction Formats 5-996

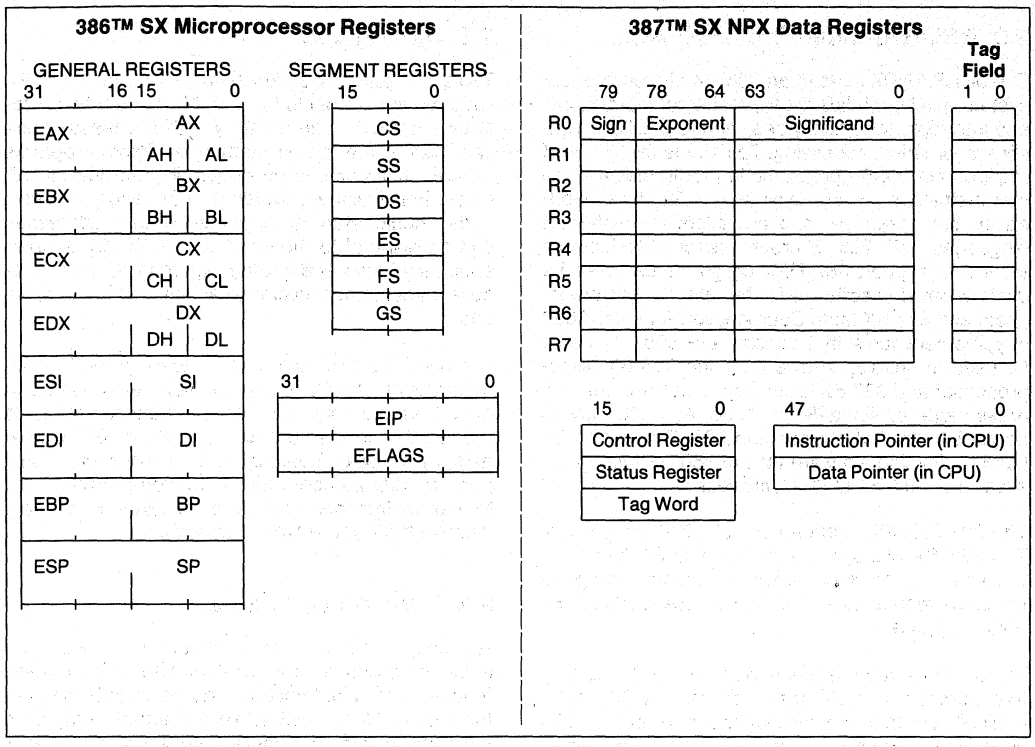


Figure 1-1. 386™ SX Microprocessor and 387™ SX Math Coprocessor Register Set

5

1.0 FUNCTIONAL DESCRIPTION

The 387™ SX Math Coprocessor Extension (NPX) provides arithmetic instructions for a variety of numeric data types. It also executes numerous built-in transcendental functions (e.g. tangent, sine, cosine, and log functions). The 387 SX NPX effectively extends the register and instruction set of its CPU for existing data types and adds several new data types as well. Figure 1-1 shows the model of registers visible to 386™ SX Microprocessor and 387 SX Math Coprocessor applications programs. Essentially, the 387 SX Math Coprocessor can be treated as an additional resource or an extension to the 386 SX Microprocessor. The 386 SX Microprocessor together with a 387 SX NPX can be used as a single unified system, the 386 SX Microprocessor and 387 SX Math Coprocessor.

The 387 SX Numerics Coprocessor Extension works the same whether the CPU is executing in real-address mode, protected mode, or virtual-8086 mode. All references to memory for numerics data or status information are performed by the CPU, and therefore obey the memory-management and protection rules of the CPU mode currently in effect. The 387 SX Numerics Coprocessor Extension merely operates on instructions and values passed to it by the CPU and therefore is not sensitive to the processing mode of the CPU.

In real-address mode and virtual-8086 mode, the 386 SX Microprocessor and 387 SX Math Coprocessor is completely upward compatible with software for the 8086/8087 and 80286/80287 real-address mode systems.

In protected mode, the 386 SX Microprocessor and 387 SX Math Coprocessor is completely upward compatible with software for the 80286/80287 protected mode system.

In all modes, the 386 SX Microprocessor and 387 SX Math Coprocessor is completely compatible with software for the 386™ Microprocessor/387™ Math Coprocessor system.

The only differences of operation that may appear when 8086/8087 programs are ported to the protected-mode 386 SX Microprocessor and 387 SX Math Coprocessor system (*not* using virtual-8086 mode) is in the format of operands for the administrative instructions FLDENV, FSTENV, FRSTOR, and FSAVE. These instruction are normally used only by exception handlers and operating systems, not by applications programs.

2.0 PROGRAMMING INTERFACE

The 387 SX NPX adds to an 386 SX Microprocessor system additional data types, registers, instructions, and interrupts specifically designed to facilitate high-speed numerics processing. To use the 387 SX NPX requires no special programming tools, because all new instructions and data types are directly supported by the assembler and compilers for high-level languages. All 386 Microprocessor development tools that support 387 NPX programs can also be used to develop software for the 386 SX Microprocessor and 387 SX Math Coprocessor. All 8086/8088 development tools that support the 8087 can also be used to develop software for the 386 SX Microprocessor and 387 SX Math Coprocessor in real-address mode or virtual-8086 mode. All 80286 development tools that support the 80287 can also be used to develop software for the 386 SX Microprocessor and 387 SX Math Coprocessor.

The 387 SX NPX supports all 387 NPX instructions. The 386 SX Microprocessor and 387 SX Math Coprocessor supports all the same programs and gives the same results as an 386 Microprocessor and 387 Math Coprocessor.

All communication between the CPU and the NPX is transparent to applications software. The CPU automatically controls the NPX whenever a numerics instruction is executed. All physical memory and virtual memory of the CPU are available for storage of the instructions and operands of programs that use the NPX. All memory addressing modes, including use of displacement, base register, index register, and scaling, are available for addressing numerics operands.

Section 7 at the end of this data sheet lists by class the instructions that the 387 SX NPX adds to the instruction set of an 386 SX Microprocessor system.

2.1 Data Types

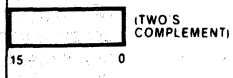
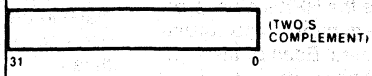
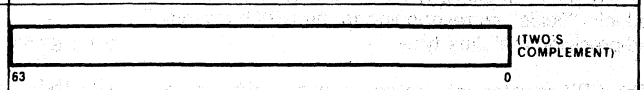
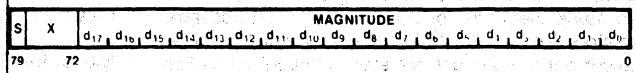
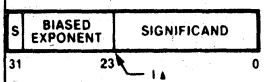
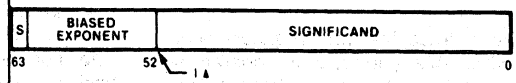
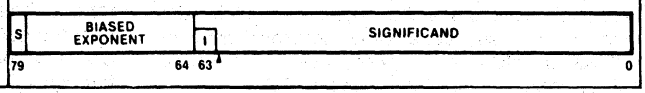
Table 2-1 lists the seven data types that the NPX supports and presents the format for each type. Operands are stored in memory with the least significant digit at the lowest memory address. Programs retrieve these values by generating the lowest address. For maximum system performance, all operands should start at physical-memory addresses that correspond to the word size of the CPU; operands may begin at any other addresses, but will require extra memory cycles to access the entire operand.

Internally, the NPX holds all numbers in the extended-precision real format. Instructions that load operands from memory automatically convert operands represented in memory as 16-, 32-, or 64-bit integers, 32- or 64-bit floating-point numbers, or 18-digit packed BCD numbers into extended-precision real format. Instructions that store operands in memory perform the inverse type conversion.

2.2 Numeric Operands

A typical NPX instruction accepts one or two operands and produces one (or sometimes two) results. In two-operand instructions, one operand is the contents of an NPX register, while the other may be a memory location. The operands of some instructions are predefined; for example, FSQRT always takes the square root of the number in the top stack element.

Table 2-1. 387™ SX NPX Data Type Representation in Memory

Data Formats	Range	Precision	Most Significant Byte = HIGHEST ADDRESSED BYTE
Word Integer	$\pm 10^4$	16 Bits	
Short Integer	$\pm 10^9$	32 Bits	
Long Integer	$\pm 10^{18}$	64 Bits	
Packed BCD	$\pm 10^{18}$	18 Digits	
Single Precision	$\pm 10^{\pm 38}$	24 Bits	
Double Precision	$\pm 10^{\pm 308}$	53 Bits	
Extended Precision	$\pm 10^{\pm 4932}$	64 Bits	

240225-2

NOTES:
(1) S = Sign bit (0 = positive, 1 = negative)
(2) d_n = Decimal digit (two per byte)
(3) X = Bits have no significance; NPX ignores when loading, zeros when storing
(4) ▲ = Position of implicit binary point
(5) I = Integer bit of significand; stored in temporary real, implicit in single and double precision
(6) Exponent Bias (normalized values):
Single: 127 (7FH)
Double: 1023 (3FFH)
Extended REal: 16383 (3FFFH)
(7) Packed BCD: $(-1)^S (D_{17}..D_0)$
(8) Real: $(-1)^S (2^{E-BIAS}) (F_0 F_1...)$

2.3 Register Set

Figure 1-1 shows the 387 SX NPX register set. When an NPX is present in a system, programmers may use these registers in addition to the registers normally available on the CPU.

2.3.1 DATA REGISTERS

387 SX NPX computations use the NPX's data registers. These eight 80-bit registers provide the equivalent capacity of 20 32-bit registers. Each of the eight data registers in the NPX is 80 bits wide and is divided into "fields" corresponding to the NPX's extended-precision real data type.

The NPX register set can be accessed either as a stack, with instructions operating on the top one or two stack elements, or as individually addressable registers. The TOP field in the status word identifies the current top-of-stack register. A "push" operation decrements TOP by one and loads a value into the new top register. A "pop" operation stores the value from the current top register and then increments TOP by one. The NPX register stack grows "down" toward lower-addressed registers.

Instructions may address the data registers either implicitly or explicitly. Many instructions operate on the register at the TOP of the stack. These instructions implicitly address the register at which TOP points. Other instructions allow the programmer to explicitly specify which register to use. This explicit register addressing is also relative to TOP.

2.3.2 TAG WORD

The tag word marks the content of each numeric data register, as Figure 2-1 shows. Each two-bit tag represents one of the eight data registers. The principal function of the tag word is to optimize the NPX's performance and stack handling by making it possible to distinguish between empty and nonempty register locations. It also enables exception handlers to identify special values (e.g. NaNs or denormals) in the contents of a stack location without the need to perform complex decoding of the actual data.

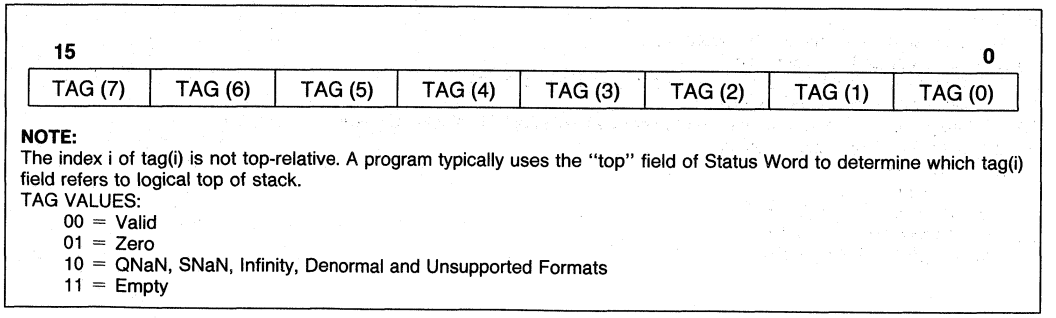
2.3.3 STATUS WORD

The 16-bit status word (in the status register) shown in Figure 2-2 reflects the overall state of the NPX. It may be read and inspected by programs.

Bit 15, the B-bit (busy bit) is included for 8087 compatibility only. It always has the same value as the ES bit (bit 7 of the status word); it does **not** indicate the status of the BUSY# output of NPX.

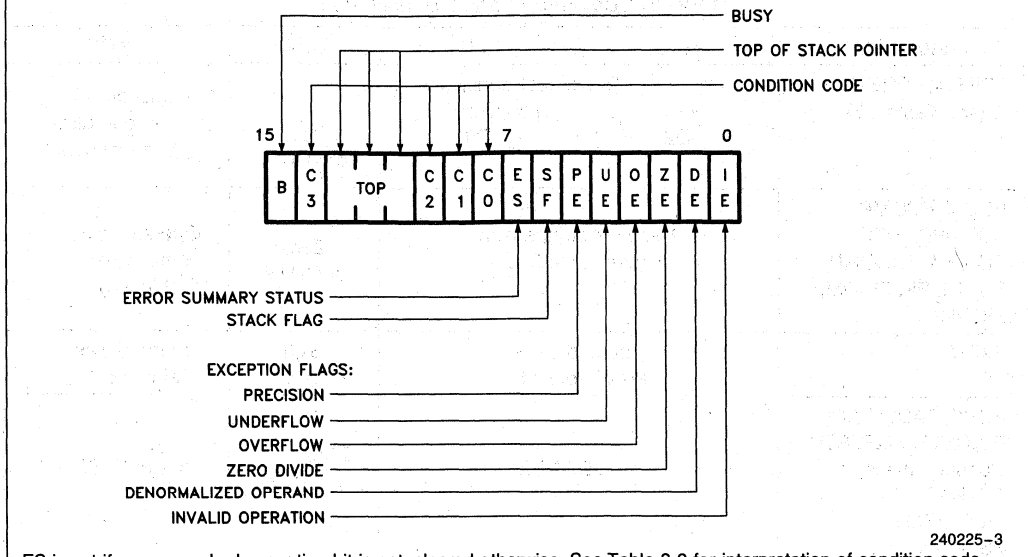
Bits 13–11 (TOP) point to the NPX register that is the current top-of-stack.

The four numeric condition code bits (C_3–C_0) are similar to the flags in a CPU; instructions that perform arithmetic operations update these bits to reflect the outcome. The effects of these instructions on the condition code are summarized in Tables 2-2 through 2-5.



NOTE:
The index i of tag(i) is not top-relative. A program typically uses the "top" field of Status Word to determine which tag(i) field refers to logical top of stack.
TAG VALUES:
00 = Valid
01 = Zero
10 = QNaN, SNaN, Infinity, Denormal and Unsupported Formats
11 = Empty

Figure 2-1. Tag Word



ES is set if any unmasked exception bit is set; cleared otherwise. See Table 2-2 for interpretation of condition code.
TOP values:
000 = Register 0 is Top of Stack
001 = Register 1 is Top of Stack
.
.
.
111 = Register 7 is Top of Stack
For definitions of exceptions, refer to the section entitled "Exception Handling"

Figure 2-2. Status Word

Bit 7 is the error summary (ES) status bit. This bit is set if any unmasked exception bit is set; it is clear otherwise. If this bit is set, the ERROR# signal is asserted.

Bit 6 is the stack flag (SF). This bit is used to distinguish invalid operations due to stack overflow or underflow from other kinds of invalid operations. When SF is set, bit 9 (C_1) distinguishes between stack overflow ($C_1 = 1$) and underflow ($C_1 = 0$).

Figure 2-2 shows the six exception flags in bits 5–0 of the status word. Bits 5–0 are set to indicate that the NPX has detected an exception while executing an instruction. A later section entitled "Exception Handling" explains how they are set and used.

Note that when a new value is loaded into the status word by the FLDENV or FRSTOR instruction, the value of ES (bit 7) and its reflection in the B-bit (bit 15) are not derived from the values loaded from memory but rather are dependent upon the values of the exception flags (bits 5–0) in the status word and their corresponding masks in the control word. If ES is set in such a case, the ERROR# output of the NPX is activated immediately.

Table 2-2. Condition Code Interpretation

Instruction	C0 (S)	C3 (Z)	C1 (A)	C2 (C)
FPREM, FPREM1 (see Table 2.3)	Three least significant bits of quotient: Q2	Q0	Q1 or O/U#	Reduction: 0 = complete, 1 = incomplete
FCOM, FCOMP, FCOMPP, FTST, FUCOM, FUCOMP, FUCOMPP, FICOM, FICOMP	Result of comparison (see Table 2.4)		Zero or O/U#	Operand is not comparable (Table 2.4)
FXAM	Operand class (see Table 2.5)		Sign or O/U#	Operand class (Table 2.5)
FCHS, FABS, FXCH, FINCSTP, FDECSTP, Constant loads, FXTRACT, FLD, FILD, FBLD, FSTP (ext real)	UNDEFINED		Zero or O/U#	UNDEFINED
FIST, FBSTP, FRNDINT, FST, FSTP, FADD, FMUL, FDIV, FDIVR, FSUB, FSUBR, FSCALE, FSQRT, FPATAN, F2XM1, FYL2X, FYL2XP1	UNDEFINED		Roundup or O/U#	UNDEFINED
FPTAN, FSIN FCOS, FSINCOS	UNDEFINED		Roundup or O/U#, undefined if C2 = 1	Reduction: 0 = complete, 1 = incomplete
FLDENV, FRSTOR	Each bit loaded from memory			
FLDCW, FSTENV, FSTCW, FSTSW, FCLEX, FINIT, FSAVE	UNDEFINED			

O/U#	When both IE and SF bits of status word are set, indicating a stack exception, this bit distinguishes between stack overflow (C1 = 1) and underflow (C1 = 0).
Reduction	If FPREM or FPREM1 produces a remainder that is less than the modulus, reduction is complete. When reduction is incomplete the value at the top of the stack is a partial remainder, which can be used as input to further reduction. For FPTAN, FSIN, FCOS,and FSINCOS, the reudction bit is set if the oeprand at the top of the stack is too large. In this case the original operand remains at the top of the stack.
Roundup	When the PE bit of the status word is set, this bit indicates whether the last rounding in the instruction was upward.
UNDEFINED	Do not rely on finding any specific value in these bits.

Table 2-3. Condition Code Interpretation after FPREM and FPREM1 Instructions

Condition Code				Interpretation after FPREM and FPREM1	
C2	C3	C1	C0		
1	X	X	X	Incomplete Reduction: further interation required for complete reduction	
	Q1	Q0	Q2	Q MOD8	Complete Reduction: C0, C3, C1 contain three least significant bits of quotient
0	0	0	0	0	
	0	1	0	1	
	1	0	0	2	
	1	1	0	3	
	0	0	1	4	
	0	1	1	5	
	1	0	1	6	
	1	1	1	7	

Table 2-4. Condition Code Resulting from Comparison

Order	C3	C2	C0
TOP > Operand	0	0	0
TOP < Operand	0	0	1
TOP = Operand	1	0	0
Unordered	1	1	1

Table 2.5. Condition Code Defining Operand Class

C3	C2	C1	C0	Value at TOP
0	0	0	0	+ Unsupported
0	0	0	1	+ NaN
0	0	1	0	− Unsupported
0	0	1	1	− NaN
0	1	0	0	+ Normal
0	1	0	1	+ Infinity
0	1	1	0	− Normal
0	1	1	1	− Infinity
1	0	0	0	+ 0
1	0	0	1	+ Empty
1	0	1	0	− 0
1	0	1	1	− Empty
1	1	0	0	+ Denormal
1	1	1	0	− Denormal

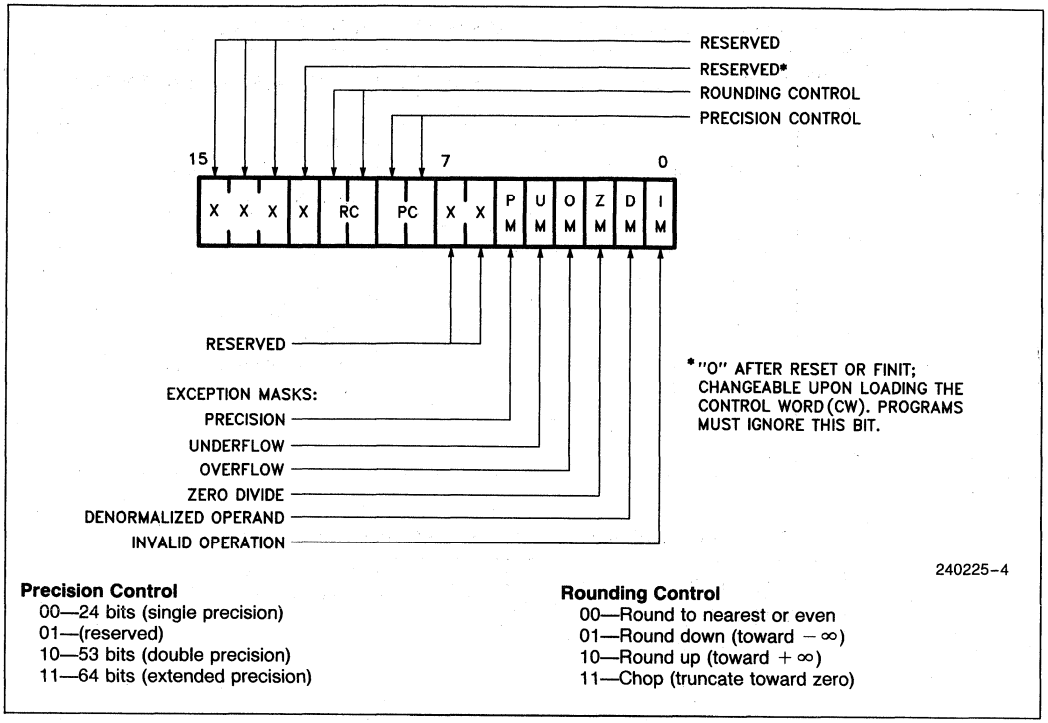


Figure 2-3. Control Word

2.3.4 CONTROL WORD

The NPX provides several processing options that are selected by loading a control word from memory into the control register. Figure 2-3 shows the format and encoding of fields in the control word.

The low-order byte of this control word configures exception masking. Bits 5–0 of the control word contain individual masks for each of the six exceptions that the NPX recognizes.

The high-order byte of the control word configures the NPX operating mode, including precision, rounding, and infinity control.

- The "infinity control bit" (bit 12) is not meaningful to the 387 SX NPX, and programs must ignore its value. To maintain compatibility with the 8087 and 80287, this bit can be programmed; however, regardless of its value, the 387 SX NPX always treats infinity in the affine sense ($-\infty < +\infty$). This bit is initialized to zero both after a hardware reset and after the FINIT instruction.
- The rounding control (RC) bits (bits 11–10) provide for directed rounding and true chop, as well as the unbiased round to nearest even mode specified in the IEEE standard. Rounding control affects only those instructions that perform rounding at the end of the operation (and thus can generate a precision exception); namely, FST, FSTP, FIST, all arithmetic instructions (except FPREM, FPREM1, FXTRACT, FABS, and FCHS), and all transcendental instructions.
- The precision control (PC) bits (bits 9–8) can be used to set the NPX internal operating precision of the significand at less than the default of 64 bits (extended precision). This can be useful in providing compatibility with early generation arithmetic processors of smaller precision. PC affects only the instructions ADD, SUB, DIV, MUL, and SQRT. For all other instructions, either the precision is determined by the opcode or extended precision is used.

2.3.5 INSTRUCTION AND DATA POINTERS

Because the NPX operates in parallel with the CPU, any exceptions detected by the NPX may be reported after the CPU has executed the ESC instruction which caused it. To allow identification of the failing numeric instruction, the 386 SX Microprocessor and 387 SX Math Coprocessor contains registers that aid in diagnosis. These registers supply the address of the failing instruction and the address of its numeric memory operand (if appropriate).

The instruction and data pointers are provided for user-written exception handlers. These registers are actually located in the CPU, but appear to be located in the NPX because they are accessed by the ESC instructions FLDENV, FSTENV, FSAVE, and FRSTOR. Whenever the CPU executes a new ESC instruction, it saves the address of the instruction (including any prefixes that may be present), the address of the operand (if present), and the opcode.

The instruction and data pointers appear in one of four formats depending on the operating mode of the CPU (protected mode or real-address mode) and depending on the operand-size attribute in effect (32-bit operand or 16-bit operand). (See Figures 2-4, 2-5, 2-6, and 2-7.) The ESC instructions FLDENV, FSTENV, FSAVE, and FRSTOR are used to transfer these values between the registers and memory. Note that the value of the data pointer is *undefined* if the prior ESC instruction did not have a memory operand.

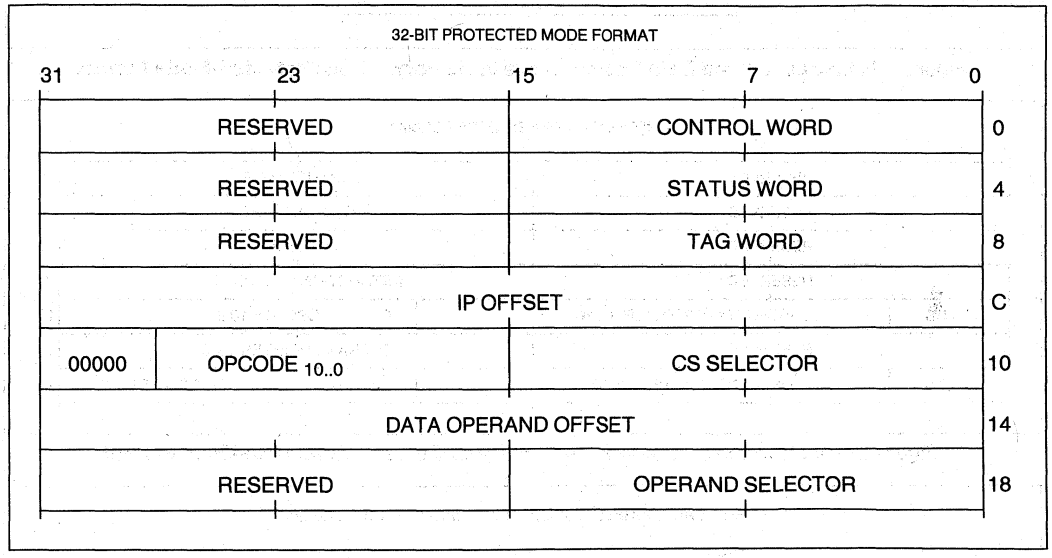


Figure 2-4.Instruction and Data Pointer Image in Memory, 32-bit Protected-Mode Format

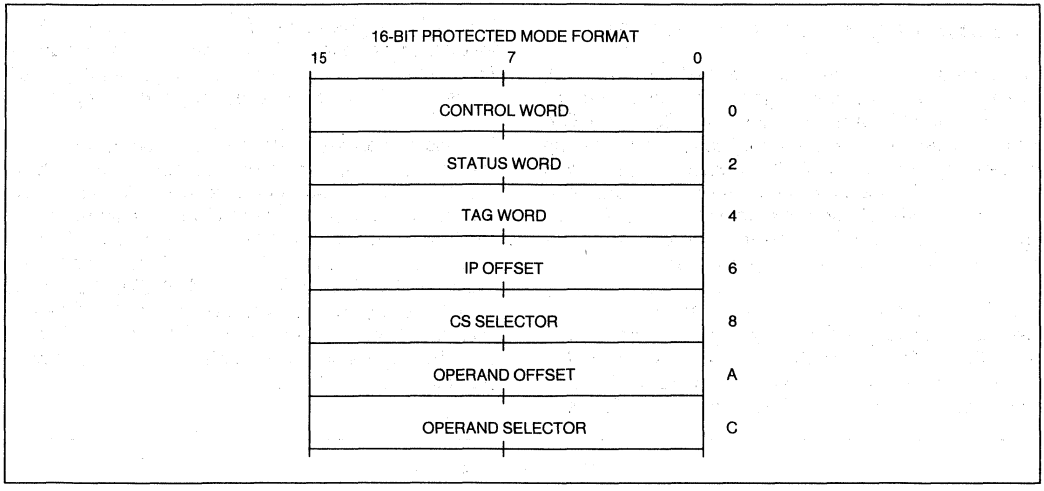


Figure 2-5. Instruction and Data Pointer Image in Memory, 16-bit Protected-Mode Format

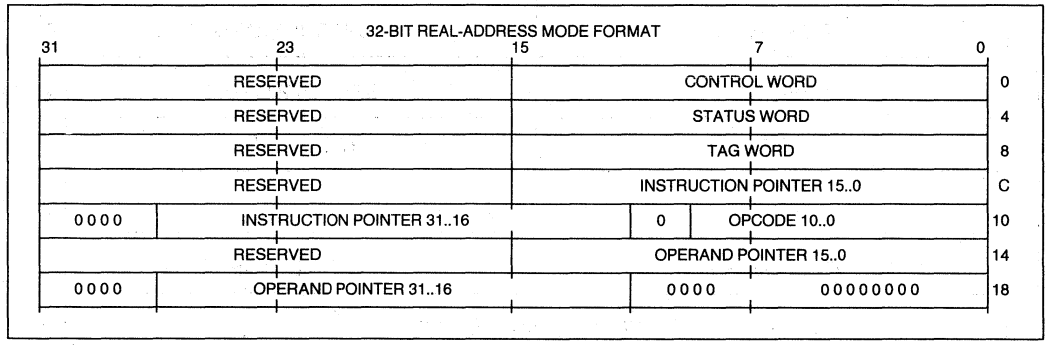


Figure 2-6. Instruction and Data Pointer Image in Memory, 32-bit Real-Mode Format

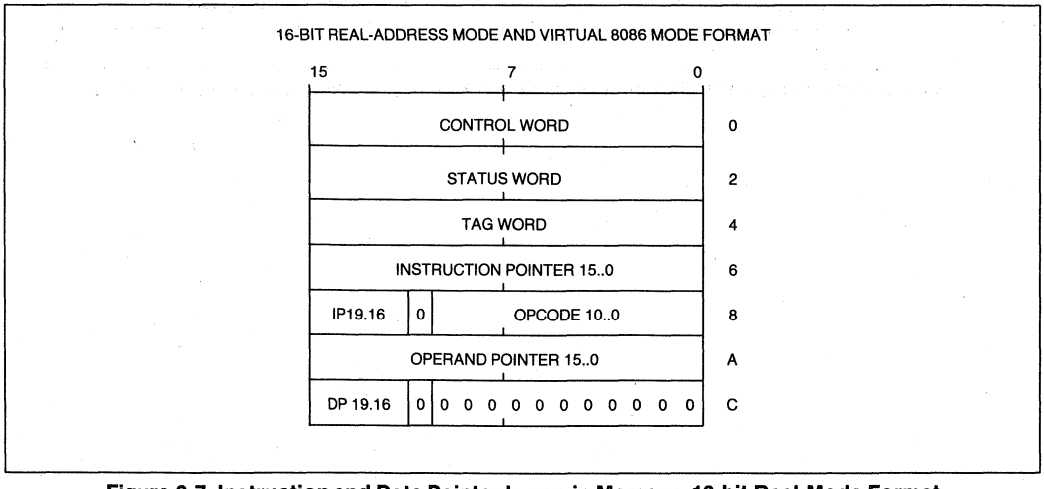


Figure 2-7. Instruction and Data Pointer Image in Memory, 16-bit Real-Mode Format

Table 2-6. CPU Interrupt Vectors Reserved for NPX

Interrupt Number	Cause of Interrupt
7	An ESC instruction was encountered when EM or TS of CPU control register zero (CR0) was set. EM = 1 indicates that software emulation of the instruction is required. When TS is set, either an ESC or WAIT instruction causes interrupt 7. This indicates that the current NPX context may not belong to the current task.
9	In a protected-mode system, an operand of a coprocessor instruction wrapped around an addressing limit (0FFFFH for expand-up segments, zero for expand-down segments) and spanned inaccessible addresses[a]. The failing numerics instruction is not restartable. The address of the failing numerics instruction and data operand may be lost; an FSTENV does not return reliable addresses. The segment overrun exception should be handled by executing an FNINIT instruction (i.e. an FINIT without a preceding WAIT). The exception can be avoided by never allowing numerics operands to cross the end of a segment.
13	In a protected-mode system, the first word of a numeric operand is not entirely within the limit of its segment. The return address pushed onto the stack of the exception handler points at the ESC instruction that caused the exception, including any prefixes. The NPX has not executed this instruction; the instruction pointer and data pointer register refer to a previous, correctly executed instruction.
16	The previous numerics instruction caused an unmasked exception. The address of the faulty instruction and the address of its operand are stored in the instruction pointer and data pointer registers. Only ESC and WAIT instructions can cause this interrupt. The CPU return address pushed onto the stack of the exception handler points to a WAIT or ESC instruction (including prefixes). This instruction can be restarted after clearing the exception condition in the NPX. FNINIT, FNCLEX, FNSTSW, FNSTENV, and FNSAVE cannot cause this interrupt.

a. An operand may wrap around an addressing limit when the segment limit is near an addressing limit and the operand is near the largest valid address in the segment. Because of the wrap-around, the beginning and ending addresses of such an operand will be at opposite ends of the segment. There are two ways that such an operand may also span inaccessible addresses: 1) if the segment limit is not equal to the addressing limit (e.g. addressing limit is FFFFH and segment limit is FFFDH) the operand will span addresses that are not within the segment (e.g. an 8-byte operand that starts at valid offset FFFCH will span addresses FFFC–FFFFH and 0000-0003H; however addresses FFFEH and FFFFH are not valid, because they exceed the limit); 2) if the operand begins and ends in present and accessible segments but intermediate bytes of the operand fall in a not-present page or in a segment or page to which the procedure does not have access rights.

2.4 Interrupt Description

CPU interrupts are used to report exceptional conditions while executing numeric programs in either real or protected mode. Table 2-6 shows these interrupts and their functions.

2.5 Exception Handling

The NPX detects six different exception conditions that can occur during instruction execution. Table 2-7 lists the exception conditions in order of precedence, showing for each the cause and the default action taken by the NPX if the exception is masked by its corresponding mask bit in the control word.

Any exception that is not masked by the control word sets the corresponding exception flag of the status word, sets the ES bit of the status word, and asserts the ERROR# signal. When the CPU attempts to execute another ESC instruction or WAIT, exception 16 occurs. The exception condition must be resolved via an interrupt service routine. The return address pushed onto the CPU stack upon entry to the service routine does not necessarily point to the failing instruction nor to the following instruction. The CPU saves the address of the floating-point instruction that caused the exception and the address of any memory operand required by that instruction.

2.6 Initialization

After FNINIT or RESET, the control word contains the value 037FH (all exceptions masked, precision control 64 bits, rounding to nearest) the same values as in an 80287 after RESET. For compatibility with the 8087 and 80287, the bit that used to indicate infinity control (bit 12) is set to zero; however, regardless of its setting, infinity is treated in the affine sense. After FNINIT or RESET, the status word is initialized as follows:

- All exceptions are set to zero.
- Stack TOP is zero, so that after the first push the stack top will be register seven (111B).
- The condition code C_3–C_0 is undefined.
- The B-bit is zero.

Table 2-7. Exceptions

Exception	Cause	Default Action (if exception is masked)
Invalid Operation	Operation on a signalling NaN, unsupported format, indeterminate for (0-∞, 0/0, (+ ∞) + (− ∞), etc.), or stack overflow/underflow (SF is also set)	Result is a quiet NaN, integer indefinite, or BCD indefinte
Denormalized Operand	At least one of the operands is denormalized, i.e., it has the smallest exponent but a nonzero significand.	Normal processing continues
Zero Divisor	The divisor is zero while the dividend is a noninfinite, nonzero number	Result is ∞
Overflow	The result is too large in magnitude to fit in the specified format	Result is largest finite value or ∞
Underflow	The true result is nonzero but too small to be represented in the specified format, and, if underflow exception is masked, denormalization causes the loss of accuracy.	Result is denormalized or zero
Inexact Result (Precision	The true result is not exactly representable in the specified format (e.g. 1/3); the result is rounded according to the rounding mode.	Normal processing continues

The tag word contains FFFFH (all stack locations are empty).

The 386 SX Microprocessor and 387 SX Math Coprocessor initialization software must execute an FNINIT instruction (i.e an FINIT without a preceding WAIT) after RESET. The FNINIT is not strictly required for the 80287 software, but Intel recommends its use to help ensure upward compatibility with other processors. After a hardware RESET, the ERROR# output is asserted to indicate that a 387 SX NPX is present. To accomplish this, the IE and ES bits of the status word are set, and the IM bit in the control word is cleared. After FNINIT, the status word and the control word have the same values as in an 80287 after RESET.

2.7 8087 and 80287 Compatibility

This section summarizes the differences between the 387 SX NPX and the 80287. Any migration from the 8087 directly to the 387 SX NPX must also take into account the differences between the 8087 and the 80287 as listed in Appendix A.

Many changes have been designed into the 387 SX NPX to directly support the IEEE standard in hardware. These changes result in increased performance by eliminating the need for software that supports the standard.

2.7.1 GENERAL DIFFERENCES

The 387 SX NPX supports only affine closure for infinity arithmetic, not projective closure.

Operands for FSCALE and FPATAN are no longer restricted in range (except for ± ∞); F2XM1 and FPTAN accept a wider range of operands.

Rounding control is in effect for FLD *constant.*

Software cannot change entries of the tag word to values (other than empty) that differ from actual register contents.

After reset, FINIT, and incomplete FPREM, the 387 SX NPX resets to zero the condition code bits C_3–C_0 of the status word.

In conformance with the IEEE standard, the 387 SX NPX does not support the special data formats pseudozero, pseudo-NaN, pseudoinfinity, and unnormal.

The denormal exception has a different purpose on the 387 SX NPX. A system that uses the denormal-exception handler solely to normalize the denormal operands, would better mask the denormal exception on the 387 SX NPX. The 387 SX NPX automatically normalizes denormal operands when the denormal exception is masked.

2.7.2 EXCEPTIONS

A number of differences exist due to changes in the IEEE standard and to functional improvements to the architecture of the 387 SX NPX:

1. When the overflow or underflow exception is masked, the 387 SX NPX differs from the 80287 in rounding when overflow or underflow occurs. The 387 SX NPX produces results that are consistent with the rounding mode.
2. When the underflow exception is masked, the 387 SX NPX sets its underflow flag only if there is also a loss of accuracy during denormalization.
3. Fewer invalid-operation exceptions due to denormal operands, because the instructions FSQRT, FDIV, FPREM, and conversions to BCD or to integer normalize denormal operands before proceeding.
4. The FSQRT, FBSTP, and FPREM instructions may cause underflow, because they support denormal operands.
5. The denormal exception can occur during the transcendental instructions and the FXTRACT instruction.
6. The denormal exception no longer takes precedence over all other exceptions.
7. When the denormal exception is masked, the 387 SX NPX automatically normalizes denormal operands. The 8087/80287 performs unnormal arithmetic, which might produce an unnormal result.
8. When the operand is zero, the FXTRACT instruction reports a zero-divide exception and leaves $-\infty$ in ST(1).
9. The status word has a new bit (SF) that signals when invalid-operation exceptions are due to stack underflow or overflow.
10. FLD *extended precision* no longer reports denormal exceptions, because the instruction is not numeric.
11. FLD *single/double precision* when the operand is denormal converts the number to extended precision and signals the denormalized operand exception. When loading a signalling NaN, FLD *single/double precision* signals an invalid-operand exception.
12. The 387 SX NPX only generates quiet NaNs (as on the 80287); however, the 387 SX NPX distinguishes between quiet NaNs and signaling NaNs. Signaling NaNs trigger exceptions when they are used as operands; quiet NaNs do not (except for FCOM, FIST, and FBSTP which also raise IE for quiet NaNs).
13. When stack overflow occurs during FPTAN and overflow is masked, both ST(0) and ST(1) contain quiet NaNs. The 80287/8087 leaves the original operand in ST(1) intact.
14. When the scaling factor is $\pm\infty$, the FSCALE (ST(0), ST(1)) instruction behaves as follows (ST(0) and ST(1) contain the scaled and scaling operands respectively):
    - FSCALE(0, ∞) generates the invalid operation exception.
    - FSCALE(finite, $-\infty$) generates zero with the same sign as the scaled operand.
    - FSCALE(finite, $+\infty$) generates ∞ with the same sign as the scaled operand.

    The 8087/80287 returns zero in the first case and raises the invalid-operation exception in the other cases.
15. The 387 SX NPX returns signed infinity/zero as the unmasked response to massive overflow/underflow. The 8087 and 80287 support a limited range for the scaling factor; within this range either massive overflow/underflow do not occur or undefined results are produced.

3.0 HARDWARE INTERFACE

In the following description of hardware interface, the # symbol at the end of a signal name indicates that the active or asserted state occurs when the signal is at a low voltage. When no # is present after the signal name, the signal is asserted when at the high voltage level.

5

3.1 Signal Description

In the following signal descriptions, the 387 SX NPX pins are grouped by function as shown by Table 3-1. Table 3-1 lists every pin by its identifier, gives a brief description of its function, and lists some of its characteristics (Refer to Figure 5-1 and Table 5-1 for pin configuration).

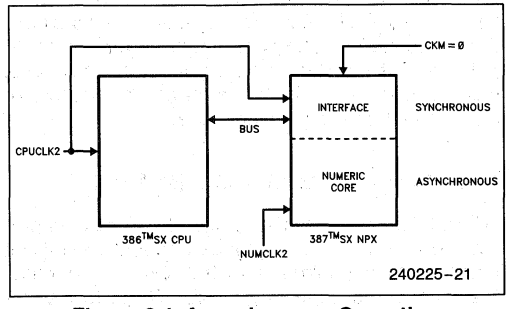


Figure 3.1. Asynchronous Operation

Table 3-1. Pin Summary

Pin Name	Function	Active State	Input/ Output	Referenced To...
Execution Control				
CPUCLK2	386™ SX Microprocessor CLocK 2		I	
NUMCLK2	NPX CLocK 2		I	
CKM	NPX ClocKing Mode		I	
RESETIN	System reset	High	I	CPUCLK2
NPX Handshake				
PEREQ	Processor Extension REQuest	High	O	STEN/CPUCLK2
BUSY#	Busy status	Low	O	STEN/CPUCLK2
ERROR#	Error status	Low	O	STEN/NUMCLK2
Bus Interface				
D15–D0	Data pins	High	I/O	CPUCLK2
W/R#	Write/Read bus cycle	Hi/Lo	I	CPUCLK2
ADS#	ADdress Strobe	Low	I	CPUCLK2
READY#	Bus ready input	Low	I	CPUCLK2
READYO#	Ready output	Low	O	STEN/CPUCLK2
Chip/Port Select				
STEN	STatus ENable	High	I	CPUCLK2
NPS1#	NPX select #1	Low	I	CPUCLK2
NPS2	NPX select #2	High	I	CPUCLK2
CMD0#	CoMmanD	Low	I	CPUCLK2
Power and Ground				
V_{CC}	System power			
V_{SS}	System ground			

All output signals are tristate; they leave floating state only when STEN is active. The output buffers of the bidirectional data pins D15–D0 are also tristate; they leave floating state only during cycles when the NPX is selected (i.e. when STEN, NPS1#, and NPS2 are all active).

3.1.1 386™ SX CPU CLOCK 2 (CPUCLK2)

This input uses the CLK2 signal of the CPU to time the bus control logic. Several other NPX signals are referenced to the rising edge of this signal. When CKM = 1 (synchronous mode) this pin also clocks the data interface and control unit and the floating-point unit of the NPX. This pin requires MOS-level input. The signal on this pin is divided by two to produce the internal clock signal CLK.

3.1.2 387™ SX NPX CLOCK 2 (NUMCLK2)

When CKM = 0 (asynchronous mode) this pin provides the clock for the data interface and control unit and the floating-point unit of the NPX. In this case, the ratio of the frequency of NUMCLK2 to the frequency of CPUCLK2 must lie within the range 10:16 to 14:10. When CKM = 1 (synchronous mode) signals on this pin are ignored; CPUCLK2 is used instead for the data interface and control unit and the floating-point unit. This pin requires MOS-level input.

3.1.3 CLOCKING MODE (CKM)

This pin is a strapping option. When it is strapped to V_{CC} (HIGH), the NPX operates in synchronous mode; when strapped to V_{SS} (LOW), the NPX operates in asynchronous mode. These modes relate to clocking of the data interface and control unit and the floating-point unit only; the bus control logic always operates synchronously with respect to the CPU.

3.1.4 SYSTEM RESET (RESETIN)

A LOW to HIGH transition on this pin causes the NPX to terminate its present activity and to enter a dormant state. RESETIN must remain active (HIGH) for at least 40 NUMCLK2 periods.

The HIGH to LOW transitions of RESETIN must be synchronous with CPUCLK2, so that the phase of the internal clock of the bus control logic (which is the CPUCLK2 divided by two) is the same as the phase of the internal clock of the CPU. After RESETIN goes LOW, at least 50 NUMCLK2 periods must pass before the first NPX instruction is written into the NPX. This pin should be connected to the CPU RESET pin. Table 3-1 shows the status of the output pins during the reset sequence. After a reset, all output pins return to their inactive states.

Table 3-2. Output Pin Status during Reset

Pin Value	Pin Name
HIGH	READYO#, BUSY#
LOW	PEREQ, ERROR#
Tri-State OFF	D15–D0

3.1.5 PROCESSOR EXTENSION REQUEST (PEREQ)

When active, this pin signals to the CPU that the NPX is ready for data transfer to/from its data FIFO. When all data is written to or read from the data FIFO, PEREQ is deactivated. This signal always goes inactive before BUSY# goes inactive. This signal is referenced to CPUCLK2. It should be connected to the CPU PEREQ input.

3.1.6 BUSY STATUS (BUSY#)

When active, this pin signals to the CPU that the NPX is currently executing an instruction. This signal is referenced to CPUCLK2. It should be connected to the CPU BUSY# pin.

3.1.7 ERROR STATUS (ERROR#)

This pin reflects the ES bit of the status register. When active, it indicates that an unmasked exception has occurred. This signal can be changed to inactive state only by the following instructions (without a preceding WAIT): FNINIT, FNCLEX, FNSTENV, FNSAVE, FLDCW, FLDENV, and FRSTOR. This pin is referenced to CPUCLK2. It should be connected to the ERROR# pin of the CPU.

3.1.8 DATA PINS (D15–D0)

These bidirectional pins are used to transfer data and opcodes between the CPU and NPX. They are normally connected directly to the corresponding CPU data pins. HIGH state indicates a value of one. D0 is the least significant data bit. Timings are referenced to CPUCLK2.

3.1.9 WRITE/READ BUS CYCLE (W/R#)

This signal indicates to the NPX whether the CPU bus cycle in progress is a read or a write cycle. This pin should be connected directly to the CPU's W/R# pin. HIGH indicates a write cycle; LOW a read cycle. This input is ignored if any of the signals STEN, NPS1#, or NPS2 is inactive. Setup and hold times are referenced to CPUCLK2.

3.1.10 ADDRESS STROBE (ADS#)

This input, in conjunction with the READY# input, indicates when the NPX bus-control logic may sample W/R# and the chip-select signals. Setup and hold times are referenced to CPUCLK2. This pin should be connected to the ADS# pin of the CPU.

3.1.11 BUS READY INPUT (READY#)

This input indicates to the NPX when a CPU bus cycle is to be terminated. It is used by the bus-control logic to trace bus activities. Bus cycles can be extended indefinitely until terminated by READY#. This input should be connected to the same signal that drives the CPU's READY# input. Setup and hold times are referenced to CPUCLK2.

3.1.12 READY OUTPUT (READYO#)

This pin is activated at such a time that write cycles are terminated after two clocks (except FLDENV and FRSTOR) and read cycles after three clocks. In configurations where no extra wait states are required, this pin must directly or indirectly drive the READY# input of the CPU. Refer to the section entitled "Bus Operation" for details. This pin is activated only during bus cycles that select the NPX. This signal is referenced to CPUCLK2.

3.1.13 STATUS ENABLE (STEN)

This pin serves as a chip select for the NPX. When inactive, this pin forces, BUSY#, PEREQ#, ERROR#, and READYO# outputs into floating state. D15–D0 are normally floating; they leave floating state only if STEN is active and additional conditions are met. STEN also causes the chip to recognize its other chip-select inputs. STEN makes it easier to do on-board testing (using the overdrive method) of other chips in systems containing the NPX. STEN should be pulled up with a resistor so that it can be pulled down when testing. In boards that do not use on-board testing. STEN should be connected to V_{CC}. Setup and hold times are relative to CPUCLK2. Note that STEN must maintain the same setup and hold times as NPS1#, NPS2, and CMD0# (i.e. if STEN changes state during an NPX bus cycle, it must change state during the same CLK period as the NPS1#, NPS2, and CMD0# signals).

3.1.14 NPX SELECT 1 (NPS1#)

When active (along with STEN and NPS2) in the first period of a CPU bus cycle, this signal indicates that the purpose of the bus cycle is to communicate with the NPX. This pin should be connected directly to the M/IO# pin of the CPU, so that the NPX is selected only when the CPU performs I/O cycles. Setup and hold times are referenced to CPUCLK2.

3.1.15 NPX SELECT 2 (NPS2)

When active (along with STEN and NPS1#) in the first period of a CPU bus cycle, this signal indicates that the purpose of the bus cycle is to communicate with the NPX. This pin should be connected directly to the A23 pin of the CPU, so that the NPX is selected only when the CPU issues one of the I/O addresses reserved for the NPX (8000F8H, 8000FCH or 8000FEH which is treated as 8000FCH by the NPX). Setup and hold times are referenced to CPUCLK2.

3.1.16 COMMAND (CMD0#)

During a write cycle, this signal indicates whether an opcode (CMD0# active) or data (CMD0# inactive) is being sent to the NPX. During a read cycle, it indicates whether the control or status register (CMD0# active) or a data register (CMD0# inactive) is being read. CMD0# should be connected directly to the A2 output of the CPU. Setup and hold times are referenced to CPUCLK2.

3.1.17 SYSTEM POWER (V_{CC})

System power provides the +5V DC supply input. All V_{CC} pins should be tied together on the circuit board and local decoupling capacitors should be used between V_{CC} and V_{SS}.

3.1.18 SYSTEM GROUND (V_{SS})

All V_{SS} pins should be tied together on the circuit board and local decoupling capacitors should be used between V_{CC} and V_{SS}.

3.2 System Configuration

The 387 SX Math Coprocessor is designed to interface with the 386 SX Microprocessor as shown by Figure 3-1. A dedicated communication protocol makes possible high-speed transfer of opcodes and operands between the CPU and NPX. The 387 SX NPX is designed so that no additional components are required for interface with the CPU. Most control pins of the NPX are connected directly to pins of the CPU.

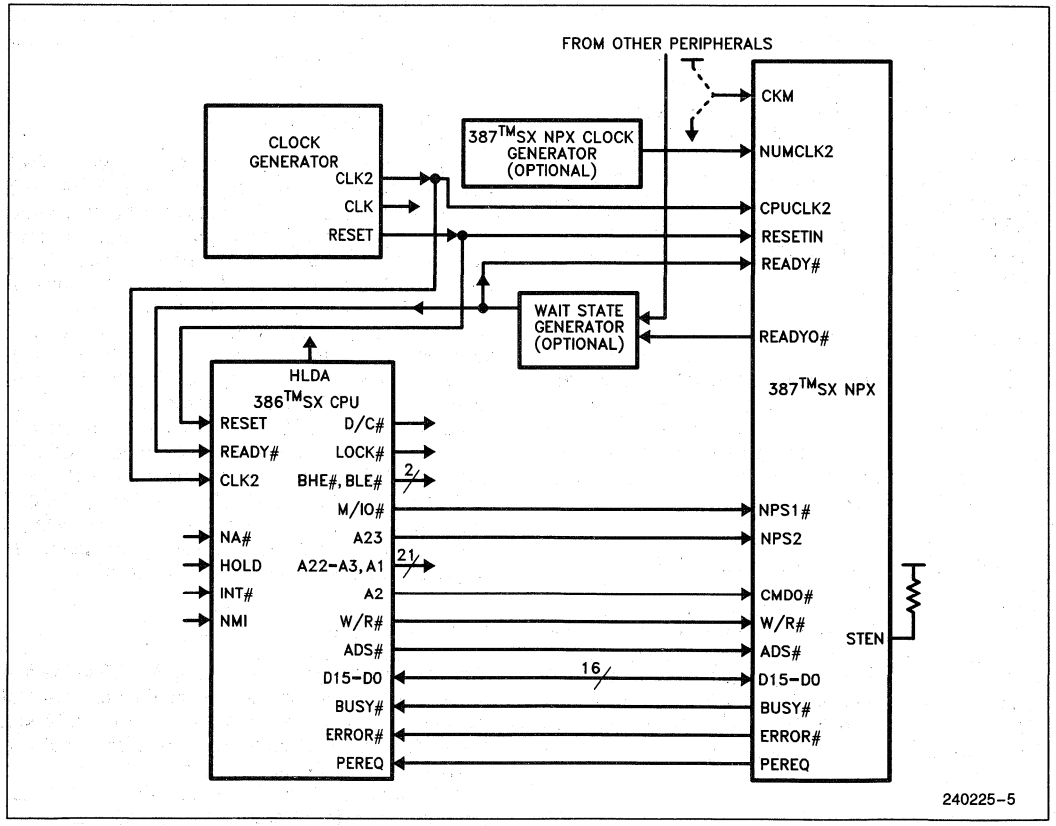


Figure 3-2. 386™ SX CPU and 387™ SX NPX System Configuration

The interface between the NPX and the CPU has these characteristics:

- The NPX shares the local bus of the 386 SX Microprocessor.
- The CPU and NPX share the same reset signals. They may also share the same clock input; however, for greatest performance, an external oscillator may be needed.
- The corresponding BUSY#, ERROR#, and PEREQ pins are connected together.
- The NPX NPS1# and NPS2 inputs are connected to the latched CPU M/IO# and A23 outputs respectively. For coprocessor cycles, M/IO# is always LOW and A23 always HIGH.
- The NPX input CMD0 is connected to the latched A_2 output. The 386 SX Microprocessor generates address 8000F8H when writing a command and address 8000FCH or 8000FEH (treated as 8000FCH by the 387 SX NPX) when writing or reading data. It does not generate any other addresses during NPX bus cycles.

3.3 Processor Architecture

As shown by the block diagram on the front page, the 387 SX NPX is internally divided into three sections: the bus control logic (BCL), the data interface and control unit, and the floating point unit (FPU). The FPU (with the support of the control unit which contains the sequencer and other support units) executes all numerics instructions. The data interface and control unit is responsible for the data flow to and from the FPU and the control registers, for receiving the instructions, decoding them, and sequencing the microinstructions, and for handling some of the administrative instructions. The BCL is responsible for CPU bus tracking and interface. The BCL is the only unit in the NPX that must run synchronously with the CPU; the rest of the NPX can run asynchronously with respect to the CPU.

3.3.1 BUS CONTROL LOGIC

The BCL communicates solely with the CPU using I/O bus cycles. The BCL appears to the CPU as a special peripheral device. It is special in two respects: the CPU initiates I/O automatically when it encounters ESC instructions, and the CPU uses reserved I/O addresses to communicate with the BCL. The BCL does not communicate directly with memory. The CPU performs all memory access, transferring input operands from memory to the NPX and transferring outputs from the NPX to memory.

3.3.2 DATA INTERFACE AND CONTROL UNIT

The data interface and control unit latches the data and, subject to BCL control, directs the data to the FIFO or the instruction decoder. The instruction decoder decodes the ESC instructions sent to it by the CPU and generates controls that direct the data flow in the FIFO. It also triggers the microinstruction sequencer that controls execution of each instruction. If the ESC instruction is FINIT, FCLEX, FSTSW, FSTSW AX, FSTCW, FSETPM, or FRSTPM, the control executes it independently of the FPU and the sequencer. The data interface and control unit is the one that generates the BUSY#, PEREQ, and ERROR# signals that synchronize NPX activities with the CPU.

3.3.3 FLOATING-POINT UNIT

The FPU executes all instructions that involve the register stack, including arithmetic, logical, transcendental, constant, and data transfer instructions. The data path in the FPU is 84 bits wide (68 significant bits, 15 exponent bits, and a sign bit) which allows internal operand transfers to be performed at very high speeds.

3.4 Bus Cycles

The pins STEN, NPS1#, NPS2, CMD0, and W/R# identify bus cycles for the NPX. Table 3-3 defines the types of NPX bus cycles.

3.4.1 387™ SX NPX ADDRESSING

The NPS1#, NPS2, and CMD0 signals allow the NPX to identify which bus cycles are intended for the NPX. The NPX responds to I/O cycles when the I/O address is 8000F8H, 8000FCH or 8000FEH (treated

Table 3-3. Bus Cycle Definition

STEN	NPS1#	NPS2	CMD0#	W/R#	Bus Cycle Type
0	x	x	x	x	NPX not selected and all outputs in floating state
1	1	x	x	x	NPX not selected
1	x	0	x	x	NPX not selected
1	0	1	0	0	CW or SW read from NPX
1	0	1	0	1	Opcode write to NPX
1	0	1	1	0	Data read from NPX
1	0	1	1	1	Data write to NPX

as 8000FCH by the 387 SX NPX). The NPX responds to I/O cycles when bit 23 of the I/O address is set. In other words, the NPX acts as an I/O device in a reserved I/O address space.

Because A23 is used to select the 387 SX Numerics Coprocessor Extension for data transfers, it is not possible for a program running on the CPU to address the NPX with an I/O instruction. Only ESC instructions cause the CPU to communicate with the NPX.

3.4.2 CPU/NPX SYNCHRONIZATION

The pins BUSY#, PEREQ, and ERROR# are used for various aspects of synchronization between the CPU and the NPX.

BUSY# is used to synchronize instruction transfer from the CPU to the NPX. When the NPX recognizes an ESC instruction, it asserts BUSY#. For most ESC instructions, the CPU waits for the NPX to deassert BUSY# before sending the new opcode.

The NPX uses the PEREQ pin of the CPU to signal that the NPX is ready for data transfer to or from its data FIFO. The NPX does not directly access memory; rather, the CPU provides memory access services for the NPX. (For this reason, memory access on behalf of the NPX always obeys the protection rules applicable to the current CPU mode.) Once the CPU initiates an NPX instruction that has operands, the CPU waits for PEREQ signals that indicate when the NPX is ready for operand transfer. Once all operands have been transferred (or if the instruction has no operands) the CPU continues program execution while the NPX executes the ESC instruction.

In 8086/8087 systems, WAIT instructions may be required to achieve synchronization of both commands and operands. In the 386 SX Microprocessor and 387 SX Math Coprocessor systems, however, WAIT instructions are required only for operand synchronization; namely, after NPX stores to memory (except FSTSW and FSTCW) or load from memory. (In 80286/80287 systems, WAIT is required before FLDENV and FRSTOR; with the 386 SX Microprocessor and 387 SX Math Coprocessor, WAIT is not required in these cases.) Used this way, WAIT ensures that the value has already been written or read by the NPX before the CPU reads or changes the value.

Once it has started to execute a numerics instruction and has transferred the operands from the CPU, the NPX can process the instruction in parallel with and independent of the host CPU. When the NPX detects an exception, it asserts the ERROR# signal, which causes a CPU interrupt.

3.4.3 SYNCHRONOUS OR ASYNCHRONOUS MODES

The internal logic of the NPX (the FPU) can operate either directly from the CPU clock (synchronous mode) or from a separate clock (asynchronous mode). The two configurations are distinguished by the CKM pin. In either case, the bus control logic (BCL) of the NPX is synchronized with the CPU clock. Use of asynchronous mode allows the CPU and the FPU section of the NPX to run at different speeds. In this case, the ratio of the frequency of NUMCLK2 to the frequency of CPUCLK2 must lie within the range 10:16 to 14:10. Use of synchronous mode eliminates one clock generator from the board design.

3.4.4 AUTOMATIC BUS CYCLE TERMINATION

In configurations where no extra wait states are required, READYO# can drive the CPU's READY# input. If this pin is used, it should be connected to the logic that ORs all READY outputs from peripherals on the CPU bus. READYO# is asserted by the NPX only during I/O cycles that select the NPX. Refer to Section 4.0 "Bus Operation" for details.

4.0 BUS OPERATION

With respect to bus interface, the 387 SX NPX is fully synchronous with the CPU. Both operate at the same rate, because each generates its internal CLK signal by dividing CPUCLK2 by two. Furthermore, both internal CLK signals are in phase, because they are synchronized by the same RESETIN signal.

A bus cycle for the NPX starts when the CPU activates ADS# and drives new values on the address and cycle-definition lines. The NPX examines the address and cycle-definition lines in the same CLK period during which ADS# is activated. This CLK period is considered the first CLK of the bus cycle. During this first CLK period, the NPX also examines the R/W# input signal to determine whether the cycle is a read or a write cycle and examines the CMD0 input to determine whether an opcode, operand, or control/status register transfer is to occur.

The 387 SX NPX supports both pipelined (i.e. overlapped) and nonpipelined bus cycles. A nonpipelined cycle is one for which the CPU asserts ADS# when no other NPX bus cycle is in progress. A pipelined bus cycle is one for which the CPU asserts ADS# and provides valid next-address and control signals before the prior NPX cycle terminates. The CPU may do this as early as the second CLK period after asserting ADS# for the prior cycle. Pipelining increas-

es the availability of the bus by at least one CLK period. The 387 SX NPX supports pipelined bus cycles in order to optimize address pipelining by the CPU for memory cycles.

Bus operation is described in terms of an abstract state machine. Figure 4-1 illustrates the states and state transitions for NPX bus cycles:

- T_I is the idle state. This is the state of the bus logic after RESET, the state to which bus logic returns after every nonpipelined bus cycle, and the state to which bus logic returns after a series of pipelined cycles.
- T_{RS} is the READY#-sensitive state. Different types of bus cycles may require a minimum of one or two successive T_{RS} states. The bus logic remains in T_{RS} state until READY# is sensed, at which point the bus cycle terminates. Any number of wait states may be implemented by delaying READY#, thereby causing additional successive T_{RS} states.
- T_P is the first state for every pipelined bus cycle. This state is not used by nonpipelined cycles.

Note that the bus logic tracks bus state regardless of the values on the chip/port select pins.

The READYO# output of the NPX indicates when an NPX bus cycle may be terminated if no extra wait states are required. For all write cycles (except those for the instructions FLDENV and FRSTOR), READYO# is always asserted during the first T_{RS} state, regardless of the number of wait states. For all read cycles and write cycles for FLDENV and FRSTOR, READYO# is always asserted in the second T_{RS} state, regardless of the number of wait states. These rules apply to both pipelined and nonpipelined cycles. Systems designers may use READYO# in one of the following ways:

1. Connect it (directly or through logic that ORs READY# signals from other devices) to the READY# inputs of the CPU and NPX.
2. Use it as one input to a wait-state generator.

The following sections illustrate different types of 387 SX NPX bus cycles. Because different instructions have different amounts of overhead before, between, and after operand transfer cycles, it is not possible to represent in a few diagrams all of the combinations of successive operand transfer cycles. The following bus-cycle diagrams show memory cycles between NPX operand-transfer cycles. Note however that, during FRSTOR, some consecutive accesses to the NPX do not have intervening memory accesses. For the timing relationship between operand transfer cycles and opcode write or other overhead activities, see the figure "Other Parameters" in section 6.

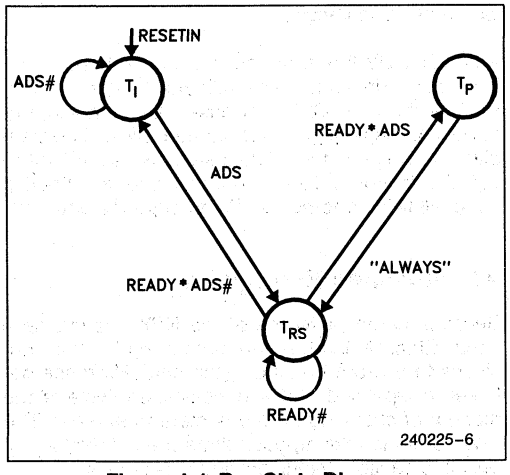


Figure 4-1. Bus State Diagram

4.1 Nonpipelined Bus Cycles

Figure 4-2 illustrates bus activity for consecutive nonpipelined bus cycles.

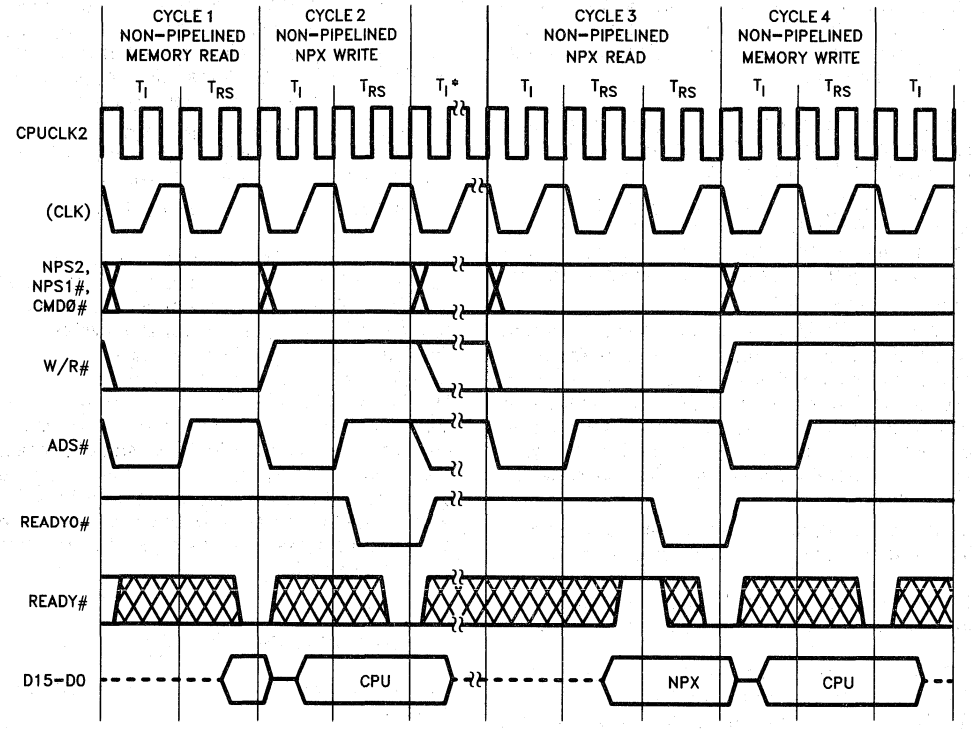
At the second clock of the bus cycle, the NPX enters the T_{RS} state. During this state, it samples the READY# input and stays in this state as long as READY# is inactive.

4.1.1 WRITE CYCLE

In write cycles, the NPX drives the READYO# signal for one CLK period during the second CLK period of the cycle (i.e. the first T_{RS} state); therefore, the fastest write cycle takes two CLK periods (see cycle 2 of Figure 4-2). For the instructions FLDENV and FRSTOR, however, the NPX forces a wait state by delaying the activation of READYO# to the second T_{RS} state (not shown in Figure 4-2).

The NPX samples the D15–D0 inputs into data latches at the falling edge of CLK as long as it stays in T_{RS} state.

When READY# is asserted, the NPX returns to the idle state. Simultaneously with the NPX's entering the idle state, the CPU may assert ADS# again, signaling the beginning of yet another cycle.



Cycles 1 & 2 represent part of the operand transfer cycle for instructions involving either 4-byte or 8-byte operand loads.
Cycles 3 & 4 represent part of the operand transfer cycle for a store operation.
*Cycles 1 & 2 could repeat here or T_I states for various non-operand transfer cycles and overhead.

Figure 4-2. Nonpipelined Read and Write Cycles

4.1.2 READ CYCLE

At the rising edge of CLK in the second CLK period of the cycle (i.e. the first T_{RS} state), the NPX starts to drive the D15–D0 outputs and continues to drive them as long as it stays in T_{RS} state.

At least one wait state must be inserted to ensure that the CPU latches the correct data. Because the NPX starts driving the data bus only at the rising edge of CLK in the second clock period of the bus cycle, not enough time is left for the data signals to propagate and be latched by the CPU before the next falling edge of CLK. Therefore, the NPX does not drive the READYO# signal until the third CLK period of the cycle. Thus, if the READYO# output drives the CPU's READY# input, one wait state is automatically inserted.

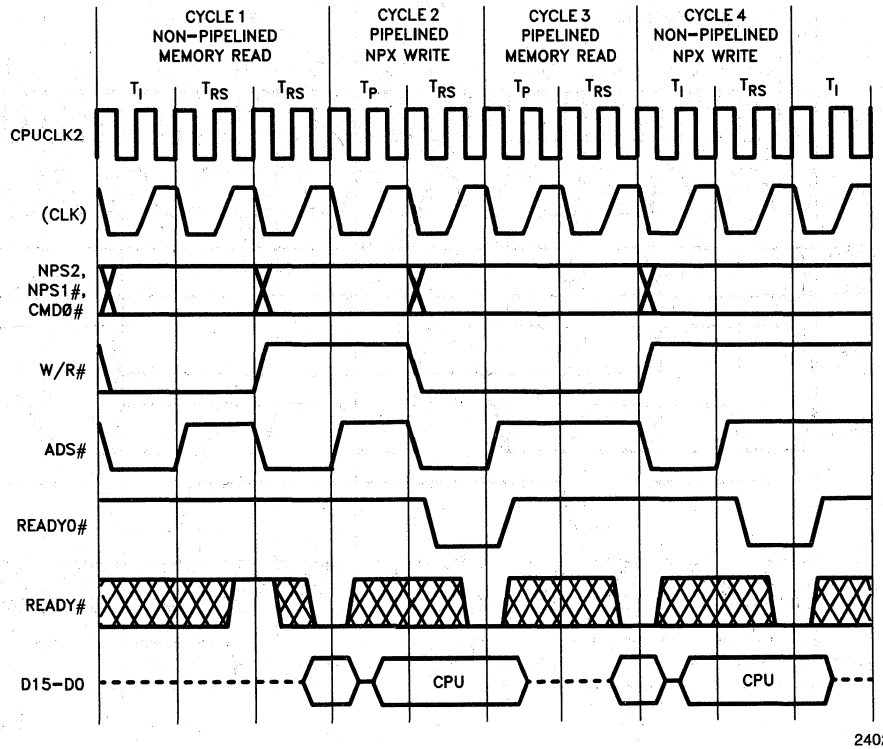
Because one wait state is required for NPX reads, the minimum length of an NPX read cycle is three CLK periods, as cycle 3 of Figure 4-2 shows.

When READY# is asserted, the NPX returns to the idle state. Simultaneously with the NPX's entering the idle state, the CPU may assert ADS# again, signaling the beginning of yet another cycle. The transition from T_{RS} state to idle state causes the NPX to put the tristate D15–D0 outputs into the floating state, allowing another device to drive the data bus.

4.2 Pipelined Bus Cycles

Because all the activities of the NPX bus interface occur either during the T_{RS} state or during the transitions to or from that state, the only difference between a pipelined and a nonpipelined cycle is the manner of changing from one state to another. The exact activities during each state are detailed in the previous section "Nonpipelined Bus Cycles".

When the CPU asserts ADS# before the end of a bus cycle, both ADS# and READY# are active dur-



Cycle 1–Cycle 4 represent the operand transfer cycle for an instruction involving a transfer of two 32-bit loads in total. The opcode write cycles and other overhead are not shown.
Note that the next cycle will be a pipelined cycle if both READY# and ADS# are sampled active at the end of a T_{RS} state of the current cycle.

Figure 4-3. Fastest Transitions to and from Pipelined Cycles

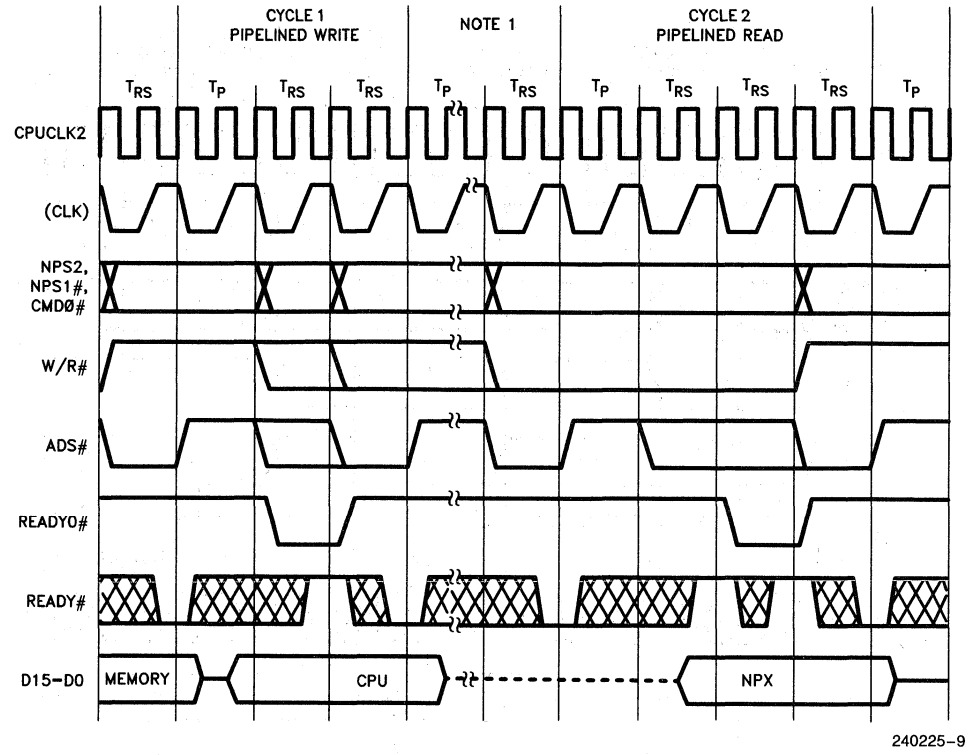
ing a T_{RS} state. This condition causes the NPX to change to a different state named T_P. One clock period after a T_P state, the NPX always returns to T_{RS} state. In consecutive pipelined cycles, the NPX bus logic uses only the T_{RS} and T_P states.

Figure 4-3 shows the fastest transitions into and out of the pipelined bus cycles. Cycle 1 in the figure represents a nonpipelined cycle. (Nonpipelined write cycles with only one T_{RS} state (i.e. no wait states) are always followed by another nonpipelined cycle, because READY# is asserted before the earliest possible assertion of ADS# for the next cycle.)

Figure 4-4 shows pipelined write and read cycles with one additional T_{RS} state beyond the minimum required. To delay the assertion of READY# requires external logic.

4.3 Bus Cycles of Mixed Type

When the NPX bus logic is in the T_{RS} state, it distinguishes between nonpipelined and pipelined cycles according to the behavior of ADS# and READY#. In a nonpipelined cycle, only READY# is activated, and the transition is from T_{RS} state to idle state. In a



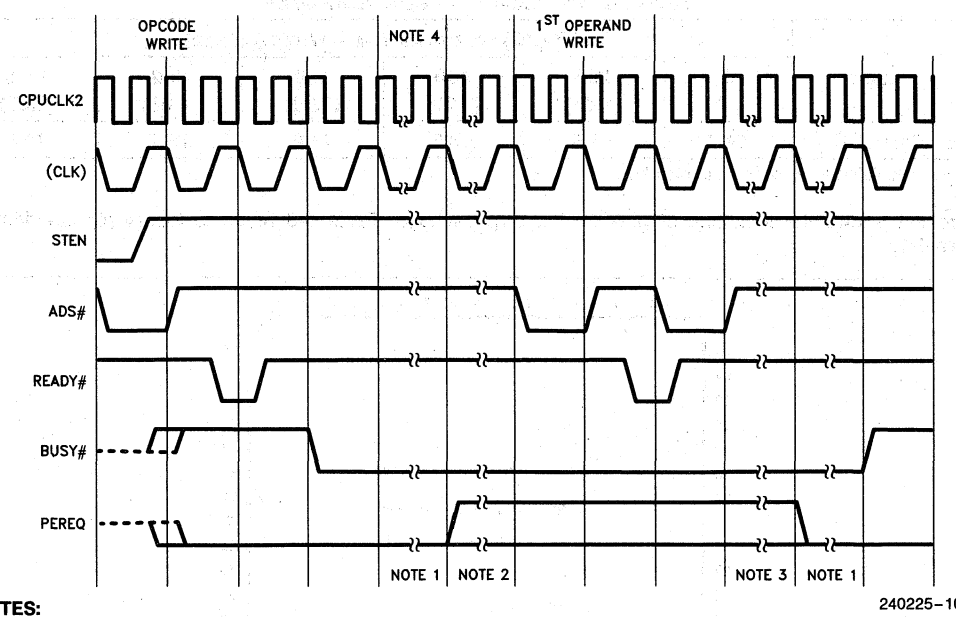
NOTE:
1. Cycles between operand write to the NPX and storing result.

Figure 4-4. Pipelined Cycles with Wait States

pipelined cycle, both READY# and ADS# are active, and the transition is first from T_{RS} state to T_P state, then, after one clock period, back to T_{RS} state.

4.4 BUSY# and PEREQ Timing Relationship

Figure 4-5 shows the activation of BUSY# at the beginning of instruction execution and its deactivation upon completion of the instruction. PEREQ is activated within this interval. If ERROR# (not shown in the figure) is ever asserted, it would be asserted at least six CPUCLK2 periods after the deactivation of PEREQ and would be deasserted at least six CPUCLK2 periods before the deactivation of BUSY#. Figure 4-5 also shows that STEN is activated at the beginning of an NPX bus cycle.



NOTES:
1. Instruction dependent.
2. PEREQ is an asynchronous input to the 386™ Microprocessor; it may not be asserted (instruction dependent).
3. More operand transfers.
4. Memory read (operand) cycle is not shown.

Figure 4-5. STEN, BUSY#, and PEREQ Timing Relationships

5

5.0 PACKAGE THERMAL SPECIFICATIONS

The 387 SX Math Coprocessor is specified for operation when case temperature is within the range of 0°C–100°C. The case temperature may be measured in any environment, to determine whether the 387 SX Math Coprocessor is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

The ambient temperature is guaranteed as long as T_C is not violated. The ambient temperature can be calculated from the θ_{jc} and θ_{ja} from the following equations:

$$T_j = T_c + P * \theta_{jc}$$

$$T_a = T_j - P * \theta_{ja}$$

$$T_c = T_a + P * [\theta_{ja} - \theta_{jc}]$$

Values for θ_{ja} and θ_{jc} are given in Table 5-1 for the 68-pin PLCC. θ_{ja} is given at various airflows. Table 5-2 shows the maximum T_a allowable (without exceeding T_C) at various airflows. Note that T_a can be improved further by attaching 'fins' or a 'heat sink' to the package. P is calculated by using the maximum hot I_{CC}.

Table 5-1. Thermal Resistances (°C/Watt) θ_{jc} and θ_{ja}

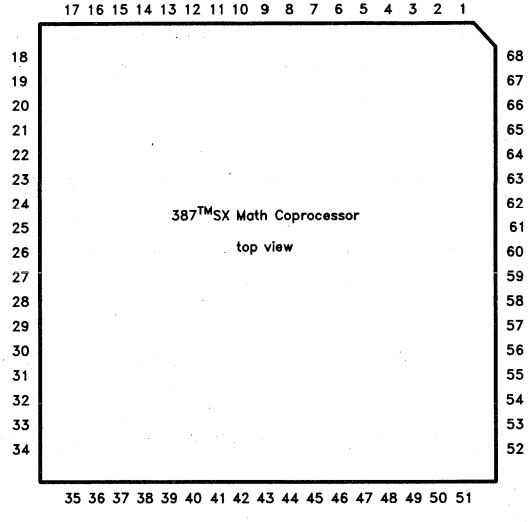
Package	θ_{jc}	θ_{ja} versus Airflow - ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
68-Pin PLCC	8	30	25	20	15.5	13	12

Table 5-2. Maximum T_A at Various Airflows

Package	T_A(°C) versus Airflow - ft/min (m/sec)					
	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
68-Pin PLCC	54.7	61.6	68.5	74.6	78.1	79.5

Max. T_A calculated at Max V_{CC} and Max I_{CC}.

Figure 5-1 shows the locations of pins on the chip package. Table 5-3 helps to locate pin identifiers in Figure 5-1.



The term "top view" means "as viewed when mounted in a printed-circuit board".

Figure 5-1. PLCC Pin Configuration

Table 5-3. Pin Cross-Reference

1 — n.c.	18 — n.c.	35 — ERROR#	52 — n.c.
2 — D07	19 — D00	36 — BUSY#	53 — NUMCLK2
3 — D06	20 — D01	37 — V_{CC}	54 — CPUCLK2
4 — V_{CC}	21 — V_{SS}	38 — V_{SS}	55 — V_{SS}
5 — V_{SS}	22 — V_{CC}	39 — V_{CC}	56 — PEREQ
6 — D05	23 — D02	40 — STEN	57 — READYO#
7 — D04	24 — D08	41 — W/R#	58 — V_{CC}
8 — D03	25 — V_{SS}	42 — V_{SS}	59 — CKM
9 — V_{CC}	26 — V_{CC}	43 — V_{CC}	60 — V_{SS}
10 — n.c.	27 — V_{SS}	44 — NPS1#	61 — V_{SS}
11 — D15	28 — D09	45 — NPS2	62 — V_{CC}
12 — D14	29 — D10	46 — V_{CC}	63 — V_{SS}
13 — V_{CC}	30 — D11	47 — ADS#	64 — V_{CC}
14 — V_{SS}	31 — V_{CC}	48 — CMD0#	65 — n.c.
15 — D13	32 — V_{SS}	49 — READY#	66 — V_{SS}
16 — D12	33 — V_{CC}	50 — V_{CC}	67 — n.c.
17 — n.c.	34 — V_{SS}	51 — RESETIN	68 — n.c.

n.c.—The corresponding pins of the 387™ SX NPX are left unconnected.

6.0 ELECTRICAL DATA

6.1 Absolute Maximum Ratings

NOTE:
Stresses above those listed may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Case temperature T_C under bias .....0°C to 100°C
Storage temperature .....−65°C to +150°C
Voltage on any pin with respect to ground −0.5 to V_{CC}+0.5V
Power dissipation .....1.5 Watt

6.2 D.C. Characteristics

Table 6-1. D.C. Specifications T_C = 0° to 100°C, V_{CC} = 5V ± 10%

Symbol	Parameter	Min	Max	Units	Test Conditions
V_{IL}	Input LO Voltage	−0.3	+0.8	V	See note 1
V_{IH}	Input HI Voltage	2.0	V_{CC}+0.3	V	See note 1
V_{CL}	CPUCLK2 and NUMCLK2 Input LO Voltage	−0.3	+0.8	V	
V_{CH}	CPUCLK2 and NUMCLK2 Input HI Voltage	V_{CC}−0.8	V_{CC}+0.3	V	
V_{OL}	Output LO Voltage		0.45	V	See note 2
V_{OH}	Output HI Voltage	2.4		V	See note 3
V_{OH}	Output HI Voltage	V_{CC}−0.8		V	See note 4
I_{CC}	Power Supply Current				
	NUMCLK2 = 40 MHz(5)		300	mA	I_{CC} typ. = 200 mA
	NUMCLK2 = 32 MHz(5)		250	mA	I_{CC} typ. = 150 mA
	NUMCLK2 = 2 MHz(5)		100	mA	
I_{LI}	Input Leakage Current		±15	μA	$0V \le V_{IN} \le V_{CC}$
I_{LO}	I/O Leakage Current		±15	μA	$0.45V \le V_O \le V_{CC}$
C_{IN}	Input Capacitance		10	pF	fc = 1MHz
C_O	I/O or Output Capacitance		12	pF	fc = 1MHz
C_{CLK}	Clock Capacitance		20	pF	fc = 1MHz

NOTES:
1. This parameter is for all inputs, excluding the clock inputs.
2. This parameter is measured at I_{OL} as follows:
   data = 4.0mA
   READY0#, ERROR#, BUSY#, PEREQ = 2.5mA
3. This parameter is measured at I_{OH} as follows:
   data = 1.0mA
   READY0#, ERROR#, BUSY#, PEREQ = 0.6mA
4. This parameter is measured at I_{OH} as follows:
   data = 0.2mA
   READY0#, ERROR#, BUSY#, PEREQ = 0.12mA
5. I_{CC} is measured at steady state, maximum capacitive loading on the outputs, and worst-case D.C. level at the inputs; CPUCLK2 at the same frequency as NUMCLK2.

6.3 A.C. Characteristics

Table 6-2a. Combinations of Bus Interface and Execution Speeds

Functional Block	80387SX-16	80387SX-20
Bus Interface Unit (MHz)	16	20
Execution Unit (MHz)	16	20

Table 6-2b. Timing Requirements of Execution Unit $T_C = 0°$ to $100°$ C, $V_{CC} = 5V \pm 10\%$

Pin	Symbol	Parameter	16 MHz		20 MHz		Test Conditions	Refer to Figure
			Min (ns)	Max (ns)	Min (ns)	Max (ns)		
NUMCLK2	t1	Period	31.25	500	25	500	2.0V	6.2
NUMCLK2	t2a	High Time	7		6		2.0V	
NUMCLK2	t2b	High Time	3		3		$V_{CC}-0.8V$	
NUMCLK2	t3a	Low Time	7		6		2.0V	
NUMCLK2	t3b	Low Time	5		4		0.8V	
NUMCLK2	t4	Fall Time		8		8	From $V_{CC}-0.8$ to 0.8V	(Note 1)
NUMCLK2	t5	Rise Time		8		8	From 0.8 to $V_{CC}-0.8V$	

NOTE:
1. If not used (CKM = 1), tie LOW.

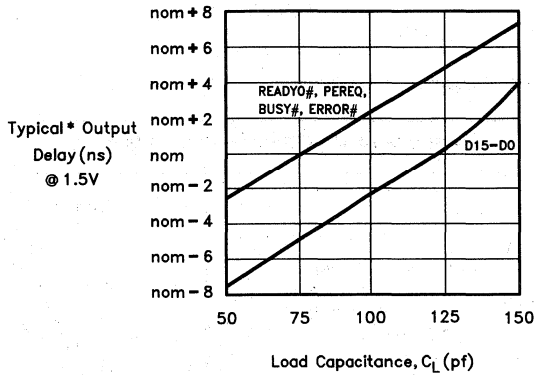
Table 6-2c. Timing Requirements of Bus Interface Unit $T_C = 0°$ to $100°$ C, $V_{CC} = 5V \pm 10\%$

Pin	Symbol	Parameter	16 MHz (1.5V)		20 MHz (1.5V)		Test Conditions	Refer to Figure
			Min (ns)	Max (ns)	Min (ns)	Max (ns)		
CPUCLK2	t1	Period	31.25	500	25	500	2.0V	6.2
CPUCLK2	t2a	High Time	7		6		2.0V	
CPUCLK2	t2b	High Time	3		3		$V_{CC}-0.8V$	
CPUCLK2	t3a	Low Time	7		6		2.0V	
CPUCLK2	t3b	Low Time	5		4		0.8V	
CPUCLK2	t4	Fall Time		8		8	From $V_{CC}-0.8$ to 0.8V	
CPUCLK2	t5	Rise Time		8		8	From 0.8 to $V_{CC}-0.8V$	
CPUCLK2/ NUMCLK2		Ratio	10/16	14/10	10/16	14/10		
READYO#	t7	Out Delay	4	34	3	31	$C_L = 75pf$	6.3
READYO#	t7	Out Delay	4	31	3	27	$C_L = 25pf$**	
PEREQ	t7	Out Delay	5	34	5	34	$C_L = 75pf$	
BUSY#	t7	Out Delay	5	34	5	29	$C_L = 75pf$	
ERROR#	t7	Out Delay	5	34	5	34	$C_L = 75pf$	
D15–D0	t8	Out Delay	1	54	1	54	$C_L = 120pf$	6.4
D15–D0	t10	Setup Time	11		11			
D15–D0	t11	Hold Time	11		11			
D15–D0	t12*	Float Time	6	33	6	27	$C_L = 120pf$	
PEREQ	t13*	Float Time	1	60	1	50	$C_L = 75pf$	6.6
BUSY#	t13*	Float Time	1	60	1	50	$C_L = 75pf$	
ERROR#	t13*	Float Time	1	60	1	50	$C_L = 75pf$	
READYO#	t13*	Float Time	1	60	1	50	$C_L = 75pf$	
ADS#	t14	Setup Time	26		21			6.4
ADS#	t15	Hold Time	4		4			
W/R#	t14	Setup Time	26		21			
W/R#	t15	Hold Time	4		4			
READY#	t16	Setup Time	19		12			6.4
READY#	t17	Hold Time	4		4			
CMD0#	t16	Setup Time	21		19			
CMD0#	t17	Hold Time	2		2			
NPS1#, NPS2	t16	Setup Time	21		19			
NPS1#, NPS2	t17	Hold Time	2		2			
STEN	t16	Setup Time	21		21			
STEN	t17	Hold Time	2		2			

Table 6-2c. Timing Requirements of Bus Interface Unit T_C = 0° to 100° C, V_{CC} = 5V ± 10% (Continued)

Pin	Symbol	Parameter	16 MHz (1.5V)		20 MHz (1.5V)		Test Conditions	Refer to Figure
			Min (ns)	Max (ns)	Min (ns)	Max (ns)		
RESETIN	t18	Setup Time	13		11			6.5
RESETIN	t19	Hold Time	3		3			

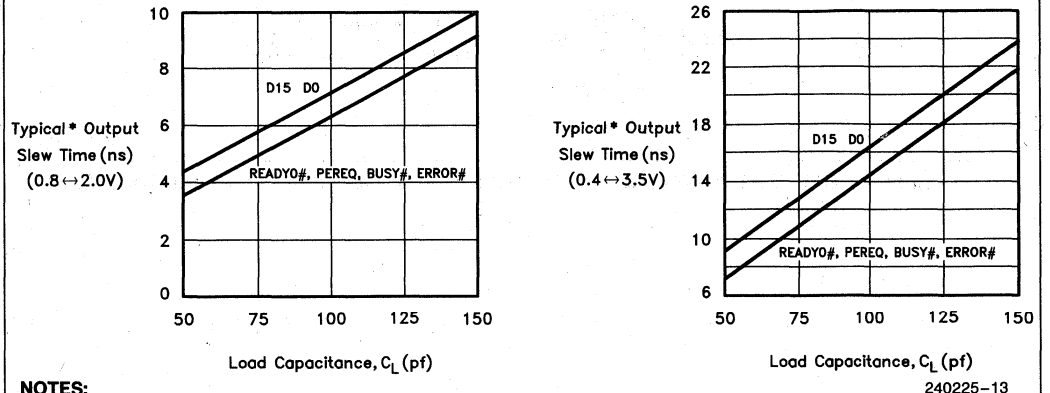
NOTES:
*Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not tested.
**Not tested at 25 pf.



NOTES:
Graphs are not linear outside the C_L range shown.
nom = nominal value given in the AC timing table
*Typical part under worst-case conditions

240225-12

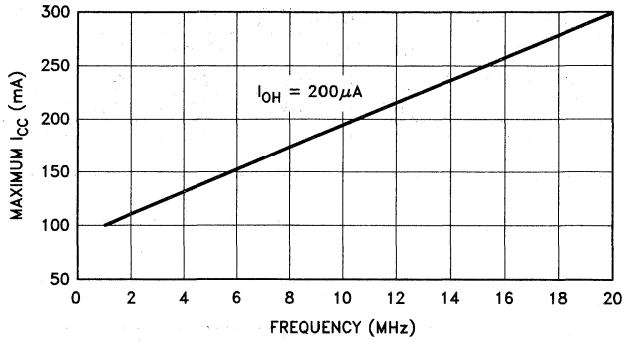
Figure 6-1a. Typical Output Valid Delay vs. Load Capacitance at Max Operating Temperature



NOTES:
Graphs are not linear outside the C_L range shown.
*Typical part under worst-case conditions

240225-13

Figure 6-1b. Typical Output Slew Time vs. Load Capacitance at Max Operating Temperature



NOTES:
Graphs are not linear outside the frequency range shown.

Figure 6-1c. Maximum I_{CC} vs. Frequency

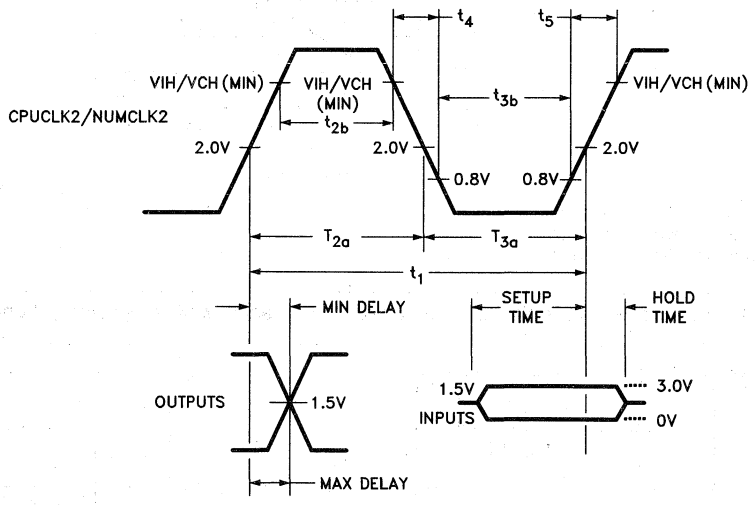


Figure 6-2. CPUCLK2/NUMCLK2 Waveform and Measurement Points for Input/Output

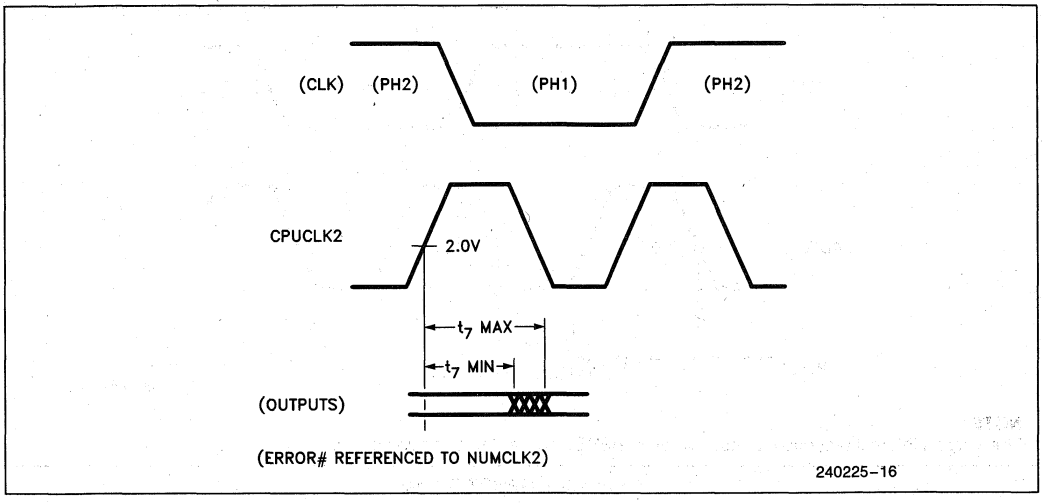


Figure 6-3. Output Signals

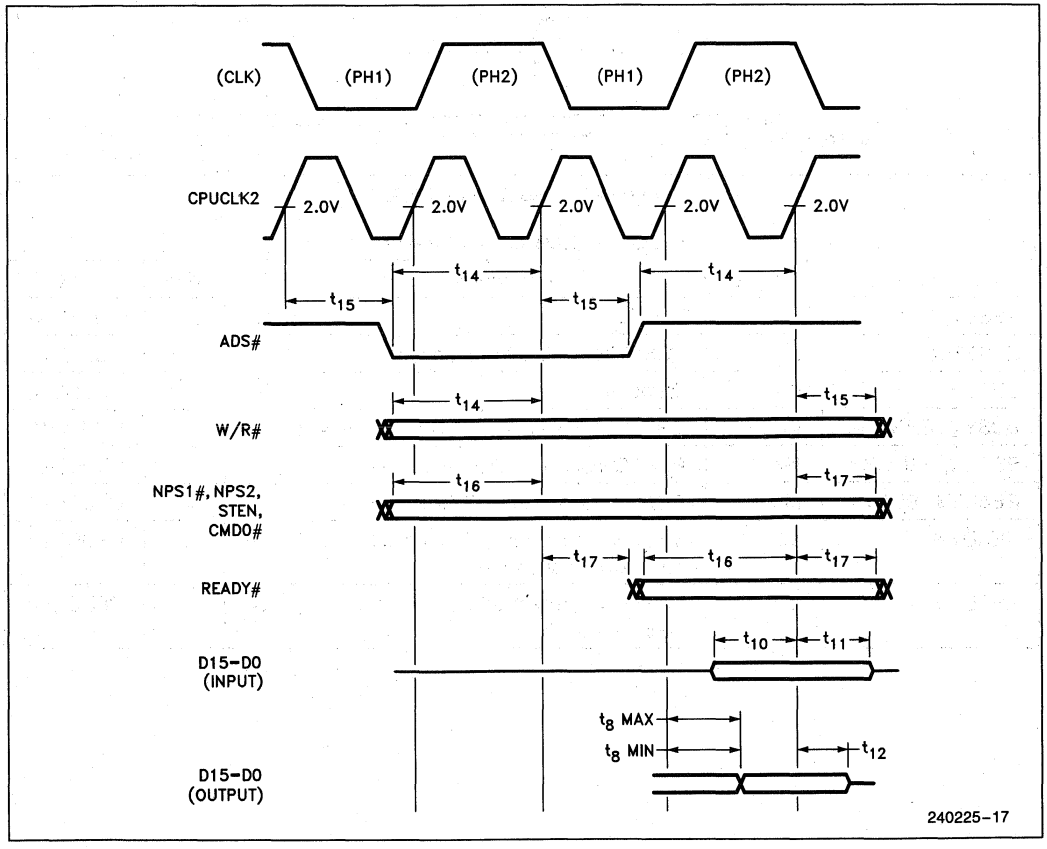
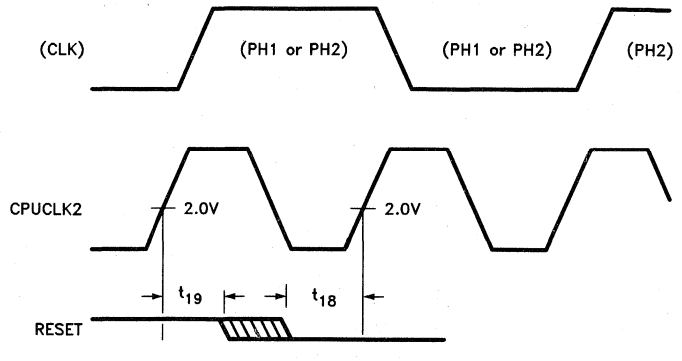


Figure 6-4. Input and I/O Signals

5



NOTE:
The second internal processor phase following RESET high to low transition is PH2.

Figure 6-5. RESET Signal

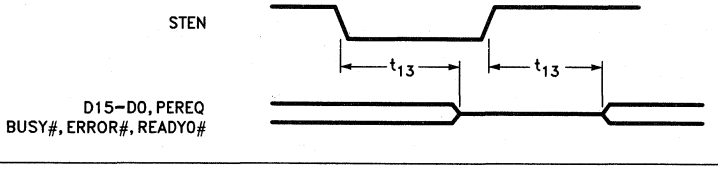
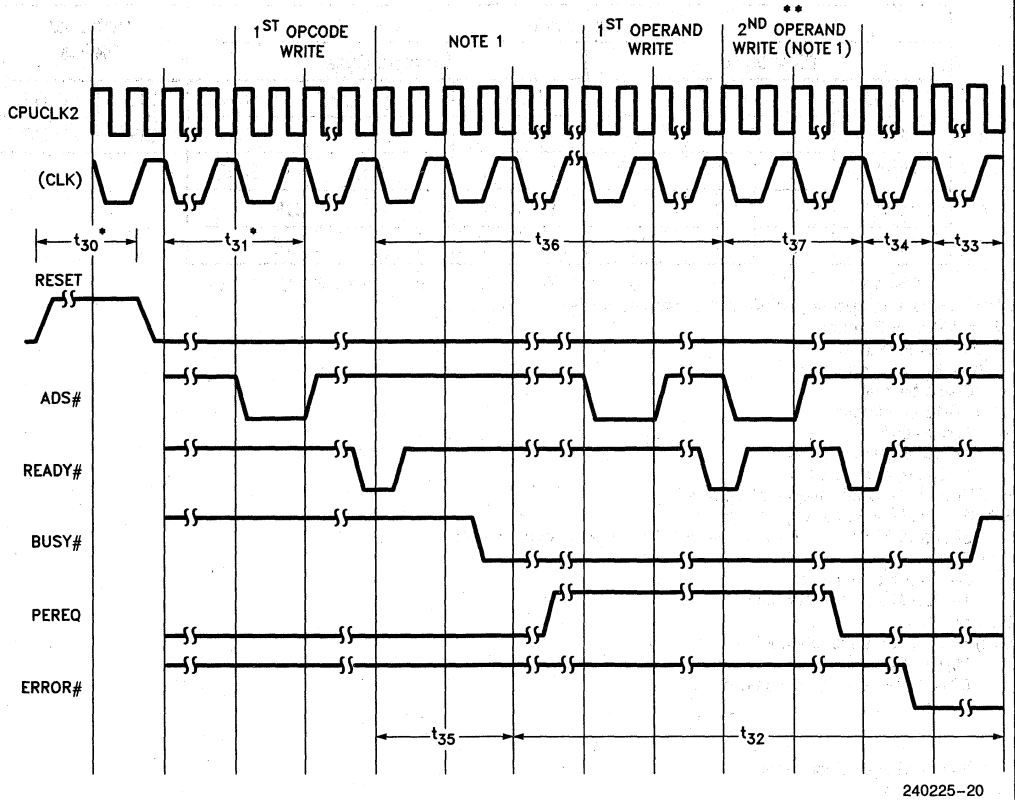


Figure 6-6. Float from STEN

Table 6-3. Other Parameters

Pin	Symbol	Parameter	Min	Max	Units
RESETIN	t30	Duration	40		NUMCLK2
RESETIN	t31	RESETIN inactive to 1st opcode write	50		NUMCLK2
BUSY#	t32	Duration	6		CPUCLK2
BUSY#, ERROR#	t33	ERROR# (in)active to BUSY# inactive	6		CPUCLK2
PEREQ, ERROR#	t34	PEREQ inactive to ERROR# active	6		CPUCLK2
READY#, BUSY#	t35	READY# active to BUSY# active	4	4	CPUCLK2
READY#	t36	Minimum time from opcode write to opcode/operand write	4		CPUCLK2
READY#	t37	Minimum time from operand write to operand write	4		CPUCLK2



* In NUMCLK2's
** or last operand

NOTE:
1. Memory read (operand) cycle is not shown.

Figure 6-7. Other Parameters

7.0 387™ SX NPX EXTENSIONS TO THE CPU'S INSTRUCTION SET

Instructions for the 387 SX NPX assume one of the five forms shown in Table 7-1. In all cases, instructions are at least two bytes long and begin with the bit pattern 11011B, which identifies the ESCAPE class of instruction. Instructions that refer to memory operands specify addresses using the CPU's addressing modes.

MOD (Mode field) and R/M (Register/Memory specifier) have the same interpretation as the corresponding fields of CPU instructions (refer to Programmer's Reference Manual for the CPU). SIB (Scale Index Base) byte and DISP (displacement) are optionally present in instructions that have MOD and R/M fields. Their presence depends on the values of MOD and R/M, as for instructions of the CPU.

The instruction summaries that follow assume that the instruction has been prefetched, decoded, and is ready for execution; that bus cycles do not require wait states; that there are no local bus HOLD requests delaying processor access to the bus; and that no exceptions are detected during instruction execution. If the instruction has MOD and R/M fields that call for both base and index registers, add one clock.

Table 7-1. Instruction Formats

	Instruction										Optional Fields	
	First Byte				Second Byte							
1	11011	OPA		1	MOD		1	OPB		R/M	SIB	DISP
2	11011	MF		OPA	MOD		OPB*			R/M	SIB	DISP
3	11011	d	P	OPA	1	1	OPB*			ST(i)		
4	11011	0	0	1	1	1	1	OP				
5	11011	0	1	1	1	1	1	OP				
	15–11	10	9	8	7	6	5	4	3	2 1 0		

OP = Instruction opcode, possibly split into two fields OPA and OPB
MF = Memory Format
  00—32-bit real
  01—32-bit integer
  10—64-bit real
  11—16-bit integer
d = Destination
  0—Destination is ST(0)
  1—Destination is ST(i)
R XOR d = 0—Destination (op) Source
R XOR d = 1—Source (op) Destination
*In FSUB and FDIV, the low-order bit of OPB is the R (reversed) bit
P = POP
  0—Do not pop stack
  1—Pop stack after operation
ESC = 11011
ST(i) = Register stack element i
  000 = Stack top
  001 = Second stack element
  .
  .
  .
  111 = Eighth stack element

387™ SX NPX Extension to the 386™ SX Microprocessor Instruction Set

Instruction	Encoding			Clock Count Range			
	Byte 0	Byte 1	Optional Bytes 2-6	32-Bit Real	32-Bit Integer	64-Bit Real	16-Bit Integer
DATA TRANSFER							
FLD = Load[a]							
Integer/real memory to ST(0)	ESC MF 1	MOD 000 R/M	SIB/DISP	24	49-56	33	61-65
Long integer memory to ST(0)	ESC 111	MOD 101 R/M	SIB/DISP		64-75		
Extended real memory to ST(0)	ESC 011	MOD 101 R/M	SIB/DISP		52		
BCD memory to ST(0)	ESC 111	MOD 100 R/M	SIB/DISP		274-283		
ST(i) to ST(0)	ESC 001	11000 ST(i)			14		
FST = Store							
ST(0) to integer/real memory	ESC MF 1	MOD 010 R/M	SIB/DISP	49	84-98	55	82-95
ST(0) to ST(i)	ESC 101	11010 ST(i)			11		
FSTP = Store and Pop							
ST(0) to integer/real memory	ESC MF 1	MOD 011 R/M	SIB/DISP	49	84-98	55	82-95
ST(0) to long integer memory	ESC 111	MOD 111 R/M	SIB/DISP		90-107		
ST(0) to extended real	ESC 011	MOD 111 R/M	SIB/DISP		63		
ST(0) to BCD memory	ESC 111	MOD 110 R/M	SIB/DISP		522-544		
ST(0) to ST(i)	ESC 101	11011 ST (i)			12		
FXCH = Exchange							
ST(i) and ST(0)	ESC 001	11001 ST(i)			18		
COMPARISON							
FCOM = Compare							
Integer/real memory to ST(0)	ESC MF 0	MOD 010 R/M	SIB/DISP	30	60-67	39	71-75
ST(i) to ST(0)	ESC 000	11010 ST(i)			24		
FCOMP = Compare and pop							
Integer/real memory to ST	ESC MF 0	MOD 011 R/M	SIB/DISP	30	60-67	39	71-75
ST(i) to ST(0)	ESC 000	11011 ST(i)			26		
FCOMPP = Compare and pop twice							
ST(1) to ST(0)	ESC 110	1101 1001			26		
FTST = Test ST(0)	ESC 001	1110 0100			28		
FUCOM = Unordered compare	ESC 101	11100 ST(i)			24		
FUCOMP = Unordered compare and pop	ESC 101	11101 ST(i)			26		
FUCOMPP = Unordered compare and pop twice	ESC 010	1110 1001			26		
FXAM = Examine ST(0)	ESC 001	11100101			30-38		
CONSTANTS							
FLDZ = Load +0.0 into ST(0)	ESC 001	1110 1110			20		
FLD1 = Load +1.0 into ST(0)	ESC 001	1110 1000			24		
FLDPI = Load pi into ST(0)	ESC 001	1110 1011			40		
FLDL2T = Load $\log_2(10)$ into ST(0)	ESC 001	1110 1001			40		

Shaded areas indicate instructions not available in 8087/80287.

NOTE:
a. When loading single- or double-precision zero from memory, add 5 clocks.

5

387™ SX NPX Extension to the 386™ SX Microprocessor Instruction Set (Continued)

Instruction	Encoding			Clock Count Range			
	Byte 0	Byte 1	Optional Bytes 2-6	32-Bit Real	32-Bit Integer	64-Bit Real	16-Bit Integer
CONSTANTS (Continued)							
FLDL2E = Load $\log_2(e)$ into ST(0)	ESC 001	1110 1010			40		
FLDLG2 = Load $\log_{10}(2)$ into ST(0)	ESC 001	1110 1100			41		
FLDLN2 = Load $\log_e(2)$ into ST(0)	ESC 001	1110 1101			41		
ARITHMETIC							
FADD = Add							
Integer/real memory with ST(0)	ESC MF 0	MOD 000 R/M	SIB/DISP	28-36	61-76	37-45	71-85
ST(i) and ST(0)	ESC d P 0	11000 ST(i)			23-31[b]		
FSUB = Subtract							
Integer/real memory with ST(0)	ESC MF 0	MOD 10 R R/M	SIB/DISP	28-36	61-76	36-44	71-83[c]
ST(i) and ST(0)	ESC d P 0	1110 R R/M			26-34[d]		
FMUL = Multiply							
Integer/real memory with ST(0)	ESC MF 0	MOD 001 R/M	SIB/DISP	31-39	65-86	40-65	76-87
ST(i) and ST(0)	ESC d P 0	1100 1 R/M			29-57[e]		
FDIV = Divide							
Integer/real memory with ST(0)	ESC MF 0	MOD 11 R R/M	SIB/DISP	93	124-131[f]	102	136-140[g]
ST(i) and ST(0)	ESC d P 0	1111 R R/M			88[h]		
FSQRT[i] = Square root	ESC 001	1111 1010			122-129		
FSCALE = Scale ST(0) by ST(1)	ESC 001	1111 1101			67-86		
FPREM = Partial remainder	ESC 001	1111 1000			74-155		
FPREM1 = Partial remainder (IEEE)	ESC 001	1111 0101			95-185		
FRNDINT = Round ST(0) to integer	ESC 001	1111 1100			66-80		
FXTRACT = Extract components of ST(0)	ESC 001	1111 0100			70-76		
FABS = Absolute value of ST(0)	ESC 001	1110 0001			22		
FCHS = Change sign of ST(0)	ESC 001	1110 0000			24-25		

Shaded areas indicate instructions not available in 8087/80287.

NOTES:
b. Add 3 clocks to the range when d = 1.
c. Add 1 clock to **each** range when R = 1.
d. Add 3 clocks to the range when d = 0.
e. typical = 52 (When d = 0, 46-54, typical = 49).
f. Add 1 clock to the range when R = 1.
g. 135-141 when R = 1.
h. Add 3 clocks to the range when d = 1.
i. $-0 \leq ST(0) \leq +\infty$.

387™ SX NPX Extension to the 386™ SX Microprocessor Instruction Set (Continued)

Instruction	Encoding			Clock Count Range
	Byte 0	Byte 1	Optional Bytes 2-6	
TRANSCENDENTAL				
FCOS[k] = Cosine of ST(0)	ESC 001	1111 1111		123-772[j]
FPTAN[k] = Partial tangent of ST(0)	ESC 001	1111 0010		191-497[j]
FPATAN = Partial arctangent	ESC 001	1111 0011		314-487
FSIN[k] = Sine of ST(0)	ESC 001	1111 1110		122-771[j]
FSINCOS[k] = Sine and cosine of ST(0)	ESC 001	1111 1011		194-809[j]
F2XM1[l] = $2^{ST(0)} - 1$	ESC 001	1111 0000		211-476
FYL2X[m] = ST(1) * $\log_2$(ST(0))	ESC 001	1111 0001		120-538
FYL2XP1[n] = ST(1) * $\log_2$(ST(0) + 1.0)	ESC 001	1111 1001		257-547
PROCESSOR CONTROL				
FINIT = Initialize NPX	ESC 011	1110 0011		33
FSTSW AX = Store status word	ESC 111	1110 0000		13
FLDCW = Load control word	ESC 001	MOD 101 R/M	SIB/DISP	19
FSTCW = Store control word	ESC 101	MOD 111 R/M	SIB/DISP	15
FSTSW = Store status word	ESC 101	MOD 111 R/M	SIB/DISP	15
FCLEX = Clear exceptions	ESC 011	1110 0010		11
FSTENV = Store environment	ESC 001	MOD 110 R/M	SIB/DISP	103-104
FLDENV = Load environment	ESC 001	MOD 100 R/M	SIB/DISP	71
FSAVE = Save state	ESC 101	MOD 110 R/M	SIB/DISP	475-476
FRSTOR = Restore state	ESC 101	MOD 100 R/M	SIB/DISP	388
FINCSTP = Increment stack pointer	ESC 001	1111 0111		21
FDECSTP = Decrement stack pointer	ESC 001	1111 0110		22
FFREE = Free ST(i)	ESC 101	1100 0 ST(i)		18
FNOP = No operations	ESC 001	1101 0000		12

Shaded areas indicate instructions not available in 8087/80287.

NOTES:
j. These timings hold for operands in the range $|x| < \pi/4$. For operands not in this range, up to 76 additional clocks may be needed to reduce the operand.
k. $0 \leq |ST(0)| < 2^{63}$.
l. $-1.0 \leq ST(0) \leq 1.0$.
m. $0 \leq ST(0) < \infty$, $-\infty < ST(1) < +\infty$.
n. $0 \leq |ST(0)| < (2 - SQRT(2))/2$, $-\infty < ST(1) < +\infty$.

5

APPENDIX A
COMPATIBILITY BETWEEN THE 80287 AND THE 8087

The 80286/80287 operating in Real-Address mode will execute 8086/8087 programs without major modification. However, because of differences in the handling of numeric exceptions by the 80287 NPX and the 8087 NPX, exception-handling routines *may* need to be changed.

This appendix summarizes the differences between the 80287 NPX and the 8087 NPX, and provides details showing how 8086/8087 programs can be ported to the 80286/80287.

1. The NPX signals exceptions through a dedicated ERROR# line to the 80286. The NPX error signal does not pass through an interrupt controller (the 8087 INT signal does). Therefore, any interrupt-controller-oriented instructions in numeric exception handlers for the 8086/8087 should be deleted.
2. The 8087 instructions FENI/FNENI and FDISI/FNDISI perform no useful function in the 80287. If the 80287 encounters one of these opcodes in its instruction stream, the instruction will effectively be ignored—none of the 80287 internal states will be updated. While 8086/8087 containing these instructions may be executed on the 80286/80287, it is unlikely that the exception-handling routines containing these instructions will be completely portable to the 80287.
3. Interrupt vector 16 must point to the numeric exception handling routine.
4. The ESC instruction address saved in the 80287 includes any leading prefixes before the ESC opcode. The corresponding address saved in the 8087 does not include leading prefixes.
5. In Protected-Address mode, the format of the 80287's saved instruction and address pointers is different than for the 8087. The instruction opcode is not saved in Protected mode—exception handlers will have to retrieve the opcode from memory if needed.
6. Interrupt 7 will occur in the 80286 when executing ESC instructions with either TS (task switched) or EM (emulation) of the 80286 MSW set (TS = 1 or EM = 1). If TS is set, then a WAIT instruction will also cause interrupt 7. An exception handler should be included in 80286/80287 code to handle these situations.
7. Interrupt 9 will occur if the second or subsequent words of a floating-point operand fall outside a segment's size. Interrupt 13 will occur if the starting address of a numeric operand falls outside a segment's size. An exception handler should be included in 80286/80287 code to report these programming errors.
8. Except for the processor control instructions, all of the 80287 numeric instructions are automatically synchronized by the 80286 CPU—the 80286 automatically tests the BUSY# line from the 80287 to ensure that the 80287 has completed its previous instruction before executing the next ESC instruction. No explicit WAIT instructions are required to assure this synchronization. For the 8087 used with 8086 and 8088 processors, explicit WAITs are required before each numeric instruction to ensure synchronization. Although 8086/8087 programs having explicit WAIT instructions will execute perfectly on the 80286/80287 without reassembly, these WAIT instructions are unnecessary.
9. Since the 80287 does not require WAIT instructions before each numeric instruction, the ASM286 assembler does not automatically generate these WAIT instructions. The ASM86 assembler, however, automatically precedes every ESC instruction with a WAIT instruction. Although numeric routines generated using the ASM86 assembler will generally execute correctly on the 80286/80287, reassembly using ASM286 may result in a more compact code image.

   The processor control instructions for the 80287 may be coded using either a WAIT or No-WAIT form of mnemonic. The WAIT forms of these instructions cause ASM286 to precede the ESC instruction with a CPU WAIT instruction, in the identical manner as does ASM86.

DATA SHEET REVISION REVIEW

The following list represents the key differences between this and the -004 versions of the 387 SX Math CoProcessor Data Sheet. Please review this summary carefully.

1. Added 20 MHz timing specs. Improved HOLD times for ADS#, W/R#, RESETIN.

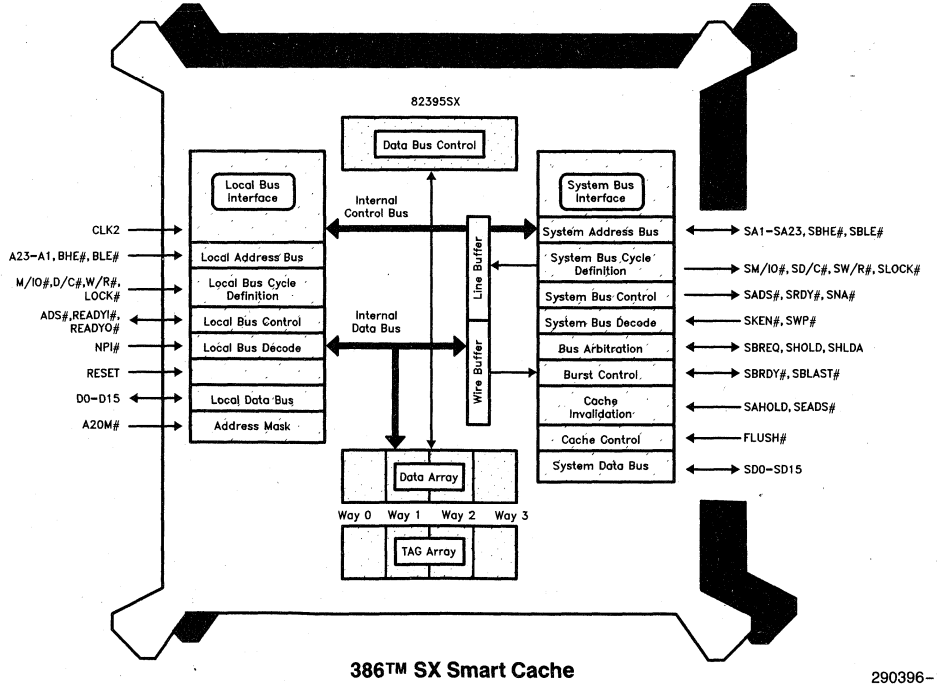
ADVANCE INFORMATION

386™ SX SMART CACHE
82395SX

- **Optimized 386 SX Microprocessor Companion**
- **Integrated 8KB Data RAM**
- **4 Way SET Associative with Pseudo LRU Algorithm**
- **Write Buffer Architecture**
- **Integrated 4 Word Write Buffer**
- **8 Byte Line Size**
- **Integrated 387™ Math Coprocessor Decode Logic**
- **Concurrent Line Buffer Cacheing**
- **Supports i486™ Microprocessor-like Burst**
- **Dual Bus Architecture**
  — Snooping Maintains Cache Coherency
- **20 MHz Clock**
- **132 Lead PQFP Package**
- **1K Tag Entries**
- **Non-Sectored Architecture**

The 386 SX Smart Cache (part number 82395SX) is a low cost, single chip, 16-bit peripheral for Intel's 386 SX Microprocessor. By storing frequently accessed codes or data from main memory, the 386 SX Smart Cache enables the 386 SX Microprocessor to run at near zero wait states. The dual bus architecture allows another bus master to access the System Bus while the 386 SX Microprocessor operates out of the 386 SX Smart Cache on the Local Bus. The 386 SX Smart Cache has a snooping mechanism which maintains cache coherency with main memory during these cycles.

The 386 SX Smart Cache is completely software transparent, protecting the integrity of system software. The advanced architectural features of the 386 SX Smart Cache offer high performance with a cache data RAM size that can be integrated on a single chip, offering the board space and cost savings needed in 386 SX Microprocessor based systems.



386™ SX Smart Cache

290396-1

387™ SX, 386™ SX, and i486™ are trademarks of Intel Corporation.

October 1990
Order Number: 290396-001

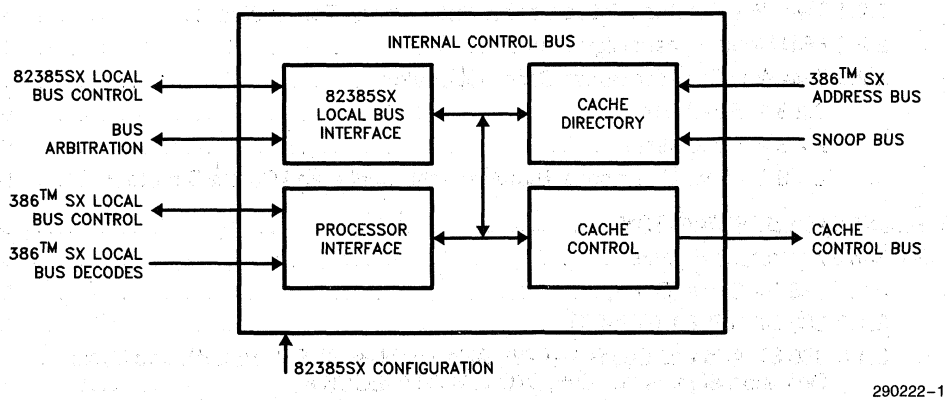
82385SX
HIGH PERFORMANCE CACHE CONTROLLER

- **Improves 386™ SX System Performance**
  - **— Reduces Average CPU Wait States to Nearly Zero**
  - **— Zero Wait State Read Hit**
  - **— Zero Wait State Posted Memory Writes**
  - **— Allows Other Masters to Access the System Bus More Readily**
- **Hit Rates up to 99%**
- **Optimized as 386 SX Companion**
  - **— Simple 386 SX Interface**
  - **— Part of Intel386™-Based Compute Engine Including 387™ SX Math Coprocessor and 82370 Integrated System Peripheral**
  - **— 16 MHz and 20 MHz Operation**
- **Software Transparent**
- **Synchronous Dual Bus Architecture**
  - **— Bus Watching Maintains Cache Coherency**
- **Maps Full 386 SX Address Space**
- **Flexible Cache Mapping Policies**
  - **— Direct Mapped or 2-Way Set Associative Cache Organization**
  - **— Supports Non-Cacheable Memory Space**
  - **— Unified Cache for Code and Data**
- **Integrates Cache Directory and Cache Management Logic**
- **High Speed CHMOS Technology**
  - **— 132-Pin PGA and 132-Lead PQFP**

The 82385SX Cache Controller is a high performance peripheral for Intel's 386™ SX Microprocessor. It stores a copy of frequently accessed code and data from main memory in a zero wait state local cache memory. The 82385SX allows the 386 SX Microprocessor to run near its full potential by reducing the average number of CPU wait states to nearly zero. The dual bus architecture of the 82385SX allows other masters to access system resources while the 386 SX CPU operates locally out of its cache. In this situation, the 82385SX's "bus watching" mechanism preserves cache coherency by monitoring the system bus address lines at no cost to system or local throughput.

The 82385SX is completely software transparent, protecting the integrity of system software. High performance and board space savings are achieved because the 82385SX integrates a cache directory and all cache management logic on one chip.

5



290222-1

82385SX Internal Block Diagram

Intel386™, 386™, 386™ SX, and 387™ SX are trademarks of Intel Corporation.

October 1990
Order Number: 290222-003

CONTENTS

PAGE

1.0 82385SX FUNCTIONAL OVERVIEW 5-1008
1.1 82385SX Overview 5-1008
1.2 System Overview I: Bus Structure 5-1008
1.2.1 386™ SX Local Bus/82358SX Local Bus/System Bus 5-1008
1.2.2 Bus Arbitration 5-1008
1.2.3 Master/Slave Operation 5-1011
1.2.4 Cache Coherency 5-1011
1.3 System Overview II: Basic Operation 5-1012
1.3.1 386 SX Memory Code and Data Read Cycles 5-1012
1.3.1.1 Read Hits 5-1012
1.3.1.2 Read Misses 5-1012
1.3.2 386 SX Memory Write Cycles 5-1012
1.3.3 Non-Cacheable Cycles 5-1012
1.3.4 386 SX Local Bus Cycles 5-1013
1.3.5 Summary of 82385SX Response to All 386 SX Cycles 5-1013
1.3.6 Bus Watching 5-1013
1.3.7 Cache Flush 5-1013

2.0 82385SX CACHE ORGANIZATION 5-1015
2.1 Direct Mapped Cache 5-1015
2.1.1 Direct Mapped Cache Structure and Terminology 5-1015
2.1.2 Direct Mapped Cache Operation 5-1016
2.1.2.1 Read Hits 5-1016
2.1.2.2 Read Misses 5-1016
2.1.2.3 Other Operations That Affect the Cache and Cache Directory 5-1016
2.2 Two Way Set Associative Cache 5-1017
2.2.1 Two Way Set Associative Cache Structure and Terminology 5-1017
2.2.2 LRU Replacement Algorithm 5-1017
2.2.3 Two Way Set Associative Cache Operation 5-1017
2.2.3.1 Read Hits 5-1017
2.2.3.2 Read Misses 5-1017
2.2.3.3 Other Operations That Affect the Cache and Cache Directory 5-1017

3.0 82835SX PIN DESCRIPTION 5-1017
3.1 386 SX Interface Signals 5-1019
3.1.1 386 SX Clock (CLK2) 5-1019
3.1.2 386 SX RESET (RESET) 5-1019
3.1.3 386 SX Address Bus (A1–A23), Byte Enables (BHE#, BLE#), and Cycle Definition Signals (M/IO#, D/C#, W/R#, LOCK#) 5-1020
3.1.4 386 SX Address Status (ADS#) and Ready Input (READYI#) 5-1020
3.1.5 386 SX Next Address Request (NA#) 5-1020
3.1.6 Ready Output (READYO#) and Bus Ready Enable (BRDYEN#) 5-1020

CONTENTS

PAGE

3.2 Cache Control Signals 5-1020
3.2.1 Cache Address Latch Enable (CALEN) 5-1020
3.2.2 Cache Transmit/Receive (CT/R#) 5-1020
3.2.3 Cache Chip Selects (CS0#–CS1#) 5-1021
3.2.4 Cache Output Enables (COEA#, COEB#) and Write Enables (CWEA#, CWEB#) 5-1021
3.3 386 SX Local Bus Decode Inputs 5-1021
3.3.1 386 SX Local Bus Access (LBA#) 5-1021
3.3.2 Non-Cacheable Access (NCA#) 5-1021
3.4 82385SX Bus Interface Signals 5-1021
3.4.1 82385SX Bus Byte Enables (BBHE#, BBLE#) 5-1022
3.4.2 82385SX Bus Lock (BLOCK#) 5-1022
3.4.3 82385SX Bus Address Status (BADS#) 5-1022
3.4.4 82385SX Bus Ready Input (BREADY#) 5-1022
3.4.5 82385SX Bus Next Address Request (BNA#) 5-1022
3.5 82385SX Bus Data Transceiver and Address Latch Control 5-1022
3.5.1 Local Data Strobe (LDSTB), Data Output Enable (DOE#), and Bus Transmit/Receive (BT/R#) 5-1022
3.5.2 Bus Address Clock Pulse (BACP) and Bus Address Output Enable (BAOE#) 5-1022
3.6 Status and Control Signals 5-1022
3.6.1 Cache Miss Indication (MISS#) 5-1022
3.6.2 Write Buffer Status (WBS) 5-1023
3.6.3 Cache Flush (FLUSH) 5-1023
3.7 Bus Arbitration Signals (BHOLD and BHLDA) 5-1023
3.8 Coherency (Bus Watching) Support Signals (SA1–SA23, SSTB#, SEN) 5-1023
3.9 Configuration Inputs (2W/D#, M/S#, DEFOE#) 5-1023
3.10 Reserved Pins (RES) 5-1023

4.0 386 SX LOCAL BUS INTERFACE 5-1024
4.1 Processor Interface 5-1024
4.1.1 Hardware Interface 5-1024
4.1.2 Ready Generation 5-1026
4.1.3 NA# and 386 SX Local Bus Pipelining 5-1026
4.1.4 LBA# and NCA# Generation 5-1029
4.2 Cache Interface 5-1029
4.2.1 Cache Configurations 5-1029
4.2.2 Cache Control . . . Direct Mapped 5-1032
4.2.3 Cache Control . . . Two Way Set Associative 5-1035
4.3 387 Interface 5-1035

5

CONTENTS

PAGE

5.0 82385SX LOCAL BUS AND SYSTEM INTERFACE 5-1035
5.1 The 823885SX Bus State Machine 5-1035
5.1.1 Master Mode 5-1035
5.1.2 Slave Mode 5-1035
5.2 The 82385SX Local Bus 5-1040
5.2.1 82385SX Bus Counterparts to 386 SX Signals 5-1048
5.2.1.1 Address Bus (BA1–BA23) and Cycle Definition Signals (BM/IO#, BD/C#, BW/R#) 5-1048
5.2.1.2 Data Bus (BD0–BD15) 5-1049
5.2.1.3 Byte Enables (BBHE#, BBLE#) 5-1049
5.2.1.4 Address Status (BADS#) 5-1049
5.2.1.5 Ready (BREADY#) 5-1049
5.2.1.6 Next Address Request (BNA#) 5-1049
5.2.1.7 Bus Lock (BLOCK#) 5-1049
5.2.2 Additional 82385SX Bus Signals 5-1050
5.2.2.1 Cache Read/Write Miss Indication (MISS#) 5-1050
5.2.2.2 Write Buffer Status (WBS) 5-1051
5.2.2.3 Cache Flush (FLUSH) 5-1051
5.3 Bus Watching (Snoop) Interface 5-1051
5.4 Reset Definition 5-1052

6.0 SYSTEM DESIGN GUIDELINES 5-1053
6.1 Introduction 5-1053
6.2 Power and Grounding 5-1053
6.2.1 Power Connections 5-1053
6.2.2 Power Decoupling 5-1053
6.2.3 Resistor Recommendations 5-1053
6.2.3.1 386 SX Local Bus 5-1053
6.2.3.2 82385SX Local Bus 5-1053
6.3 82385SX Signal Connections 5-1053
6.3.1 Configuration Inputs 5-1053
6.3.2 CLK2 and RESET 5-1054
6.4 Unused Pin Requirements 5-1054
6.5 Cache SRAM Requirements 5-1054
6.5.1 Cache Memory without Transceivers 5-1054
6.5.2 Cache Memory with Transceivers 5-1055

CONTENTS

PAGE

7.0 SYSTEM TEST CONSIDERATIONS 5-1055
7.1 Introduction 5-1055
7.2 Main Memory (DRAM) Testing 5-1055
7.2.1 Memory Testing Routine 5-1055
7.3 82385SX Cache Memory Testing 5-1055
7.3.1 Test Routine in the NCA# or LBA# Memory Map 5-1056
7.3.2 Test Routine in Cacheable Memory 5-1056
7.4 82385SX Cache Directory Testing 5-1056
7.5 Special Function Pins 5-1056

8.0 MECHANICAL DATA 5-1056
8.1 Introduction 5-1056
8.2 Pin Assignment 5-1056
8.3 Package Dimensions and Mounting 5-1061
8.4 Package Thermal Specification 5-1066

9.0 ELECTRICAL DATA 5-1066
9.1 Introduction 5-1066
9.2 Maximum Ratings 5-1066
9.3 D.C. Specifications 5-1066
9.4 A.C. Specifications 5-1067
9.4.1 Frequency Dependent Signals 5-1067

5

1.0 82385SX FUNCTIONAL OVERVIEW

The 82385SX Cache Controller is a high performance peripheral for Intel's 386™ SX microprocessor. This chapter provides an overview of the 82385SX, and of the basic architecture and operation of a 386 SX CPU/82385SX system.

1.1 82385SX Overview

The main function of a cache memory system is to provide fast local storage for frequently accessed code and data. The cache system intercepts 386 SX memory references to see if the required data resides in the cache. If the data resides in the cache (a hit), it is returned to the 386 SX without incurring wait states. If the data is not cached (a miss), the reference is forwarded to the system and the data retrieved from main memory. An efficient cache will yield a high "hit rate" (the ratio of cache hits to total 386 SX accesses), such that the majority of accesses are serviced with zero wait states. The net effect is that the wait states incurred in a relatively infrequent miss are averaged over a large number of accesses, resulting in an average of nearly zero wait states per access. Since cache hits are serviced locally, a processor operating out of its local cache has a much lower "bus utilization" which reduces system bus bandwidth requirements, making more bandwidth available to other bus masters.

The 82385SX Cache Controller integrates a cache directory and all cache management logic required to support an external 16 kbyte cache. The cache directory structure is such that the entire physical address range of the 386 SX is mapped into the cache. Provision is made to allow areas of memory to be set aside as non-cacheable. The user has two cache organization options: direct mapped and 2-way set associative. Both provide the high hit rates necessary to make a large, relatively slow main memory array look like fast, zero wait state memory to the 386 SX.

A good hit rate is an essential ingredient of a successful cache implementation. Hit rate is the measure of how efficient a cache is in maintaining a copy of the most frequently requested code and data. However, efficiency is not the only factor for performance consideration. Just as essential are sound cache management policies. These policies refer to the handling of 386 SX writes, preservation of cache coherency, and ease of system design. The 82385SX's "posted write" capability allows the majority of 386 SX writes, including most non-cacheable cycles, to run with zero wait states, and the 82385SX's "bus watching" mechanism preserves cache coherency with no impact on system performance. Physically, the 82385SX ties directly to the 386 SX with virtually no external logic.

1.2 System Overview I: Bus Structure

A good grasp of bus structure of a 386 SX CPU/82385SX system is essential in understanding both the 82385SX and its role in a 386 SX system. The following is a description of this structure.

1.2.1 386™ SX LOCAL BUS/82385SX LOCAL BUS/SYSTEM BUS

Figure 1-1 depicts the bus structure of a typical 386 SX system. The "386 SX Local Bus" consists of the physical 386 SX address, data, and control busses. The local address and data busses are buffered and/or latched to become the "system" address and data busses. The local control bus is decoded by bus control logic to generate the various system bus read and write commands.

The addition of an 82385SX Cache Controller causes a separation of the 386 SX bus into two distinct busses: the actual 386 SX local bus and the "82385SX Local Bus" (Figure 1-2). The 82385SX local bus is designed to look like the front end of a 386 SX by providing 82385SX local bus equivalents to all appropriate 386 SX signals. The system ties to this "386 SX-like" front end just as it would to an actual 386 SX. The 386 SX simply sees a fast system bus, and the system sees a 386 SX front end with low bus bandwidth requirements. The cache subsystem is transparent to both. Note that the 82385SX local bus is not simply a buffered version of the 386 SX bus, but rather is distinct from, and able to operate in parallel with the 386 SX bus. Other masters residing on either the 82385SX local bus or system bus are free to manage system resources while the 386 SX operates out of its cache.

1.2.2 BUS ARBITRATION

The 82385SX presents the "386 SX-like" interface which is called the 82385SX local bus. Whereas the 386 SX provides a Hold Request/ Hold Acknowledge bus arbitration mechanism via its HOLD and HLDA pins, the 82385SX provides an equivalent mechanism via its BHOLD and BHLDA pins. (These signals are described in Section 3.7.) When another master requests the 82385SX local bus, it issues the request to the 82385SX via BHOLD. Typically, at the end of the current 82385SX local bus cycle, the 82385SX will release the 82385SX local bus and acknowledge the request via BHLDA. The 386 SX is of course free to continue operating on the 386 SX local bus while another master owns the 82385SX local bus.

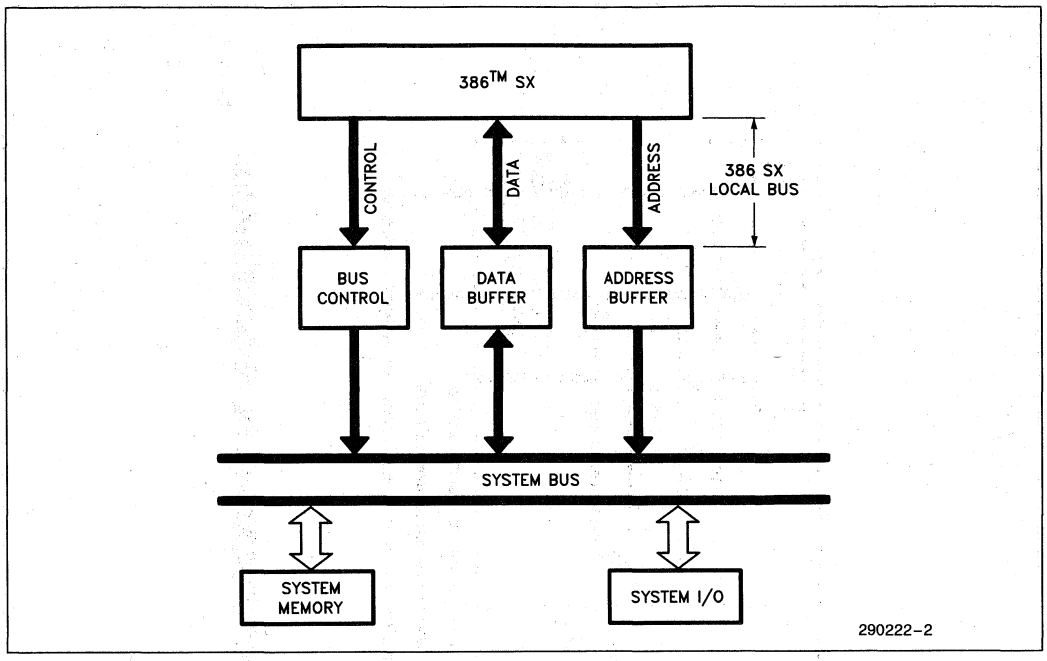


Figure 1-1. 386™ SX System Bus Structure

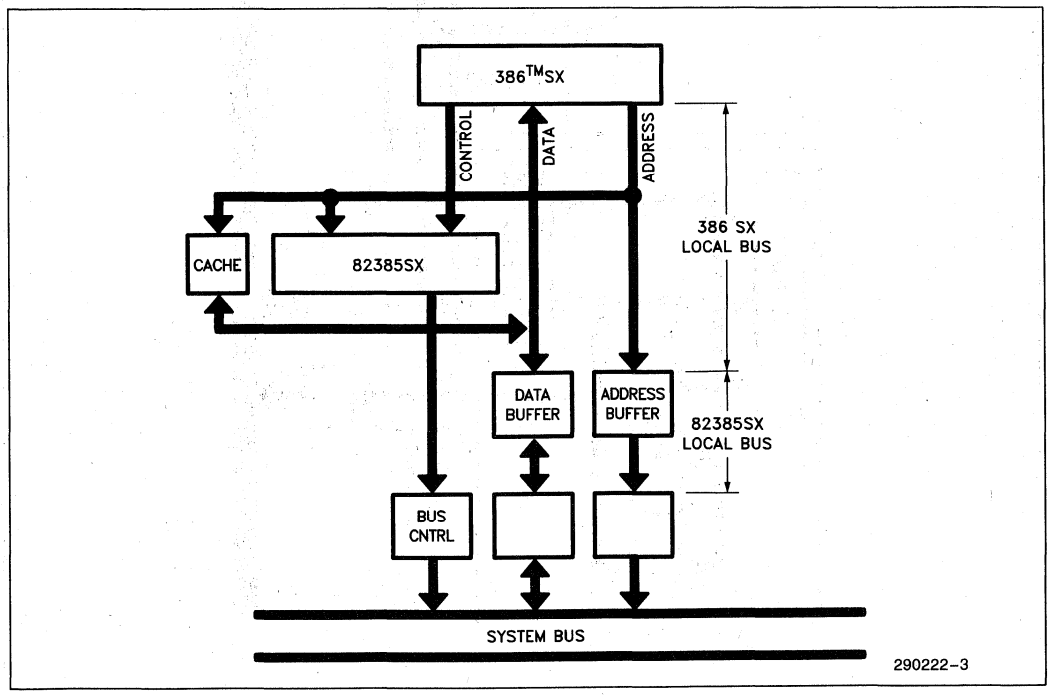


Figure 1-2. 386™ SX and 82385SX System Bus Structure

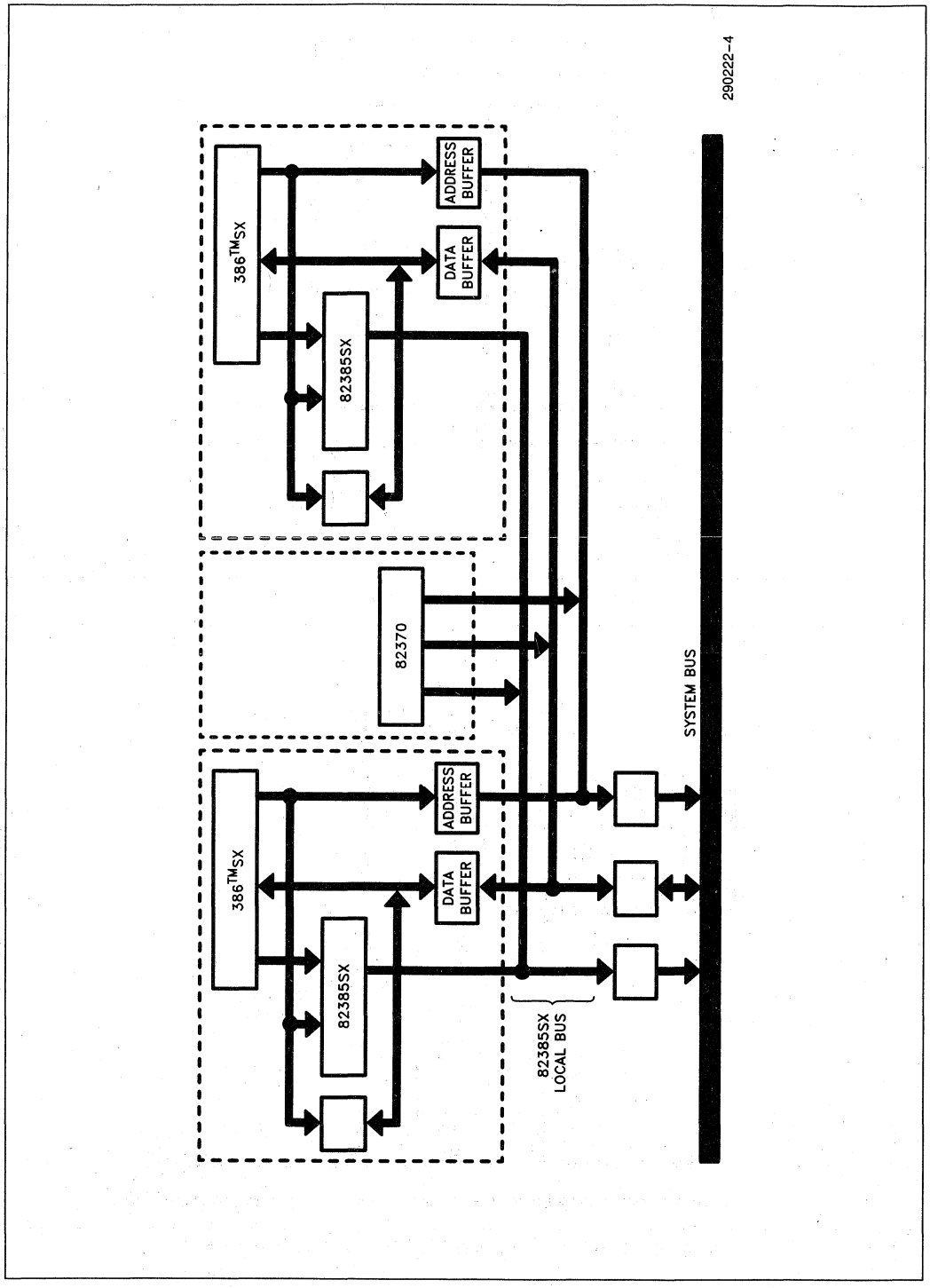


Figure 1-3. Multi-Master/Multi-Cache Environment

1.2.3 MASTER/SLAVE OPERATION

The above 82385SX local bus arbitration discussion is true when the 82385SX is programmed for "Master" mode operation. The user can, however, configure the 82385SX for "Slave" mode operation. (Programming is done via a hardware strap option.) The roles of BHOLD and BHLDA are reversed for an 82385SX in slave mode; BHOLD becomes an output indicating a request to control the bus, and BHLDA becomes an input indicating that a request has been granted. An 82385SX programmed in slave mode drives the 82385SX local bus only when it has requested and subsequently been granted bus control. This allows multiple 386 SX CPU/82385SX subsystems to reside on the same 82385SX local bus (Figure 1-3).

1.2.4 CACHE COHERENCY

Ideally, a cache contains a copy of the most heavily used portions of main memory. To maintain cache "coherency" is to make sure that this local copy is identical to main memory. In a system where multiple masters can access the same memory, there is always a risk that one master will alter the contents of a memory location that is duplicated in the local cache of another master. (The cache is said to contain "stale" data.) One rather restrictive solution is to not allow cache subsystems to cache shared memory. Another simple solution is to flush the cache anytime another master writes to system memory. However, this can seriously degrade system performance as excessive cache flushing will reduce the hit rate of what may otherwise be a highly efficient cache.

The 82385SX preserves cache coherency via "bus watching" (also called snooping), a technique that neither impacts performance nor restricts memory mapping. An 82385SX that is not currently bus master monitors system bus cycles, and when a write cycle by another master is detected (a snoop), the system address is sampled and used to see if the referenced location is duplicated in the cache. If so (a snoop hit), the corresponding cache entry is invalidated, which will force the 386 SX to fetch the up-to-date data from main memory the next time it accesses this modified location. Figure 1-4 depicts the general form of bus watching.

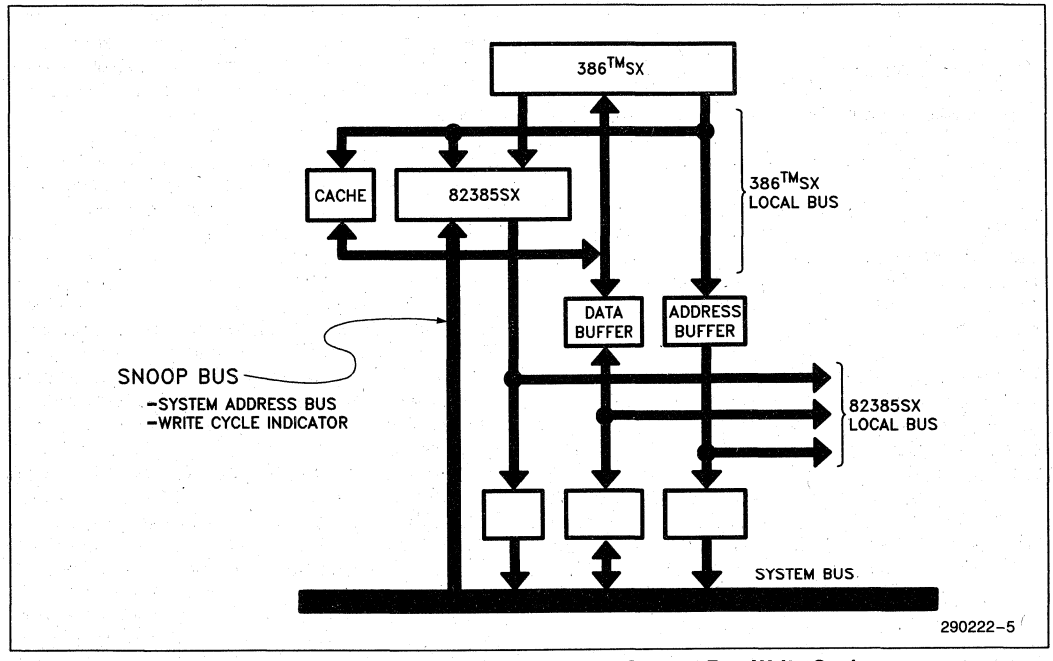


Figure 1-4. 82385SX Bus Watching—Monitor System Bus Write Cycles

1.3 System Overview II: Basic Operation

This discussion is an overview of the basic operation of a 386 SX CPU/82385SX system. Items discussed include the 82385SX's response to all 386 SX cycles, including interrupt acknowledges, halts, and shutdowns. Also discussed are non-cacheable and local accesses.

1.3.1 386™ SX MEMORY CODE AND DATA READ CYCLES

1.3.1.1 Read Hits

When the 386 SX initiates a memory code or data read cycle, the 82385SX compares the high order bits of the 386 SX address bus with the appropriate addresses (tags) stored in its on-chip directory. (The directory structure is described in Section 2.1.1) If the 82385SX determines that the requested data is in the cache, it issues the appropriate control signals that direct the cache to drive the requested data onto the 386 SX data bus, where it is read by the 386 SX. The 82385SX terminates the 386 SX cycle without inserting any wait states.

1.3.1.2 Read Misses

If the 82385SX determines that the requested data is not in the cache, the request is forwarded to the 82385SX local bus and the data retrieved from main memory. As the data returns from main memory, it is directed to the 386 SX and also written into the cache. Concurrently, the 82385SX updates the cache directory such that the next time this particular piece of information is requested by the 386 SX, the 82385SX will find it in the cache and return it with zero wait states.

The basic unit of transfer between main memory and cache memory in a cache subsystem is called the line size. In an 82385SX system, the line size is one 16-bit word. During a read miss, both 82385SX local bus byte enables are active. This insures that the 16-bit entry is written into the cache. (The 386 SX simply ignores what it did not request.) In any other type of 386 SX cycle that is forwarded to the 82385SX local bus, the logic levels of the 386 SX byte enables are duplicated on the 82385SX local bus.

The 82385SX does not actively fetch main memory data independently of the 386 SX. The 82385SX is essentially a passive device which only monitors the address bus and activates control signals. The read miss is the only mechanism by which main memory data is copied into the cache and validated in the cache directory.

In an isolated read miss, the number of wait states seen by the 386 SX is that required by the system memory to respond with data plus the cache comparison cycle (hit/miss decision). The cache system must determine that the cycle is a miss before it can begin the system memory access. However, since misses most often occur consecutively, the 82385SX will begin 386 SX address pipelined cycles to effectively "hide" the comparison cycle beyond the first miss (refer to Section 4.1.3).

The 82385SX can execute a memory access on the 82385SX local bus only if it currently owns the bus. If not, an 82385SX in master mode will run the cycle after the current master releases the bus. An 82385SX in slave mode will issue a hold request, and will run the cycle as soon as the request is acknowledged. (This is true for any read or write cycle that needs to run on the 82385SX local bus.)

1.3.2 386™ SX MEMORY WRITE CYCLES

The 82385SX's "posted write" capability allows the majority of 386 SX memory write cycles to run with zero wait states. The primary memory update policy implemented in a posted write is the traditional cache "write through" technique, which implies that main memory is always updated in any memory write cycle. If the referenced location also happens to reside in the cache (a write hit), the cache is updated as well.

Beyond this, a posted write latches the 386 SX address, data, and cycle definition signals, and the 386 SX local bus is terminated without any wait states, even though the corresponding 82385SX local bus cycle is not yet completed, or perhaps not even started. A posted write is possible because the 82385SX's bus state machine, which is almost identical to the 386 SX bus state machine, is able to run 82385SX local bus cycles independently of the 386 SX. The only time the 386 SX sees write cycle wait states is when a previously latched (posted) write has not yet been completed on the 82385SX local bus or during an I/O write (which is not posted). An 386 SX write can be posted even if the 82385SX does not currently own the 82385SX local bus. In this case, an 82385SX in master mode will run the cycle as soon as the current master releases the bus, and an 82385SX in slave mode will request the bus and run the cycle when the request is acknowledged. The 386 SX is free to continue operating out of its cache (on the 386 SX local bus) during this time.

1.3.3 NON-CACHEABLE CYCLES

Non-cacheable cycles fall into one of two categories: cycles decoded as non-cacheable, and cycles

that are by default non-cacheable according to the 82385SX's design. All non-cacheable cycles are forwarded to the 82385SX local bus. Non-cacheable cycles have no effect on the cache or cache directory.

The 82385SX allows the system designer to define areas of main memory as non-cacheable. The 386 SX address bus is decoded and the decode output is connected to the 82385SX's non-cacheable access (NCA#) input. This decoding is done in the first 386 SX bus state in which the non-cacheable cycle address becomes available. Non-cacheable read cycles resemble cacheable read miss cycles, except that the cache and cache directory are unaffected. NCA# defined non-cacheable writes, like most writes, are posted.

The 82385SX defines certain cycles as non-cacheable without using its non-cacheable access input. These include I/O cycles, interrupt acknowledge cycles, and halt/shutdown cycles. I/O reads and interrupt acknowledge cycles execute as any other non-cacheable read. I/O write cycles are not posted. The 386 SX is not allowed to continue until a ready signal is returned from the system. Halt/Shutdown cycles are posted. During a halt/shutdown condition, the 82385SX local bus duplicates the behavior of the 386 SX, including the ability to recognize and respond to a BHOLD request. (The 82385SX's bus watching mechanism is functional in this condition.)

1.3.4 386™ SX LOCAL BUS CYCLES

386 SX Local Bus Cycles are accesses to resources on the 386 SX local bus other than to the 82385SX itself. The 82385SX simply ignores these accesses: they are neither forwarded to the system nor do they affect the cache. The designer sets aside memory and/or I/O space for local resources by decoding the 386 SX address bus and feeding the decode to the 82385SX's local bus access (LBA#) input. The designer can also decode the 386 SX cycle definition signals to keep specific 386 SX cycles from being forwarded to the system. For example, a multiprocessor design may wish to capture and remedy a 386 SX shutdown locally without having it detected by the rest of the system. Note that in such a design, the local shutdown cycle must be terminated by local bus control logic. The 387 SX Math Coprocessor is considered a 386 SX local bus resource, but it need not be decoded as such by the user since the 82385SX is able to internally recognize 387 SX accesses via the M/IO# and A23 pins.

1.3.5 SUMMARY OF 82385SX RESPONSE TO ALL 386™ SX CYCLES

Table 1-1 summarizes the 82385SX response to all 386 SX bus cycles, as conditioned by whether or not the cycle is decoded as local or non-cacheable. The table describes the impact of each cycle on the cache and on the cache directory, and whether or not the cycle is forwarded to the 82385SX local bus. Whenever the 82385SX local bus is marked "IDLE", it implies that this bus is available to other masters.

1.3.6 BUS WATCHING

As previously discussed, the 82385SX "qualifies" a 386 SX bus cycle in the first bus state in which the address and cycle definition signals of the cycle become available. The cycle is qualified as read or write, cacheable or non-cacheable, etc. Cacheable cycles are further classified as hit or miss according to the results of the cache comparison, which accesses the 82385SX directory and compares the appropriate directory location (tag) to the current 386 SX address. If the cycle turns out to be non-cacheable or a 386 SX local bus access, the hit/miss decision is ignored. The cycle qualification requires one 386 SX state. Since the fastest 386 SX access is two states, the second state can be used for bus watching.

When the 82385SX does not own the system bus, it monitors system bus cycles. If another master writes into main memory, the 82385SX latches the system address and executes a cache look-up to see if the altered main memory location resides in the cache. If so (a snoop hit), the cache entry is marked invalid in the cache directory. Since the directory is at most only being used every other state to qualify 386 SX accesses, snoop look-ups are interleaved between 386 SX local bus look-ups. The cache directory is time multiplexed between the 386 SX address and the latched system address. The result is that all snoops are caught and serviced without slowing down the 386 SX, even when running zero wait state hits on the 386 SX local bus.

1.3.7 CACHE FLUSH

The 82385SX offers a cache flush input. When activated, this signal causes the 82385SX to invalidate all data which had previously been cached. Specifically, all tag valid bits are cleared. (Refer to the 82385SX directory structure in Section 2.1.1.) There-

Table 1-1. 82385SX Response to 386™ SX Cycles

386 SX Bus Cycle Definition					82385SX Response when Decoded as Cacheable			82385SX Response when Decoded as Non-Cacheable			82385SX Response when Decoded as a 386SX Local Bus Access		
M/IO#	D/C#	W/R#	386 SX Cycle		Cache	Cache Directory	82385SX Local Bus	Cache	Cache Directory	82385SX Local Bus	Cache	Cache Directory	82385SX Local Bus
0	0	0	INT ACK	N/A	—	—	INT ACK	—	—	INT ACK	—	—	IDLE
0	0	1	UNDEFINED	N/A			UNDEFINED			UNDEFINED			IDLE
0	1	0	I/O READ	N/A	—	—	I/O READ	—	—	I/O READ	—	—	IDLE
0	1	1	I/O WRITE	N/A	—	—	I/O WRITE	—	—	I/O WRITE	—	—	IDLE
1	0	0	MEM CODE READ	HIT	CACHE READ	—	IDLE	—	—	MEM CODE READ	—	—	IDLE
				MISS	CACHE WRITE	DATA VALIDATION	MEM CODE READ						
1	0	1	HALT/ SHUTDOWN	N/A	—	—	HALT/ SHUTDOWN	—	—	HALT/ SHUTDOWN	—	—	IDLE
1	1	0	MEM DATA READ	HIT	CACHE READ	—	IDLE	—	—	MEM DATA READ	—	—	IDLE
				MISS	CACHE WRITE	DATA VALIDATION	MEM DATA READ						
1	1	1	MEM DATA WRITE	HIT	CACHE WRITE	—	MEM DATA WRITE	—	—	MEM DATA WRITE	—	—	IDLE
				MISS	—	—	MEM DATA WRITE						

NOTES:
- A dash (—) indicates that the cache and cache directory are unaffected. This table does not reflect how an access affects the LRU bit.
- An "IDLE" 82385SX Local Bus implies that this bus is available to other masters.
- The 82385SX's response to 387™ SX accesses is the same as when decoded as a 386 SX Local Bus Access.
- The only other operations that affect the cache directory are:

1. RESET or Cache Flush—all tag valid bits cleared.
2. Snoop Hit—corresponding line valid bit cleared.

fore, the cache is empty and subsequent cycles are misses until the 386 SX begins repeating the new accesses (hits). The primary use of the FLUSH input is for diagnostics and multi-processor support.

NOTE:
The use of this pin as a coherency mechanism may impact software transparency.

2.0 82385SX CACHE ORGANIZATION

The 82385SX supports two cache organizations: a simple direct mapped organization and a slightly more complex, higher performance two way set associative organization. The choice is made by strapping an 82385SX input (2W/D#) either high or low. This chapter describes the structure and operation of both organizations.

2.1 Direct Mapped Cache

2.1.1 DIRECT MAPPED CACHE STRUCTURE AND TERMINOLOGY

Figure 2-1 depicts the relationship between the 82385SX's internal cache directory, the external cache memory, and the 386 SX's physical address space. The 386 SX address space can conceptually be thought of as cache "pages" each being 8K words (16 Kbytes) deep. The page size matches the cache size. The cache can be further divided into 1024 (0 thru 1023) sets of eight words (8 x 16 bits). Each 16-bit word is called a "line". The unit of transfer between the main memory and cache is one line.

Each block in the external cache has an associated 19-bit entry in the 82385SX's internal cache directory. This entry has three components: a 10-bit "tag", a "tag valid" bit, and eight "line valid" bits. The tag acts as a main memory page number (10 tag bits support 2^{10} pages). For example, if line 9 of page 2 currently resides in the cache, then a binary 2 is stored in the Set 1 tag field. (For any 82385SX direct mapped cache page in main memory, Set 0 consists of lines 0–7, Set 1 consists of lines 8–15, etc. Line 9 is shaded in Figure 2-1.) An important characteristic of a direct mapped cache is that line 9 of any page can only reside in line 9 of the cache. All identical page offsets map to a single cache location.

The data in a cache set is considered valid or invalid depending on the status of its tag valid bit. If clear, the entire set is considered invalid. If true, an individual line within the set is considered valid or invalid depending on the status of its line valid bit.

The 82385SX sees the 386 SX address bus (A1–A23) as partitioned into three fields: a 10-bit "tag"

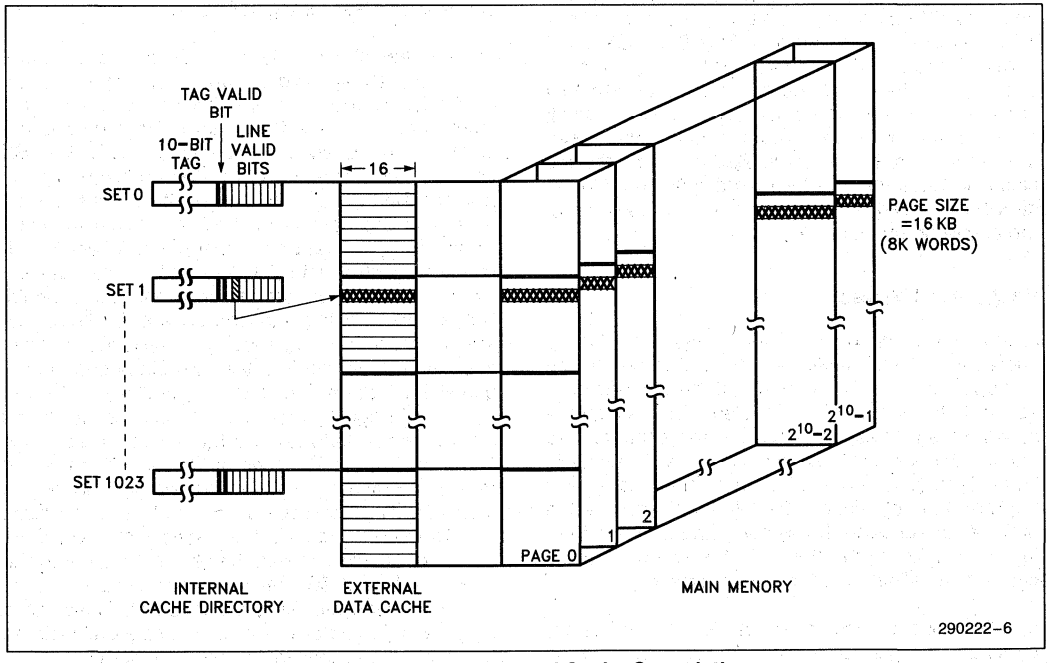


Figure 2-1. Direct Mapped Cache Organiztion

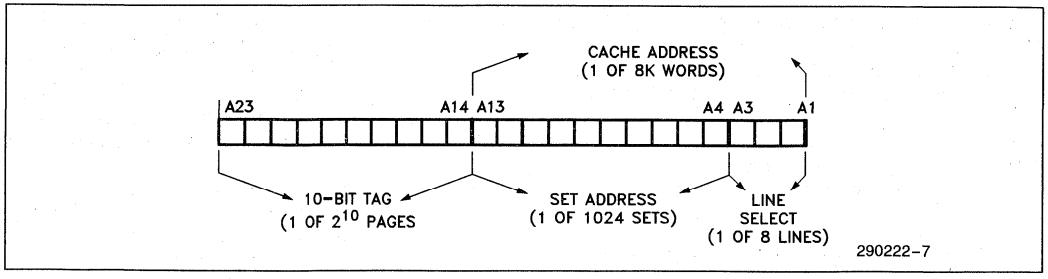


Figure 2-2. 386™ SX Address Bus Bit Fields—Direct Mapped Organization

field (A14–A23), a 10-bit "set address" field (A4–A13), and a 3-bit "line select" field (A1–A3). (See Figure 2-2.) The lower 13 address bits (A1–A13) also serve as the "cache address" which directly selects one of 8K words in the external cache.

2.1.2 DIRECT MAPPED CACHE OPERATION

The following is a description of the interaction between the 386 SX, cache, and cache directory.

2.1.2.1 Read Hits

When the 386 SX initiates a memory read cycle, the 82385SX uses the 10-bit set address to select one of 1024 directory entries, and the 3-bit line select field to select one of eight line valid bits within the entry. The 13-bit cache address selects the corresponding word in the cache. The 82385SX compares the 10-bit tag field (A14–A23 of the 386 SX access) with the tag stored in the selected directory entry. If the tag and upper address bits match, and if both the tag and appropriate line valid bits are set, the result is a hit, and the 82385SX directs the cache to drive the selected word onto the 386 SX data bus. A read hit does not alter the contents of the cache or directory.

2.1.2.2 Read Misses

A read miss can occur in two ways. The first is known as a "line" miss, and occurs when the tag and upper address bits match and the tag valid bit is set, but the line valid bit is clear. The second is called a "tag" miss, and occurs when either the tag and upper address bits do not match, or the tag valid bit is clear. (The line valid bit is a "don't care" in a tag miss.) In both cases, the 82385SX forwards the 386 SX reference to the system, and as the returning data is fed to the 386 SX, it is written into the cache and validated in the cache directory.

In a line miss, the incoming data is validated simply by setting the previously clear line valid bit. In a tag miss, the upper address bits overwrite the previously stored tag, the tag valid bit is set, the appropriate line valid bit is set, and the other seven line valid bits are cleared. Subsequent tag hits with line misses will only set the appropriate line valid bit. (Any data associated with the previous tag is no longer considered resident in the cache.)

2.1.2.3 Other Operations That Affect the Cache and Cache Directory

The other operations that affect the cache and/or directory are write hits, snoop hits, cache flushes, and 82385SX resets. In a write hit, the cache is updated along with main memory, but the directory is unaffected. In a snoop hit, the cache is unaffected, but the affected line is invalidated by clearing its line valid bit in the directory. Both an 82385SX reset and cache flush clear all tag valid bits.

When a 386 SX CPU/82385SX system "wakes up" upon reset, all tag valid bits are clear. At this point, a read miss is the only mechanism by which main memory data is copied into the cache and validated in the cache directory. Assume an early 386 SX code access seeks (for the first time) line 9 of page 2. Since the tag valid bit is clear, the access is a tag miss, and the data is fetched from main memory. Upon return, the data is fed to the 386 SX and simultaneously written into line 9 of the cache. The set directory entry is updated to show this line as valid. Specifically, the tag and appropriate line valid bits are set, the remaining seven line valid bits cleared, and binary 2 written into the tag. Since code is sequential in nature, the 386 SX will likely next want line 10 of page 2, then line 11, and so on. If the 386 SX sequentially fetches the next six lines, these fetches will be line misses, and as each is fetched from main memory and written into the cache, its corresponding line valid bit is set. This is the basic flow of events that fills the cache with valid data. Only after a piece of data has been copied into the cache and validated can it be accessed in a zero wait state read hit. Also, a cache entry must have been validated before it can be subsequently altered by a write hit, or invalidated by a snoop hit.

An extreme example of "trashing" is if line 9 of page two is an instruction to jump to line 9 of page one, which is an instruction to jump back to line 9 of page two. Trashing results from the direct mapped cache characteristic that all identical page offsets map to a single cache location. In this example, the page one access overwrites the cached page two data, and the page two access overwrites the cached page one data. As long as the code jumps back and forth the hit rate is zero. This is of course an extreme case. The effect of trashing is that a direct mapped cache exhibits a slightly reduced overall hit rate as compared to a set associative cache of the same size.

2.2 Two Way Set Associative Cache

2.2.1 TWO WAY SET ASSOCIATIVE CACHE STRUCTURE AND TERMINOLOGY

Figure 2-3 illustrates the relationship between the directory, cache, and 386 SX address space. Whereas the direct mapped cache is organized as one bank of 8K words, the two way set associative cache is organized as two banks (A and B) of 4K words each. The page size is halved, and the number of pages doubled. (Note the extra tag bit.) The cache now has 512 sets in each bank. (Two banks times 512 sets gives a total of 1024. The structure can be thought of as two half-sized direct mapped caches in parallel.) The performance advantage over a direct mapped cache is that all identical page offsets map to two cache locations instead of one, reducing the potential for thrashing. The 82385SX's partitioning of the 386 SX address bus is depicted in Figure 2-4.

2.2.2 LRU REPLACEMENT ALGORITHM

The two way set associative directory has an additional feature: the "least recently used" or LRU bit. In the event of a read miss, either bank A or bank B will be updated with new data. The LRU bit flags the candidate for replacement. Statistically, of two blocks of data, the block most recently used is the block most likely to be needed again in the near future. By flagging the least recently used block, the 82385SX ensures that the cache block replaced is the least likely to have data needed by the CPU.

2.2.3 TWO WAY SET ASSOCIATIVE CACHE OPERATION

2.2.3.1 Read Hits

When the 386 SX initiates a memory read cycle, the 82385SX uses the 9-bit set address to select one of 512 sets. The two tags of this set are simultaneously compared with A13–A23, both tag valid bits checked, and both appropriate line valid bits checked. If either comparison produces a hit, the corresponding cache bank is directed to drive the selected word onto the 386 SX data bus. (Note that both banks will never concurrently cache the same main memory location.) If the requested data resides in bank A, the LRU bit is pointed toward B. If B produces the hit, the LRU bit is pointed toward A.

2.2.3.2 Read Misses

As in direct mapped operation, a read miss can be either a line or tag miss. Let's start with a tag miss example. Assume the 386 SX seeks line 9 of page 2, and that neither the A or B directory produces a tag match. Assume also, as indicated in Figure 2-3, that the LRU bit points to A. As the data returns from main memory, it is loaded into offset 9 of bank A. Concurrently, this data is validated by updating the set 1 directory entry for bank A. Specifically, the upper address bits overwite the previous tag, the tag valid bit is set, the appropriate line valid bit is set, and the other seven line valid bits cleared. Since this data is the most recently used, the LRU bit is turned toward B. No change to bank B occurs.

If the next 386 SX request is line 10 of page two, the result will be a line miss. As the data returns from main memory, it will be written into offset 10 of bank A (tag hit/line miss in bank A), and the appropriate line valid bit will be set. A line miss in one bank will cause the LRU bit to point to the other bank. In this example, however, the LRU bit has already been turned toward B.

2.2.3.3 Other Operations That Affect the Cache and Cache Directory

Other operations that affect the cache and cache directory are write hits, snoop hits, cache flushes, and 82385SX resets. A write hit updates the cache along with main memory. If directory A detects the hit, bank A is updated. If directory B detects the hit, bank B is updated. If one bank is updated, the LRU bit is pointed towards the other.

If a snoop hit invalidates an entry, for example, in cache bank A, the corresponding LRU bit is pointed toward A. This insures that invalid data is the prime candidate for replacement in a read miss. Finally, resets and flushes behave just as they do in a direct mapped cache, clearing all tag valid bits.

3.0 82385SX PIN DESCRIPTION

The 82385SX creates the 82385SX local bus, which is a functional 386 SX interface. To facilitate under-

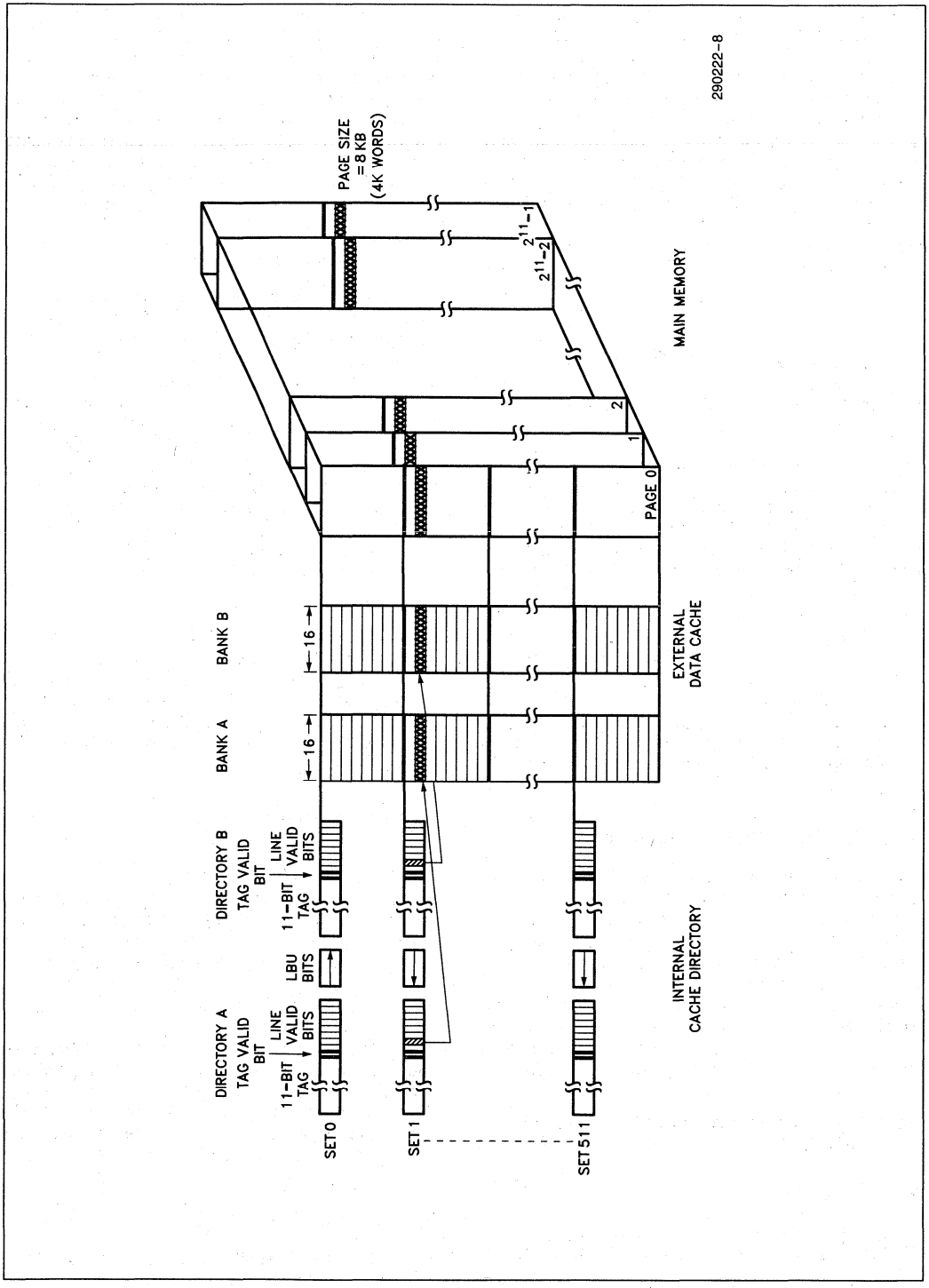


Figure 2-3. Two-Way Set Associative Cache Organization

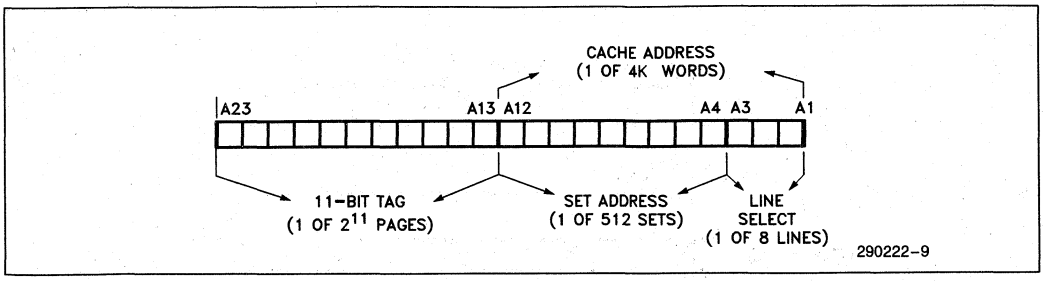


Figure 2-4. 386™ SX Address Bus Bit Fields—Two-Way Set Associative Organization

standing, 82385SX local bus signals go by the same name as their 386 SX equivalents, except that they are preceded by the letter "B". The 82385SX local bus equivalent to ADS# is BADS#, the equivalent to NA# is BNA#, etc. This convention applies to bus states as well. For example, BT1P is the 82385SX local bus state equivalent to the 386 SX T1P state.

3.1 386™ SX CPU/82385SX Interface Signals

These signals form the direct interface between the 386 SX and the 82385SX.

3.1.1 386™ SX CPU/82385SX Clock (CLK2)

CLK2 provides the fundamental timing for a 386 SX CPU/82385SX system, and is driven by the same source that drives the 386 SX CLK2 input. The 82385SX, like the 386 SX, divides CLK2 by two to generate an internal "phase indication" clock. (See Figure 3-1.) The CLK2 period whose rising edge drives the internal clock low is called PHI1, and the CLK2 period that drives the internal clock high is called PHI2. A PHI1–PHI2 combination (in that order) is known as a "T" state, and is the basis for 386 SX bus cycles.

3.1.2 386™ SX CPU/82385SX RESET (RESET)

This input resets the 82385SX, bringing it to an initial known state, and is driven by the same source that drives the 386 SX RESET input. A reset effectively flushes the cache by clearing all cache directory tag valid bits. The falling edge of RESET is synchronized to CLK2, and used by the 82385SX to properly establish the phase of its internal clock. (See Figure 3-2.) Specifically, the second internal phase following the falling edge of RESET is PHI2.

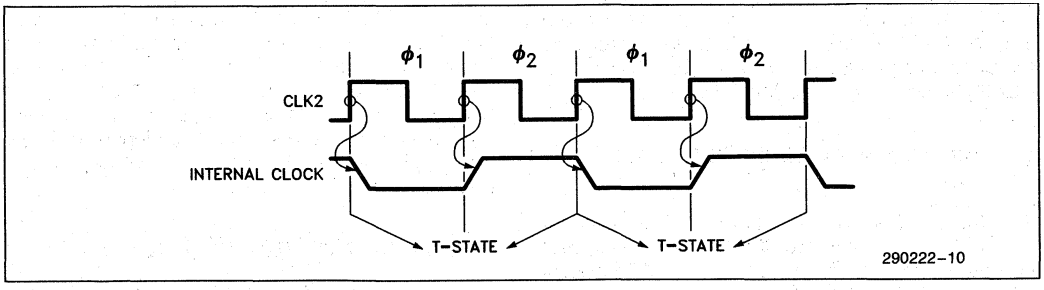


Figure 3-1. CLK2 and Internal Clock

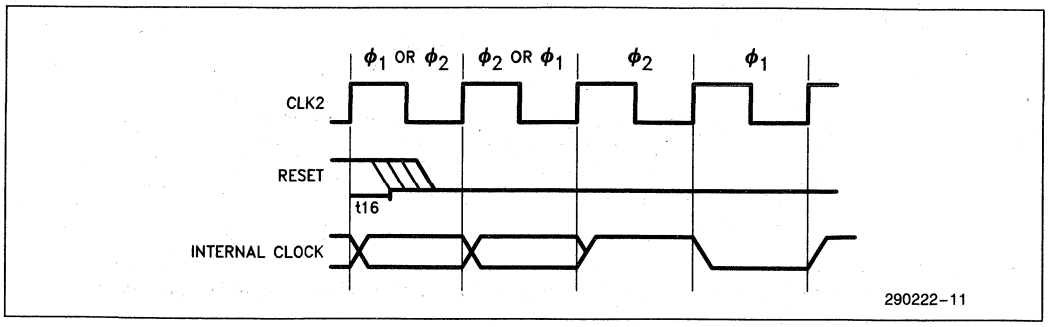


Figure 3-2. Reset/Internal Phase Relationship

3.1.3 386™ SX CPU/82385SX ADDRESS BUS (A1–A23), BYTE ENABLES (BHE#, BLE#), AND CYCLE DEFINITION SIGNALS (M/IO#, D/C#, W/R#, LOCK#)

The 82385SX directly connects to these 386 SX outputs. The 386 SX address bus is used in the cache directory comparison to see if data referenced by 386 SX resides in the cache, and the byte enables inform the 82385SX as to which portions of the data bus are involved in a 386 SX cycle. The cycle definition signals are decoded by the 82385SX to determine the type of cycle the 386 SX is executing.

3.1.4 386™ SX CPU/82385SX ADDRESS STATUS (ADS#) AND READY INPUT (READYI#)

ADS#, a 386 SX output, tells the 82385SX that new address and cycle definition information is available. READYI#, an input to both the 386 SX (via the 386 SX READY# input pin) and 82385SX, indicates the completion of a 386 SX bus cycle. ADS# and READYI# are used to track the 386 SX bus state.

3.1.5 386™ SX NEXT ADDRESS REQUEST (NA#)

This 82385SX output controls 386 SX pipelining. It can be tied directly to the 386 SX NA# input, or it can be logically "AND"ed with other 386 SX local bus next address requests.

3.1.6 READY OUTPUT (READYO#) AND BUS READY ENABLE (BRDYEN#)

The 82385SX directly terminates all but two types of 386 SX bus cycles with its READYO# output. 386 SX local bus cycles must be terminated by the local device being accessed. This includes devices decoded using the 82385SX LBA# signal and 387 accesses. The other cycles not directly terminated by the 82385SX are 82385SX local bus reads, specifically cache read misses and non-cacheable reads. (Recall that the 82385SX forwards and runs such cycles on the 82385SX bus.) In these cycles the signal that terminates the 82385SX local bus access is BREADY# which is gated through to the 386 SX local bus such that the 386 SX and 82385SX local bus cycles are concurrently terminated. BRDYEN# is used to gate the BREADY# signal to the 386 SX.

3.2 Cache Control Signals

These 82385SX outputs control the external 16 KB cache data memory.

3.2.1 CACHE ADDRESS LATCH ENABLE (CALEN)

This signal controls the latch (typically an F or AS series 74373) that resides between the low order 386 SX address bits and the cache SRAM address inputs. (The outputs of this latch are the "cache address" described in the previous chapter.) When CALEN is high the latch is transparent. The falling edge of CALEN latches the current inputs which remain applied to the cache data memory until CALEN returns to an active high state.

3.2.2 CACHE TRANSMIT/RECEIVE (CT/R#)

This signal defines the direction of an optional data transceiver (typically an F or AS series 74245) between the cache and 386 SX data bus. When high, the transceiver is pointed towards the 386 SX local data bus (the SRAMs are output enabled). When low, the transceiver points towards the cache data memory. A transceiver is required if the cache is designed with SRAMs that lack an output enable control. A transceiver may also be desirable in a system that has a heavily loaded 386 SX local data bus. These devices are not necessary when using SRAMs which incorporate an output enable.

3.2.3 CACHE CHIP SELECTS (CS0#, CS1#)

These active low signals tie to the cache SRAM chip selects, and individually enable both bytes of the 16-bit wide cache. CS0# enables D0–D7 and CS1# enables D8–D15. During read hits, both bytes are enabled regardless of whether or not the 386 SX byte enables are active. (The 386 SX ignores what it did not request.) Also, both cache bytes are enabled in a read miss so as to update the cache with a complete line (word). In a write hit, only the cache bytes that correspond to active byte enables are selected. This prevents cache data from being corrupted in a partial word write.

3.2.4 CACHE OUTPUT ENABLES (COEA#, COEB#) AND WRITE ENABLES (CWEA#, CWEB#)

COEA# and COEB# are active low signals which tie to the cache SRAM or Transceiver output enables and respectively enable cache bank A or B. The state of DEFOE# (define cache output enable), an 82385SX configuration input, determines the functional definition of COEA# and COEB#.

If DEFOE# = V_{IL}, in a two-way set associative cache, either COEA# or COEB# is active during read hit cycles only, depending on which bank is selected. In a direct mapped cache, both are activated during read hits, so the designer is free to use either one. This COEx# definition best suits cache SRAMs with output enables.

If DEFOE# = V_{IH}, COEx# is active during a read hit, read miss (cache update) and write hit cycles only. This COEx# definition best suits cache SRAMs without output enables. In such systems, transceivers are needed and their output enables must be active for writing, as well as reading, the cache SRAMs.

CWEA# and CWEB# are active low signals which tie to the cache SRAM write enables, and respectively enable cache bank A or B to receive data from the 386 SX data bus (386 SX write hit or read miss update). In a two-way set associative cache, one or the other is enabled in a read miss or write hit. In a direct mapped cache, both are activated, so the designer is free to use either one.

The various cache configurations supported by the 82385SX are described in Section 4.2.1.

3.3 386™ SX Local Bus Decode Inputs

These 82385SX inputs are generated by decoding the 386 SX address and cycle definition lines. These active low inputs are sampled at the end of the first state in which the address of a new 386 SX cycle becomes available. (T1 or first T2P.)

3.3.1 386™ SX LOCAL BUS ACCESS (LBA#)

This input identifies a 386 SX access as directed to a resource (other than the cache) on the 386 SX local bus. (The 387 SX Math Coprocessor is considered a 386 SX local bus resource, but LBA# need not be generated as the 82385SX internally decodes 387 SX accesses.) The 82385SX simply ignores these cycles. They are neither forwarded to the system nor do they affect the cache or cache directory. Note that LBA# has priority over all other types of cycles. If LBA# is asserted, the cycle is interpreted as a 386 SX local bus access, regardless of the cycle type or status of NCA#. This allows any 386 SX cycle (memory, I/O, interrupt acknowledge, etc.) to be kept on the 386 SX local bus if desired.

3.3.2 NON-CACHEABLE ACCESS (NCA#)

This active low input identifies a 386 SX cycle as non-cacheable. The 82385SX forwards non-cacheable cycles to the 82385SX local bus and runs them. The cache and cache directory are unaffected.

NCA# allows a designer to set aside a portion of main memory as non-cacheable. Potential applications include memory-mapped I/O and systems where multiple masters access dual ported memory via different busses. Another possibility makes use of the 386 SX D/C# output. The 82385SX by default implements a unified code and data cache, but driving NCA# directly by D/C# creates a data only cache. If D/C# is inverted first, the result is a code only cache.

3.4 82385SX Local Bus Interface Signals

The 82385SX presents an "386 SX-like" front end to the system, and the signals discussed in this section are 82385SX local bus equivalents to actual 386 SX signals. These signals are named with respect to their 386 SX counterparts, but with the letter "B" appended to the front.

Note that the 82385SX itself does not have equivalent output signals to the 386 SX data bus (D0–D15) address bus (A1–A23), and cycle definition signals (M/IO#, D/C#, W/R#). The 82385SX data bus (BD0–BD15) is actually the system side of a latching transceiver, and the 82385SX address bus and cycle definition signals (BA1–BA23, BM/IO#, BD/C#,

BW/R#) are the outputs of an edge-triggered latch. The signals that control this data transceiver and address latch are discussed in Section 3.5.

3.4.1 82385SX BUS BYTE ENABLES (BBHE#, BBLE#)

BBHE# and BBLE# are the 82385SX local bus equivalents to the 386 SX byte enables. In a cache read miss, the 82385SX drives both signals low, regardless of whether or not the 386 SX byte enables are active. This insures that a complete line (word) is fetched from main memory for the cache update. In all other 82385SX local bus cycles, the 82385SX duplicates the logic levels of the 386 SX byte enables. The 82385SX tri-states these outputs when it is not the current bus master.

3.4.2 82385SX BUS LOCK (BLOCK#)

BLOCK# is the 82385SX local bus equivalent to the 386 SX LOCK# output, and distinguishes between locked and unlocked cycles. When the 386 SX runs a locked sequence of cycles (and LBA# is negated), the 82385SX forwards and runs the sequence on the 82385SX local bus, regardless of whether any locations referenced in the sequence reside in the cache. A read hit will be run as if it is a read miss, but a write hit will update the cache as well as being completed to system memory. In keeping with 386 SX behavior, the 82385SX does not allow another master to interrupt the sequence. BLOCK# is tri-stated when the 82385SX is not the current bus master.

3.4.3 82385SX BUS ADDRESS STATUS (BADS#)

BADS# is the 82385SSX local bus equivalent of ADS#, and indicates that a valid address (BA1–BA23, BBHE#, BBLE#) and cycle definition (BM/IO#, BW/R#, BD/C#) are available. It is asserted in BT1 and BT2P states, and is tri-stated when the 82385SX does not own the bus.

3.4.4 82385SX BUS READY INPUT (BREADY#)

82385SX local bus cycles are terminated by BREADY#, just as 386 SX cycles are terminated by the 386 SX READY# input. In 82385SX local bus read cycles, BREADY# is gated by BRDYEN# onto the 386 SX local bus, such that it terminates both the 386 SX and 82385SX local bus cycles.

3.4.5 82385SX BUS NEXT ADDRESS REQUEST (BNA#)

BNA# is the 82385SX local bus equivalent to the 386 SX NA# input, and indicates that the system is prepared to accept a pipelined address and cycle definition. If BNA# is asserted and the new cycle information is available, the 82385SX begins a pipelined cycle on the 82385SX local bus.

3.5 82385SX Bus Data Transceiver and Address Latch Control Signals

The 82385SX data bus is the system side of a latching transceiver (typically for F or AS series 74646), and the 82385SX address bus and cycle definition signals are the outputs of an edge-triggered latch (F or AS series 74374). The following is a discussion of the 82385SX outputs that control these devices. An important characteristic of these signals and the devices they control is that they ensure that BD0–BD15, BA1–BA23, BM/IO#, BD/C# and BW/R# reproduce the functionality and timing behavior of their 386 SX equivalents.

3.5.1 LOCAL DATA STROBE (LDSTB), DATA OUTPUT ENABLE (DOE#), AND BUS TRANSMIT/RECEIVE (BT/R#)

These signals control the latching data transceiver. BT/R# defines the transceiver direction. When high, the transceiver drives the 82385SX data bus in write cycles. When low, the transceiver drives the 386 SX data bus in 82385SX local bus read cycles. DOE# enables the transceiver outputs.

The rising edge of LDSTB latches the 386 SX data bus in all write cycles. The interaction of this signal and the latching transceiver is used to perform the 82385SX's posted write capability.

3.5.2 BUS ADDRESS CLOCK PULSE (BACP) AND BUS ADDRESS OUTPUT ENABLE (BAOE#)

These signals control the latch that drives BA1–BA23, BM/IO#, BW/R#, and BD/C#. In any 386 SX cycle that is forwarded to the 82385SX local bus, the rising edge of BACP latches the 386 SX address and cycle definition signals. BAOE# enables the latch outputs when the 82385SX is the current bus master and disables them otherwise.

3.6 Status and Control Signals

3.6.1 CACHE MISS INDICATION (MISS#)

This output accompanies cacheable read and write miss cycles. This signal transitions to its active low state when the 82385SX determines that a cacheable 386 SX access is a miss. Its timing behavior

follows that of the 82385SX local bus cycle definition signals (BM/IO#, BD/C#, BW/R#) so that it becomes available with BADS# in BT1 or the first BT2P. MISS# is floated when the 82385SX does not own the bus, such that multiple 82385SX's can share the same node in multi-cache systems. (As discussed in Chapter 7, this signal also serves a reserved function in testing the 82385SX.)

3.6.2 WRITE BUFFER STATUS (WBS)

The latching data transceiver is also known as the "posted write buffer". WBS indicates that this buffer contains data that has not yet been written to the system even though the 386 SX may have begun its next cycle. It is activated when 386 SX data is latched, and deactivated when the corresponding 82385SX local bus write cycle is completed (BREADY#). (As discussed in Chapter 7, this signal also serves a reserved function in testing the 82385SX.)

WBS can serve several functions. In multi-processor applications, it can act as a coherency mechanism by informing a bus arbiter that it should let a write cycle run on the system bus so that main memory has the latest data. If any other 82385SX cache subsystems are on the bus, they will monitor the cycle via their bus watching mechanisms. Any 82385SX that detects a snoop hit will invalidate the corresponding entry in its local cache.

3.6.3 CACHE FLUSH (FLUSH)

When activated, this signal causes the 82385SX to clear all of its directory tag valid bits, effectively flushing the cache. (As discussed in Chapter 7, this signal also serves a reserved function in testing the 82385SX.) The primary use of the FLUSH input is for diagnostics and multi-processor support. The use of this pin as a coherency mechanism may impact software transparency.

The FLUSH input must be held active for at least 4 CLK (8 CLK2) cycles to complete the flush sequence. If FLUSH is still active after 4 CLK cycles, any accesses to the cache will be misses and the cache will not be updated (since FLUSH is active).

3.7 Bus Arbitration Signals (BHOLD and BHLDA)

In master mode, BHOLD is an input that indicates a request by a slave device for bus ownership. The 82385SX acknowledges this request via its BHLDA output. (These signals function identically to the 386 SX HOLD and HLDA signals.)

The roles of BHOLD and BHLDA are reversed for an 82385SX in slave mode. BHOLD is now an output indicating a request for bus ownership, and BHLDA an input indicating that the request has been granted.

3.8 Coherency (Bus Watching) Support Signals (SA1–SA23, SSTB#, SEN)

These signals form the 82385SX's bus watching interface. The Snoop Address Bus (SA1–SA23) connects to the system address lines if masters reside at both the system and 82385SX local bus levels, or the 82385SX local bus address lines if masters reside only at the 82385SX local bus level. Snoop Strobe (SSTB#) indicates that a valid address is on the snoop address inputs. Snoop Enable (SEN) indicates that the cycle is a write. In a system with masters only at the 82385SX local bus level, SA1–SA23, SSTB#, and SEN can be driven respectively by BA1–BA23, BADS#, and BW/R# without any support circuitry.

3.9 Configuration Inputs (2W/D#, M/S#, DEFOE#)

These signals select the configurations supported by the 82385SX. They are hardware strap options and must not be changed dynamically. 2W/D# (2-Way/Direct Mapped Select) selects a two-way set associative cache when tied high, or a direct mapped cache when tied low. M/S# (Master/Slave Select) chooses between master mode (M/S# high) and slave mode (M/S# low). DEFOE# defines the functionality of the 82385SX cache output enables (COEA# and COEB#). DEFOE# allows the 82385SX to interface to SRAMs with output enables (DEFOE# low) or to SRAMs requiring transceivers (DEFOE# high).

3.10 Reserved Pins (RES)

Some pins on the 82385SX are reserved for internal testing and future cache features. To assure compatibility and functionality, these reserved pins must be configured as shown in Table 3.10.1.

Table 3.10.1. Reserved Pin Connections

PGA Pin Location	PQFP Pin Location	Logic Level
A12	1	High
A13	131	High
B10	7	High
B11	3	High
B12	132	High
C10	4	High
C11	2	High
G13	117	High
H12	110	High
J3	60	High
J14	109	High
K1	58	High
K2	59	High
K3	62	High
L1	61	High
L2	63	High
L3	64	High
L12	100	No Connect
L14	102	High
M13	101	No Connect
N6	75	No Connect
P5	76	No Connect

4.0 386 SX LOCAL BUS INTERFACE

The following is a detailed description of how the 82385SX interfaces to the 386 SX and to 386 SX local bus resources. Items specifically addressed are the interfaces to the 386 SX, the cache SRAMs, and the 387 SX Math Coprocessor.

The many timing diagrams in this and the next chapter provide insight into the dual pipelined bus structure of a 386 SX CPU/82385SX system. It's important to realize, however, that one need not know every possible cycle combination to use the 82385SX. The interface is simple, and the dual bus operation invisible to the 386 SX and system. To facilitate discussion of the timing diagrams, several conventions have been adopted. Refer to Figure 4-2A, and note that 386 SX bus cycles, 386 SX bus states, and 82385SX bus states are identified along the top. All states can be identified by the "frame numbers" along the bottom. The cycles in Figure 4-2A include a cache read hit (CRDH), a cache read miss (CRDM), and a write (WT). WT represents any write, cacheable or not. When necessary to distinguish cacheable writes, a write hit goes by CWTH and a write miss by CWTM. Non-cacheable system reads go by SBRD. Also, it is assumed that system bus pipelining occurs even though the BNA# signal is not shown. When the system pipeline begins is a function of the system bus controller.

386 SX bus cycles can be tracked by ADS# and READYI#, and 82385SX cycles by BADS# and BREADY#. These four signals are thus a natural choice to help track parallel bus activity. Note in the timing diagrams that 386 SX cycles are numbered using ADS# and READYI#, and 82385SX cycles using BADS# and BREADY#. For example, when the address of the first 386 SX cycle becomes available, the corresponding assertion of ADS# is marked "1", and the READYI# pulse that terminates the cycle is marked "1" as well. Whenever a 386 SX cycle is forwarded to the system, its number is forwarded as well so that the corresponding 82385SX bus cycle can be tracked by BADS# and BREADY#.

The "N" value in the timing diagrams is the assumed number of main memory wait states inserted in a non-pipelined 82385SX bus cycle. For example, a non-pipelined access to N=2 memory requires a total of four bus states, while a pipelined access requires three. (The pipeline advantage effectively hides one main memory wait state.)

4.1 Processor Interface

This section presents the 386 SX CPU/82385SX hardware interface and discusses the interaction and timing of this interface. Also addressed is how to decode the 386 SX address bus to generate the 82385SX inputs LBA# and NCA#. (Recall that LBA# allows memory and/or I/O space to be set aside for 386 SX local bus resources; and NCA# allows system memory to be set aside as non-cacheable.)

4.1.1 HARDWARE INTERFACE

Figure 4-1 is a diagram of a 386 SX CPU/82385SX system, which can be thought of as three distinct interfaces. The first is the 386 SX CPU/82385SX interface (including the Ready Logic). The second is the cache interface, as depicted by the cache control bus in the upper left corner of Figure 4-1. The third is the 82385SX bus interface, which includes both direct connects and signals that control the 74374 address/cycle definition latch and 74646 latching data transceiver. (The 82385SX bus interface is the subject of the next chapter.)

As seen in Figure 4-1, the 386 SX CPU/82385SX interface is a straightforward connection. The only necessary support logic is that required to sum all ready sources.

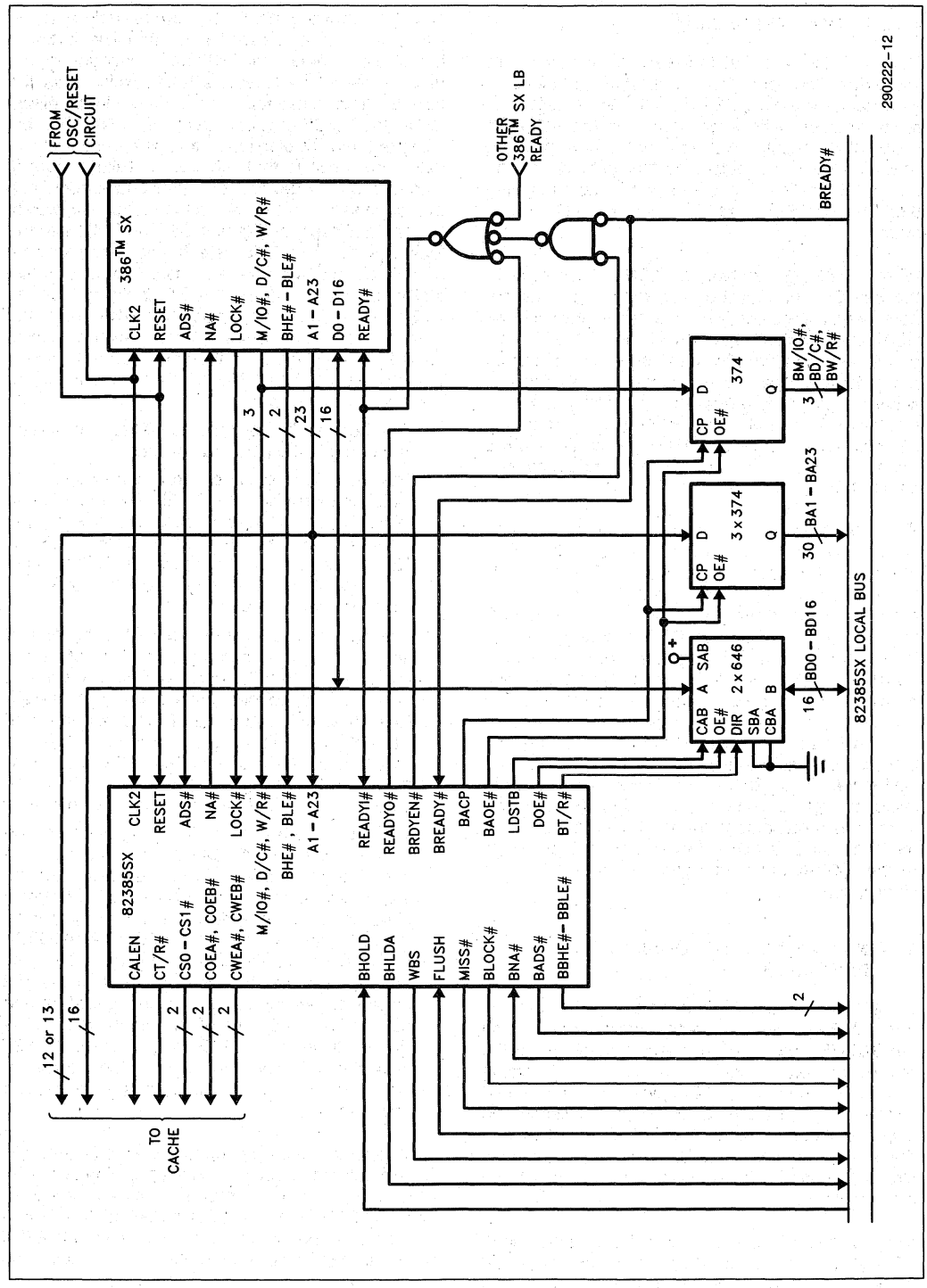


Figure 4-1. 386™ SX CPU/82385SX Interface

4.1.2 READY GENERATION

Note in Figure 4-1 that the ready logic consists of two gates. The upper three-input AND gate (shown as a negative logic OR) sums all 386 SX local bus ready sources. One such source is the 82385SX READYO# output, which terminates read hits and posted writes. The output of this gate drives the 386 SX READY# input and is monitored by the 82385SX (via READYI#) to track the 386 SX bus state.

When the 82385SX forwards a 386 SX read cycle to the 82385SX bus (cache read miss or non-cacheable read), it does not directly terminate the cycle via READYO#. Instead, the 386 SX and 82385SX bus cycles are concurrently terminated by a system ready source. This is the purpose of the additional two-input OR gate (negative logic AND) in Figure 4-1. When the 82385SX forwards a read to the 82385SX bus, it asserts BRDYEN# which enables the system ready signal (BREADY#) to directly terminate the 386 SX bus cycle.

Figure 4-2A and 4-2B illustrate the behavior of the signals involved in ready generation. Note in cycle 1 of Figure 4-2A that the 82385SX READYO# directly terminates the hit cycle. In cycle 2, READYO# is not activated. Instead the 82385SX BRDYEN# is activated in BT2, BT2P, or BT2I states such that BREADY# can concurrently terminate the 386 SX and 82385SX bus cycles (frame 6). Cycle 3 is a posted write. The write data becomes available in T1P (frame 7), and the address, data, and cycle definition of the write are latched in T2 (frame 8). The 386 SX cycle is terminated by READYO# in frame 8 with no wait states. The 82385SX, however, sees the write cycle through to completion on the 82385SX bus where it is terminated in frame 10 by BREADY#. In this case, the BREADY# signal is not gated through to the 386 SX. Refer to Figures 4-2A and 4-2B for clarification.

4.1.3 NA# AND 386 SX LOCAL BUS PIPELINING

Cycle 1 of Figure 4-2A is a typical cache read hit. The 386 SX address becomes available in T1, and the 82385SX uses this address to determine if the referenced data resides in the cache. The cache look-up is completed and the cycle qualified as a hit or miss in T1. If the data resides in the cache, the cache is directed to drive the 386 SX data bus, and the 82385SX drives its READYO# output so the cycle can be terminated at the end of the first T2 with no wait states.

Although cycle 2 starts out like cycle 1, at the end of T1 (frame 3), it is qualified as a miss and forwarded to the 82385SX bus. The 82385SX bus cycle begins one state after the 386 SX bus cycle, implying a one wait state overhead associated with cycle 2 due to the look-up. When the 82385SX encounters the miss, it immediately asserts NA#, which puts the 386 SX into pipelined mode. Once in pipelined mode, the 82385SX is able to qualify a 386 SX cycle using the 386 SX pipelined address and control signals. The result is that the cache look-up state is hidden in all but the first of a contiguous sequence of read misses. This is shown in the first two cycles, both read misses, of Figure 4-2B. The CPU sees the look-up state in the first cycle, but not in the second. In fact, the second miss requires a total of only two states, as not only does 386 SX pipelining hide the look-up state, but system pipelining hides one of the main memory wait states. (System level pipelining via BNA# is discussed in the next chapter.) Several characteristics of the 82385SX's pipelining of the 386 SX are as follows:

— The above discussion applies to all system reads, not just cache read misses.

— The 82385SX provides the fastest possible switch to pipelining, T1-T2-T2P. The exception to this is when a system read follows a posted write. In this case, the sequence is T1-T2-T2-T2P. (Refer to cycle 4 of Figure 4-2A.) The number of T2 states is dependent on the number of main memory wait states.

— Refer to the read hit in Figure 4-2A (cycle 1), and note that NA# is actually asserted before the end of T1, before the hit/miss decision is made. This is of no consequence since even though NA# is sampled active in T2, the activation of READYO# in the same T2 renders NA# a "don't care". NA# is asserted in this manner to meet 386 SX timing requirements and to insure the fastest possible switch to pipelined mode.

— All read hits and the majority of writes can be serviced by the 82385SX with zero wait states in non-pipelined mode, and the 82385SX accordingly attempts to run all such cycles in non-pipelined mode. An exception is seen in the hit cycles (cycles 3 and 4) of Figure 4-2B. The 82385SX does not know soon enough that cycle 3 is a hit, and thus sustains the pipeline. The result is that three sequential hits are required before the 386 SX is totally out of pipelined mode. (The three hits look like T1P-T2P, T1P-T2, T1-T2.) Note that this does not occur if the number of main memory wait states is equal to or greater than two.

As far as the design is concerned, NA# is generally tied directly to the 386 SX NA# input. However, other local NA# sources may be logically "AND"ed with the 82385SX NA# output if desired. It is essential, however, that no device other than the 82385SX drive the 386 SX NA# input unless that device re-

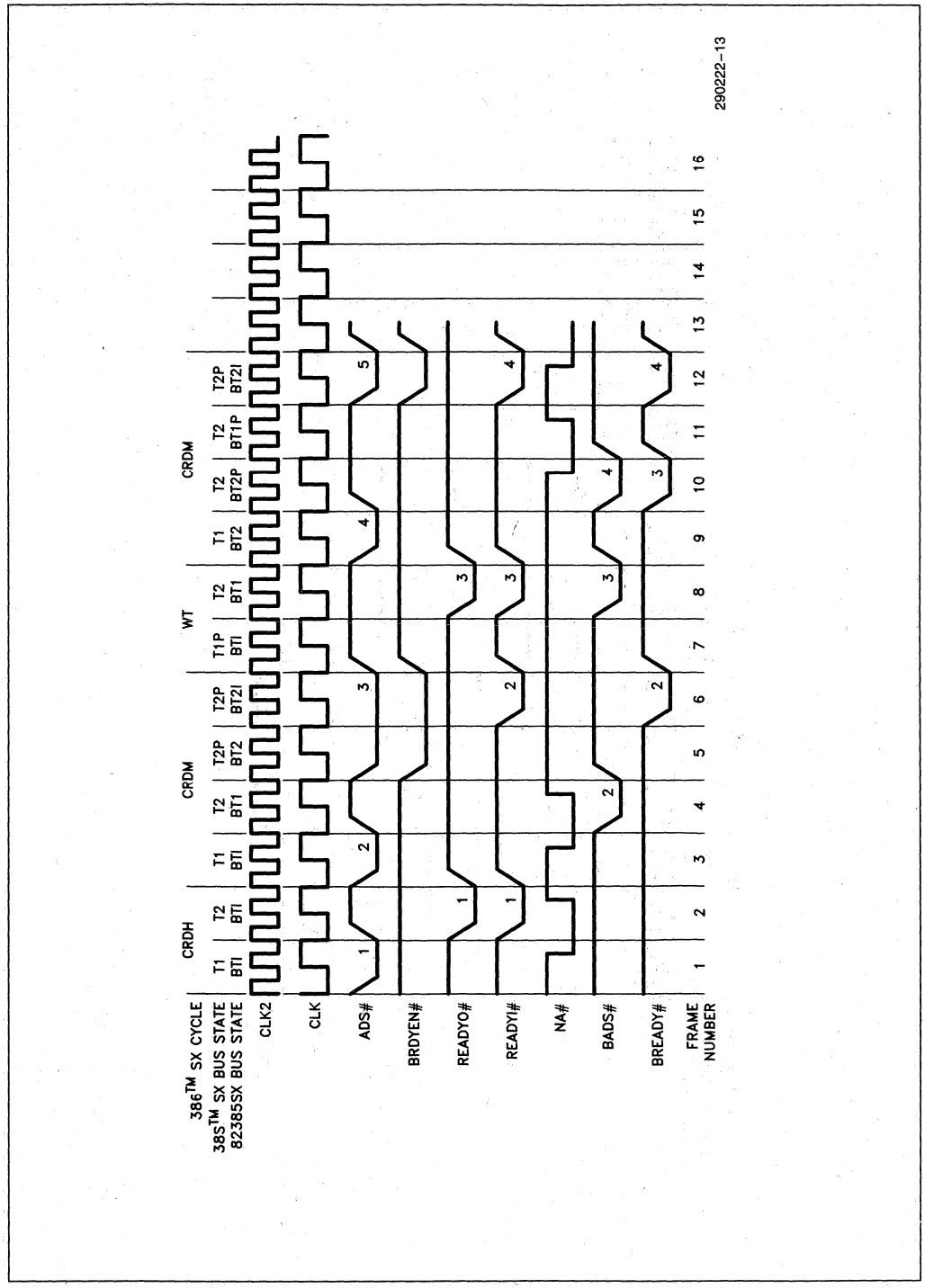


Figure 4-2A. READYO#, BRDYEN#, and NA# (N = 1)

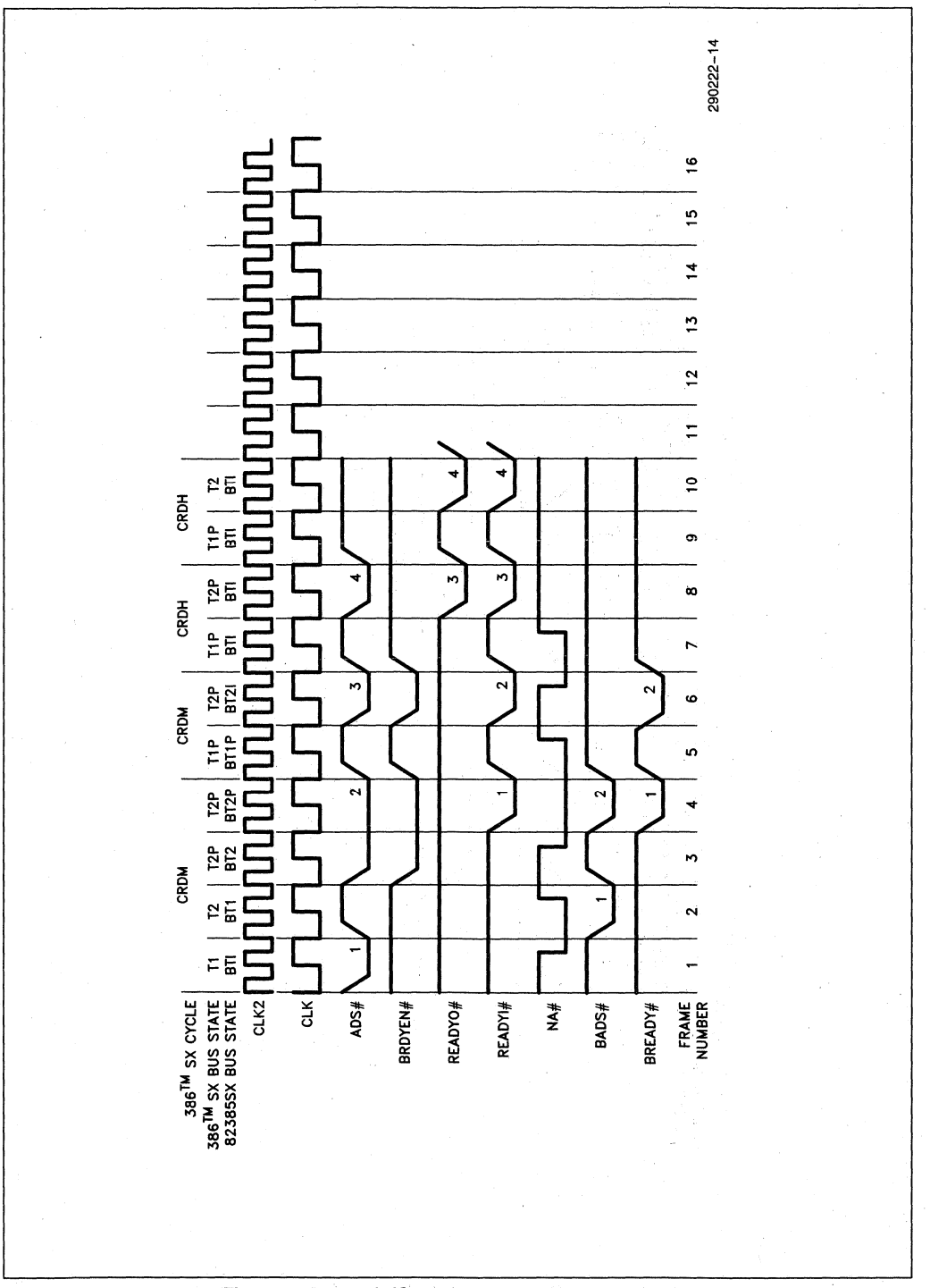


Figure 4-2B. READYO#, BRDYEN#, and NA# (N = 1)

sides on the 386 SX local bus in space decoded via LBA#. If desired, the 82385SX NA# output can be ignored and the 386 SX NA# input tied high. The 386 SX NA# input should never be tied low, which would always keep it active.

4.1.4 LBA# AND NCA# GENERATION

The 82385SX inputs signals LBA# and NCA# are generated by decoding the 386 SX address (A1–A23) and cycle definition (W/R#, D/C#, M/IO#) lines. The 82385SX samples them at the end of the first state in which they become available, which is either T1 or the first T2P cycle. The decode configuration and timings are illustrated respectively in Figures 4-3A and 4-3B.

4.2 Cache Interface

The following is a description of the external data cache and 82385SX cache interface.

4.2.1 CACHE CONFIGURATIONS

The 82385SX controls the cache memory via the control signals shown in Figure 4-1. These signals drive one of four possible cache configurations, as depicted in Figures 4-4A through 4-4D. Figure 4-4A shows a direct mapped cache organized as 8K words. The likely design choice is two 8K x 8 SRAMs. Figure 4-4B depicts the same cache memory but with a data transceiver between the cache and 386 SX data bus. In this configuration, CT/R# controls the transceiver direction, COEA# drives the transceiver output enable (COEB# could also be used), and DEFOE# is strapped high. A data buffer is required if the chosen SRAM does not have a separate output enable. Additionally, buffers may be used to ease SRAM timing requirements or in a system with a heavily loaded data bus. (Guidelines for SRAM selection are included in Chapter 6.)

Figure 4-4C depicts a two-way set associative cache organized as two banks (A and B) of 4K words each. The likely design choice is eight 4K x 4 SRAMs. Finally, Figure 4-4D depicts the two-way organization with data buffers between the cache memory and data bus.

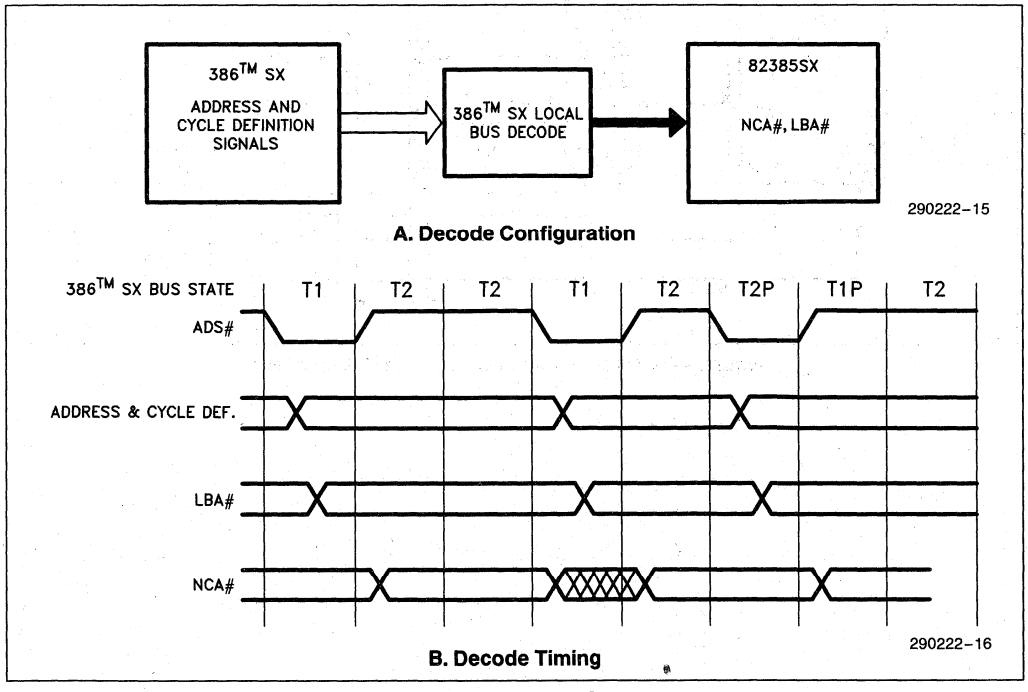


Figure 4-3. NCA#, LBA# Generation

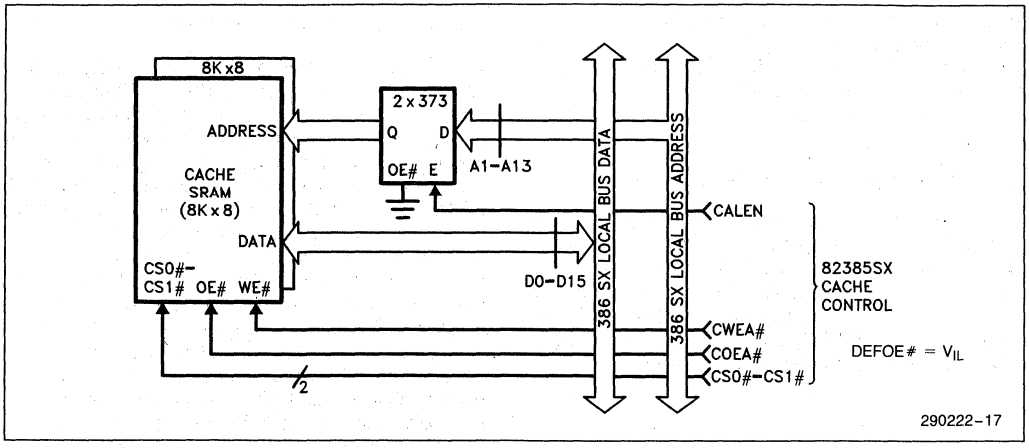


Figure 4-4A. Direct Mapped Cache without Data Buffers

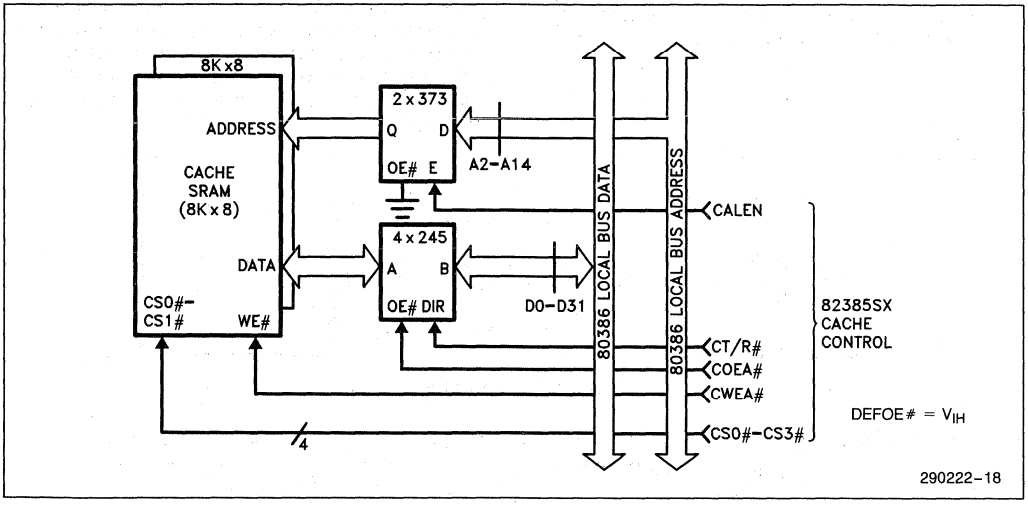


Figure 4-4B. Direct Mapped Cache with Data Buffers

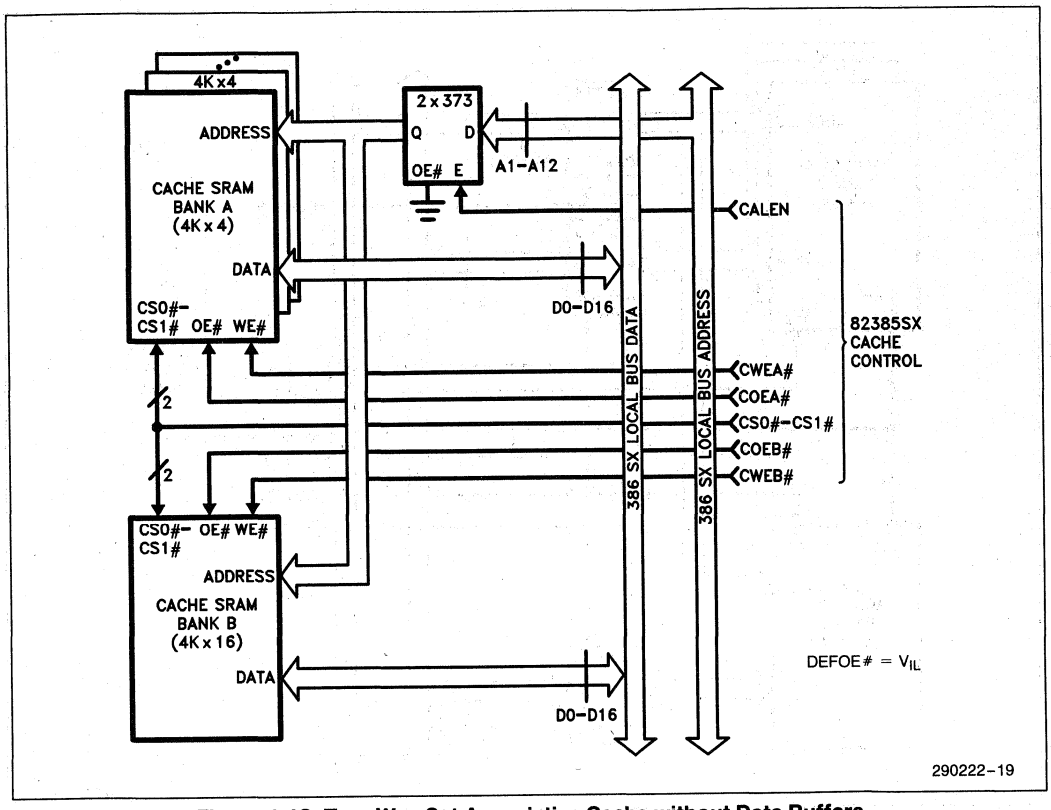


Figure 4-4C. Two-Way Set Associative Cache without Data Buffers

5

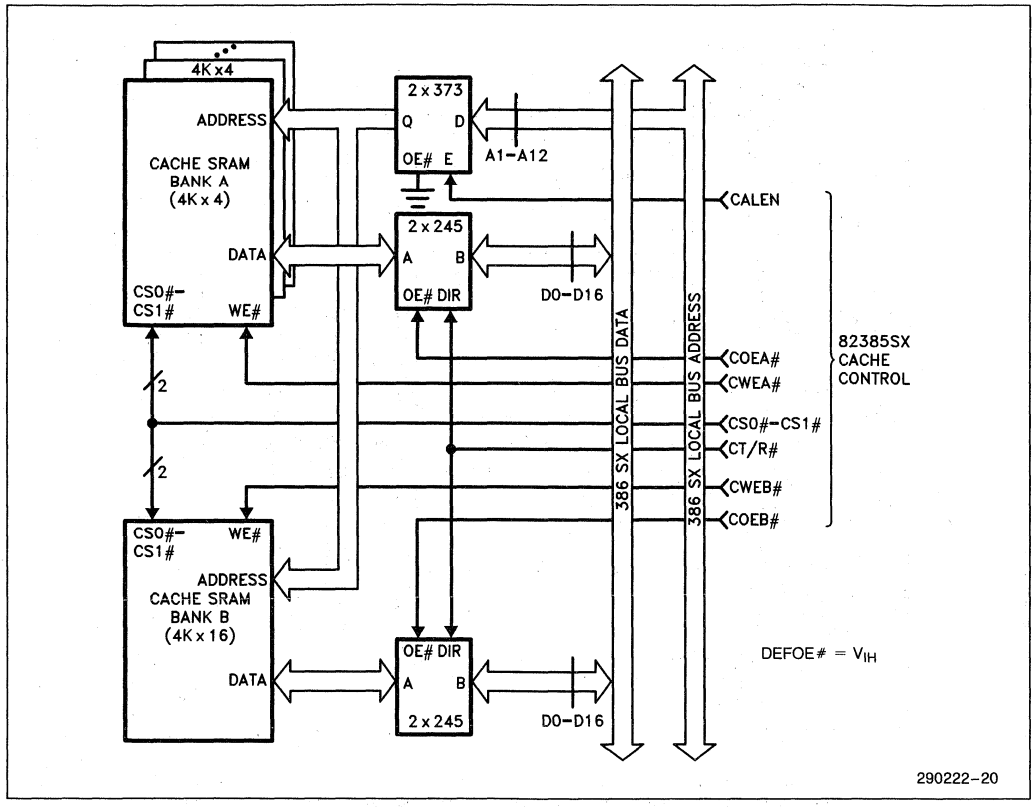
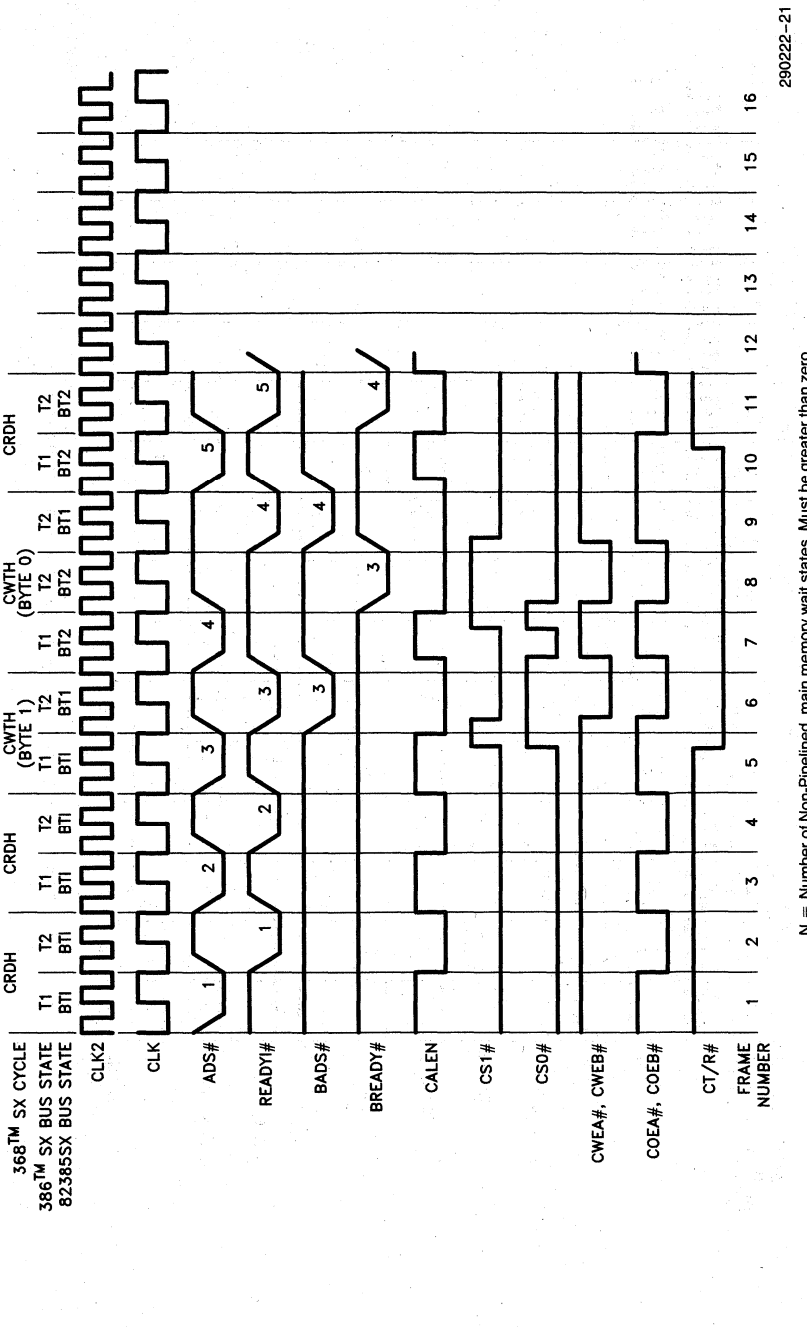


Figure 4-4D. Two-Way Set Associative Cache with Data Buffers

4.2.2 CACHE CONTROL . . . DIRECT MAPPED

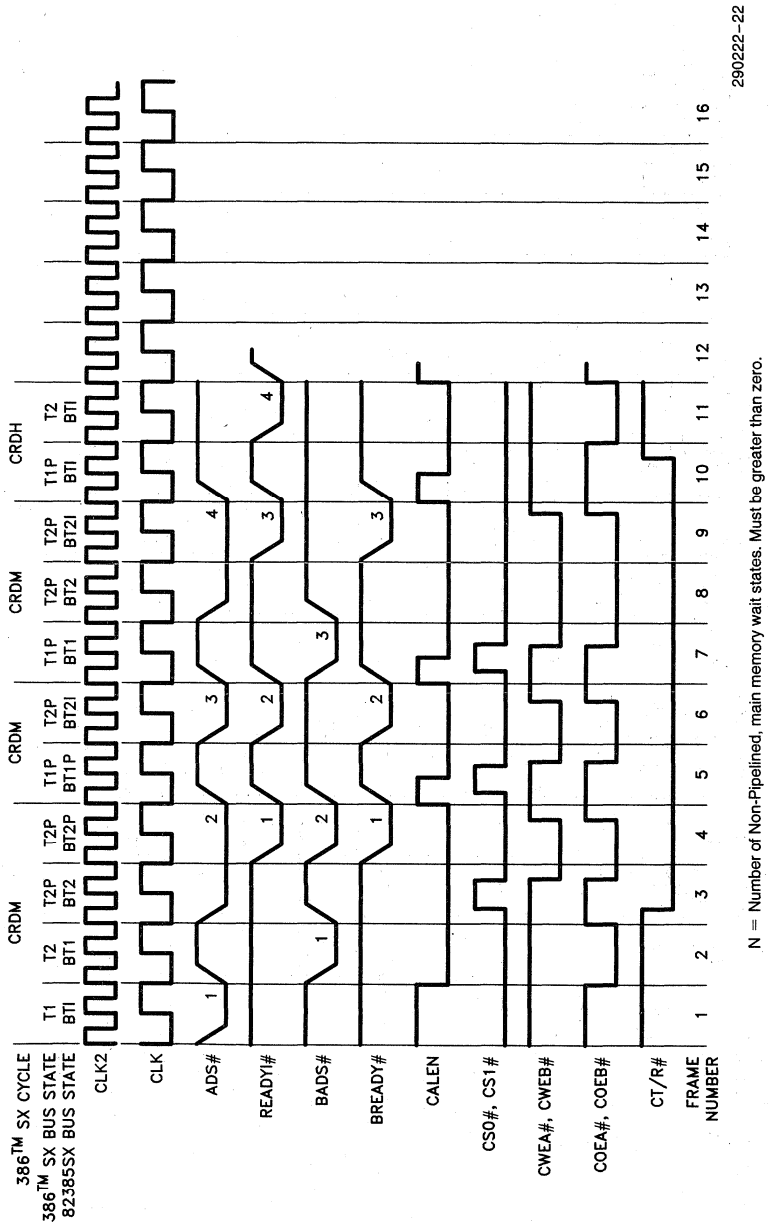
Figure 4-5A illustrates the timing of cache read and write hits, while Figure 4-5B illustrates cache updates. In a read hit, the cache output enables are driven from the beginning of T2 (cycle 1 of Figure 4-5A). If at the end of T1 the cycle is qualified as a cacheable read, the output enables are asserted on the assumption that the cycle will be a hit. (Driving the output enables before the actual hit/miss decision is made eases SRAM timing requirements.)

Cycle 1 of Figure 4-5B illustrates what happens when the assumption of a hit turns out to be wrong. Note that the output enables are asserted at the beginning of of T2, but then disabled at the end of T2. Once the output enables are inactive, the 82385SX turns the transceiver around (via CT/R#) and drives the write enables to begin the cache update cycle. Note in Figure 4-5B that once the 386 SX is in pipelined mode, the output enables need not be driven prior to a hit/miss decision, since the decision is made earlier via the pipelined address information.



NOTES:
CRDH = Cache Read Hit
CWTH = Cache Write Hit

Figure 4-5A. Cache Read and Write Cycles—Direct Mapped (N = 1)



NOTE:
CRDM = Cache Read Miss

Figure 4-5B. Cache Update Cycles—Direct Mapped (N = 1)

One consequence of driving the output enables low in a miss before the hit/miss decision is made is that since the cache starts driving the 386 SX data bus, the 82385SX cannot enable the 74646 transceiver (Figure 4-1) until after the cache outputs are disabled. (The timing of the 74646 control signals is described in the next chapter.) The result is that the 74646 cannot be enabled soon enough to support N=0 main memory ("N" was defined in Section 4.0 as the number of non-pipelined main memory wait states). This means that memory which can run with zero wait states in a non-pipelined cycle should not be mapped into cacheable memory. This should not present a problem, however, as a main memory system built with N=0 memory has no need of a cache. (The main memory is as fast as the cache.) Zero wait state memory can be supported if it is decoded as non-cacheable. The 82385SX knows that a cycle is non-cacheable in time not to drive the cache output enables, and can thus enable the 74646 sooner.

In a write hit, the 82385SX only updates the cache bytes that are meant to be updated as directed by the 386 SX byte enables. This prevents corrupting cache data in partial doubleword writes. Note in Figure 4-5A that the appropriate bytes are selected via the cache byte select lines CS0# and CS1#. In a read hit, both select lines are driven as the 386 SX will simply ignore data it does not need. Also, in a cache update (read miss), both selects are active in order to update the cache with a complete line (word).

4.2.3 CACHE CONTROL ... TWO-WAY SET ASSOCIATIVE

Figures 4-6A and 4-6B illustrate the timing of cache read hits, write hits, and updates for a two-way set associative cache. (Note that the cycle sequences are the same as those in Figure 4-5A and 4-5B.) In a cache read hit, only one bank on the other is enabled to drive the 386 SX data bus, so unlike the control of a direct mapped cache, the appropriate cache output enable cannot be driven until the outcome of the hit/miss decision is known. (This implies stricter SRAM timing requirements for a two-way set associative cache.) In write hits and read misses, only one bank or the other is updated.

4.3 387 SX Interface

The 387 SX Math Coprocessor interfaces to the 386 SX just as it would in a system without an 82385SX. The 387 SX READYO# output is logically "AND"ed along with all other 386 SX local bus ready sources (Figure 4-1), and the output is fed to the 387 SX READY#, 82385SX READYI#, and 386 SX READY# inputs.

The 386 SX uniquely addresses the 387 SX by driving M/IO# low and A23 high. The 82385SX decodes this internally and treats 387 SX accesses in the same way it treats 386 SX cycles in which LBA# is asserted, it ignores them.

5.0 82385SX LOCAL BUS AND SYSTEM INTERFACE

The 82385SX system interface is the 82385SX Local Bus, which presents a "386 SX-like" front end to the system. The system ties to it just as it would to a 386 SX. Although this 386 SX-like front end is functionally equivalent to a 386 SX, there are timing differences which can easily be accounted for in a system design.

The following is a description of the 82385SX system interface. After presenting the 82385SX bus state machine, the 82385SX bus signals are described, as are techniques for accommodating any differences between the 82385SX bus and 386 SX bus. Following this is a discussion of the 82385SX's condition upon reset.

5.1 The 82385SX Bus State Machine

5.1.1 MASTER MODE

Figure 5-1A illustrates the 82385SX bus state machine when the 82385SX is programmed in master mode. Note that it is almost identical to the 386 SX bus state machine, only the bus states are 82385SX bus states (BT1P, BTH, etc.) and the state transitions are conditioned by 82385SX bus inputs (BNA# BHOLD, etc.). Whereas a "pending request" to the 386 SX state machine indicates that the 386 SX execution or prefetch unit needs bus access, a pending request to the 82385SX state machine indicates that a 386 SX bus cycle needs to be forwarded to the system (read miss, non-cacheable read, write, etc.). The only difference between the state machines is that the 82385SX does not implement a direct BT1P-BT2P transition. If BNA# is asserted in BT1P, the resulting state sequence is BT1P-BT2I-BT2P. The 82385SX's ability to sustain a pipeline is not affected by the lack of this transition.

5.1.2 SLAVE MODE

The 82385SX's slave mode state machine (Figure 5-1B) is similar to the master mode machine except that now transitions are conditioned by BHLDA rather than BHOLD. (Recall that in slave mode, the roles of BHOLD and BHLDA are reversed from their master mode roles.) Figure 5-2 clarifies slave mode state machine operation. Upon reset, a slave mode

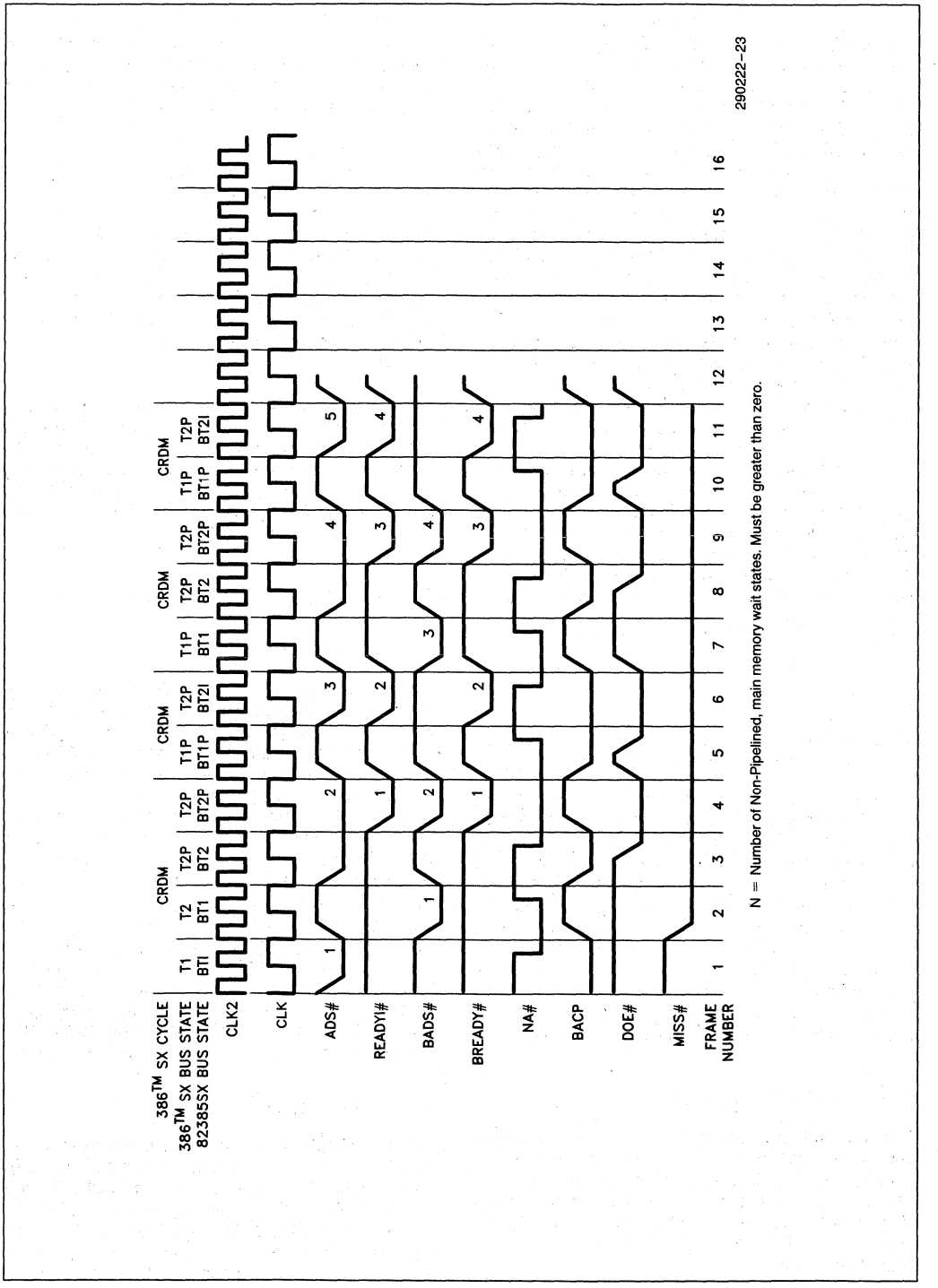


Figure 4-6A. Cache Read and Write Cycles—Two Way Associative (N = 1)

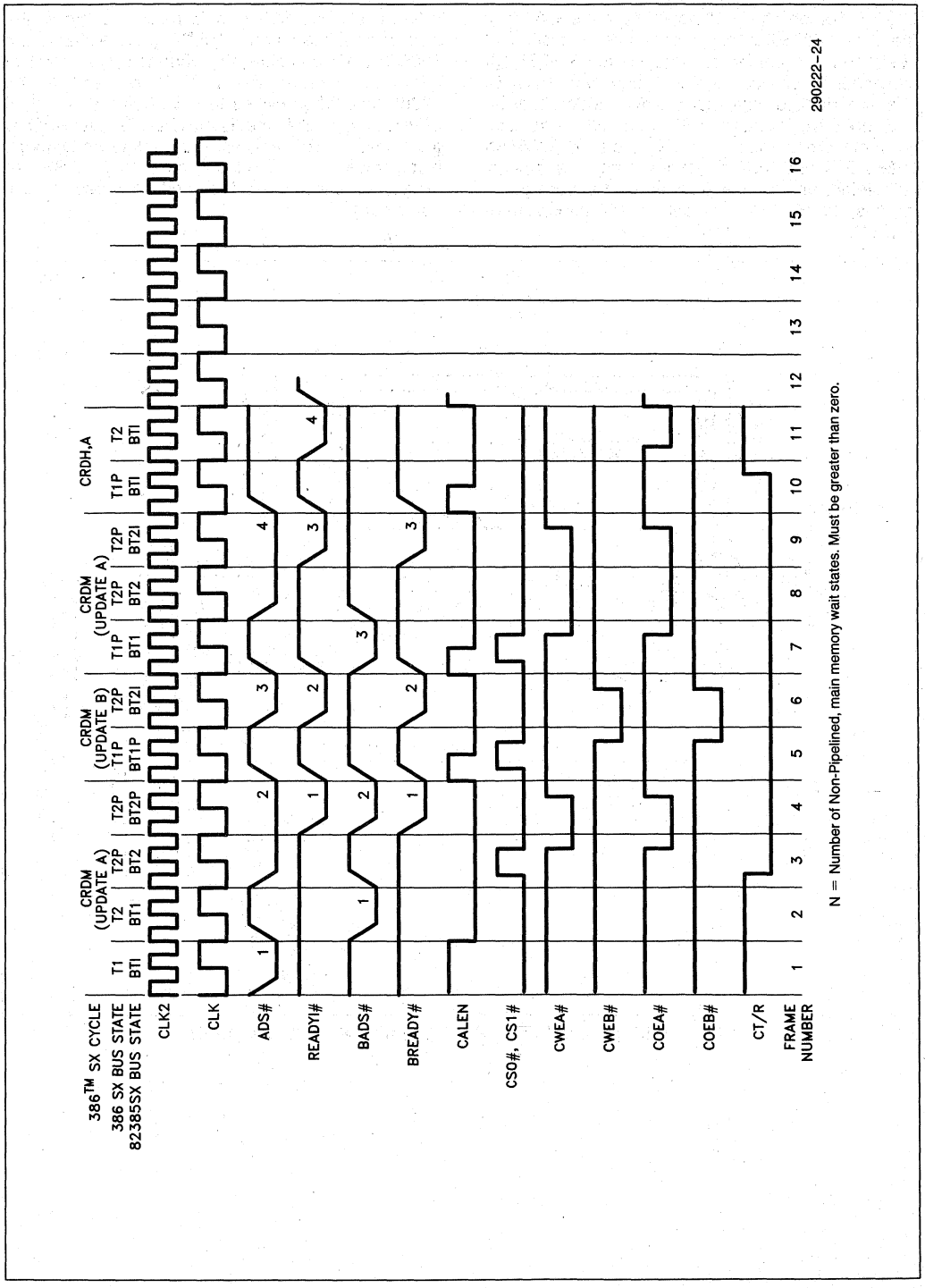


Figure 4-6B. Cache Update Cycles—Two Way Set Associative (N = 1)

82385SX enters the BTH state. When the 386 SX of the slave 82385SX subsystem has a cycle that needs to be forwarded to the system, the 82385SX moves to BTI and issues a hold request via BHOLD. It is important to note that a slave mode 82385SX does not drive the bus in a BTI state. When the master or bus arbiter returns BHLDA, the slave 82385SX enters BT1 and runs the cycle. When the cycle is completed, and if no additional requests are pending, the 82385SX moves back to BTH and disables BHOLD.

If, while a slave 82385SX is running a cycle, the master or arbiter drops BHLDA (Figure 5-2B), the 82385SX will complete the current cycle, move to BTH and remove the BHOLD request. If the 82385SX still had cycles to run when it was kicked off the bus, it will immediately assert a new BHOLD and move to BTI to await bus acknowledgement. Note, however, that it will only move to BTI if BHLDA is negated, insuring that the handshake sequence is completed.

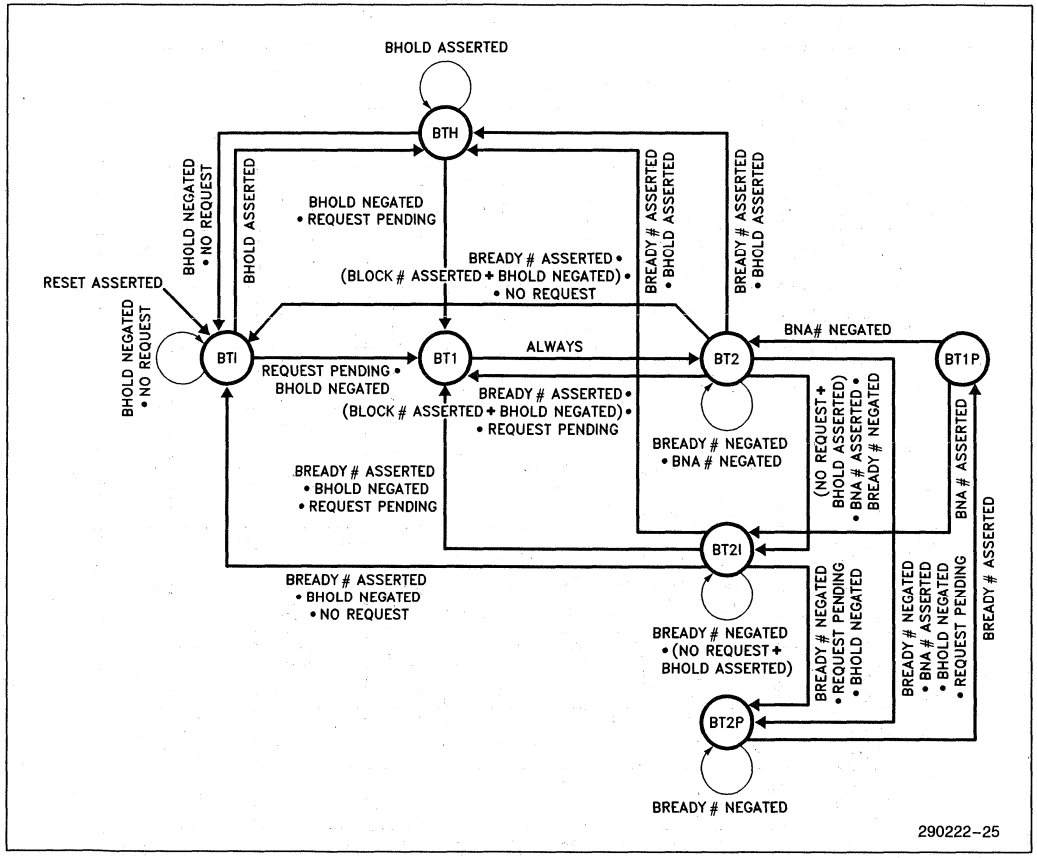


Figure 5-1A. 82385SX Local Bus State Machine—Master Mode

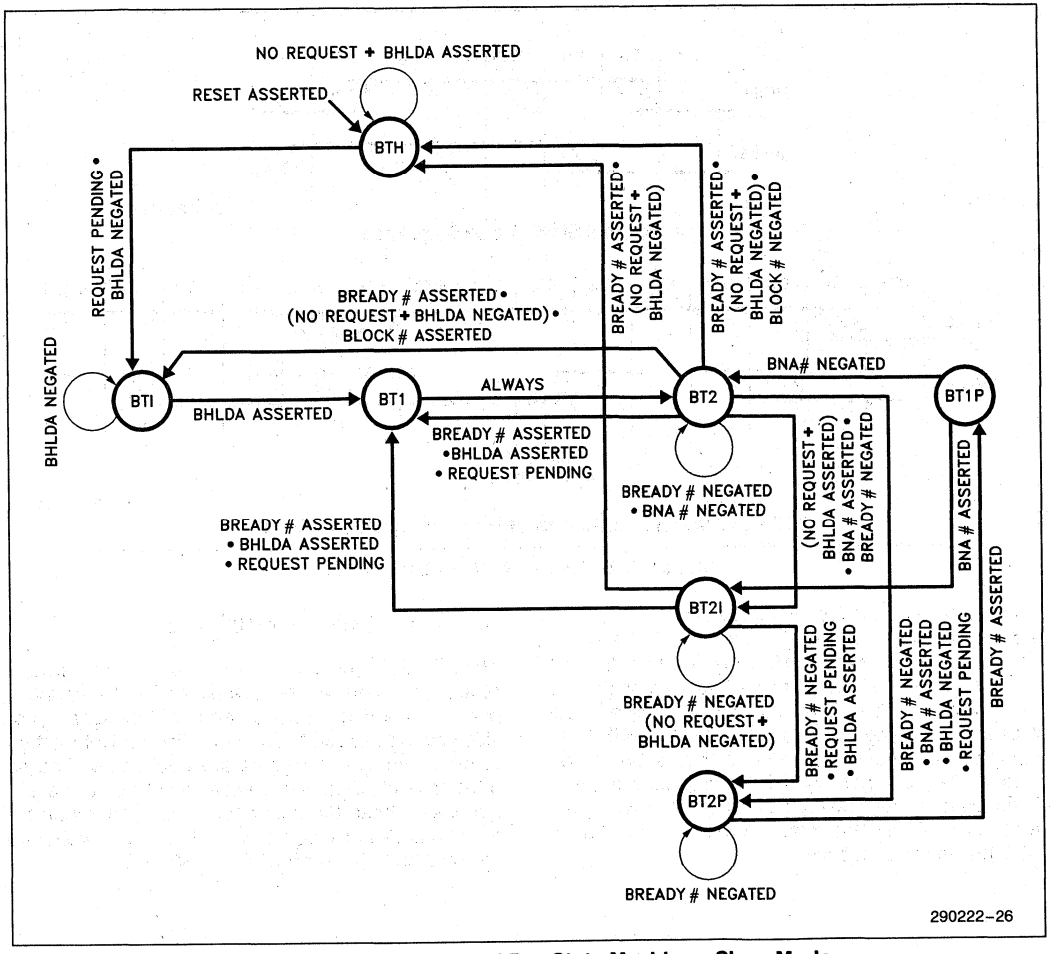
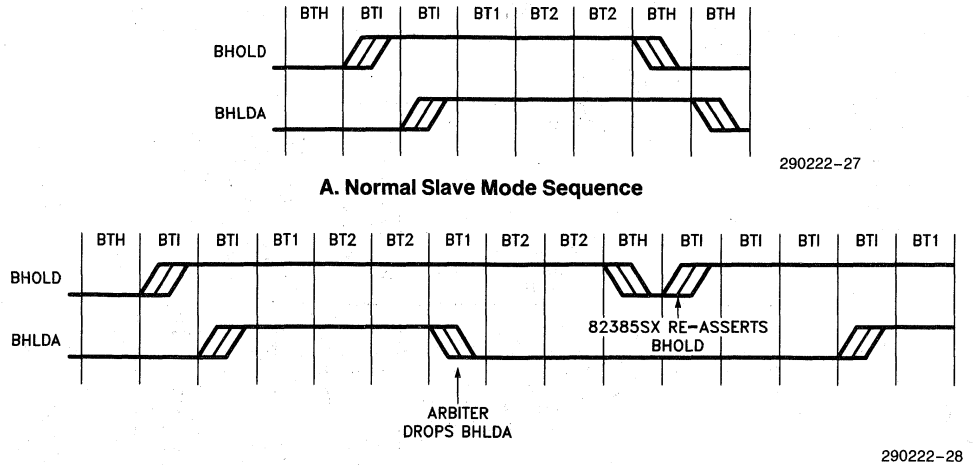


Figure 5-1B. 82385SX Local Bus State Machine—Slave Mode



A. Normal Slave Mode Sequence

B. Sequence of Events if Master or Arbiter Drops BHLDA

Figure 5-2. BHOLD/BHLDA—Slave Mode

There are several cases in which a slave 82385SX will not immediately release the bus if BHLDA is dropped. For example, if BHLDA is dropped during a BT2P state, the 82385SX has already committed to the next system bus pipelined cycle and will execute it before releasing the bus. Also, the 82385SX will complete a sequence of locked cycles before releasing the bus. This should not present any problems, as a properly designed arbiter will not assume that the 82385SX has released the bus until it sees BHOLD become inactive.

5.2 The 82385SX Local Bus

The 82385SX bus can be broken up into two groups of signals: those which have direct 386 SX counterparts, and additional status and control signals provided by the 82385SX. The operation and interaction of all 82385SX bus signals are depicted in Figures 5-3A through 5-3L for a wide variety of cycle sequences. These diagrams serve as a reference for the 82385SX bus discussion and provide insight into the dual bus operation of the 82385SX.

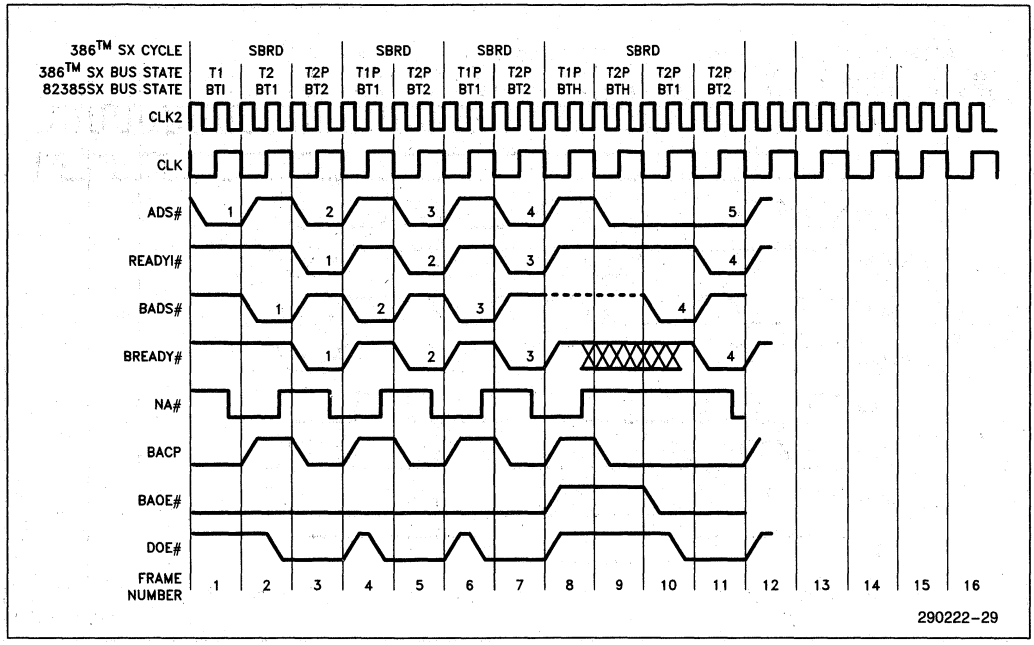


Figure 5-3A. Consecutive SBRD Cycles—(N = 0)

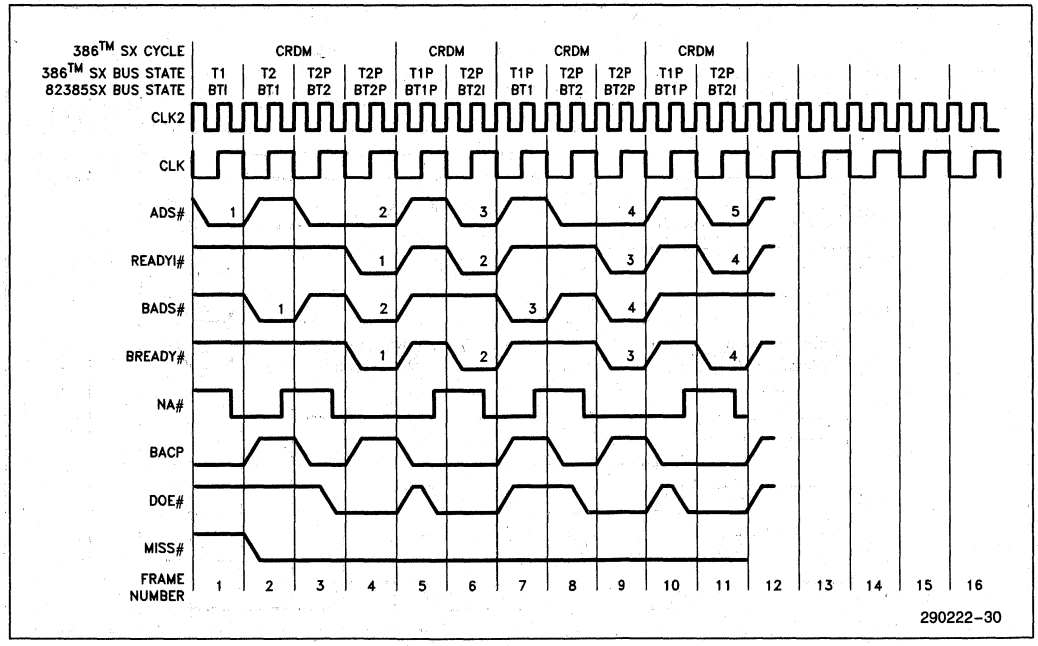


Figure 5-3B. Consecutive CRDM Cycles—(N = 1)

5

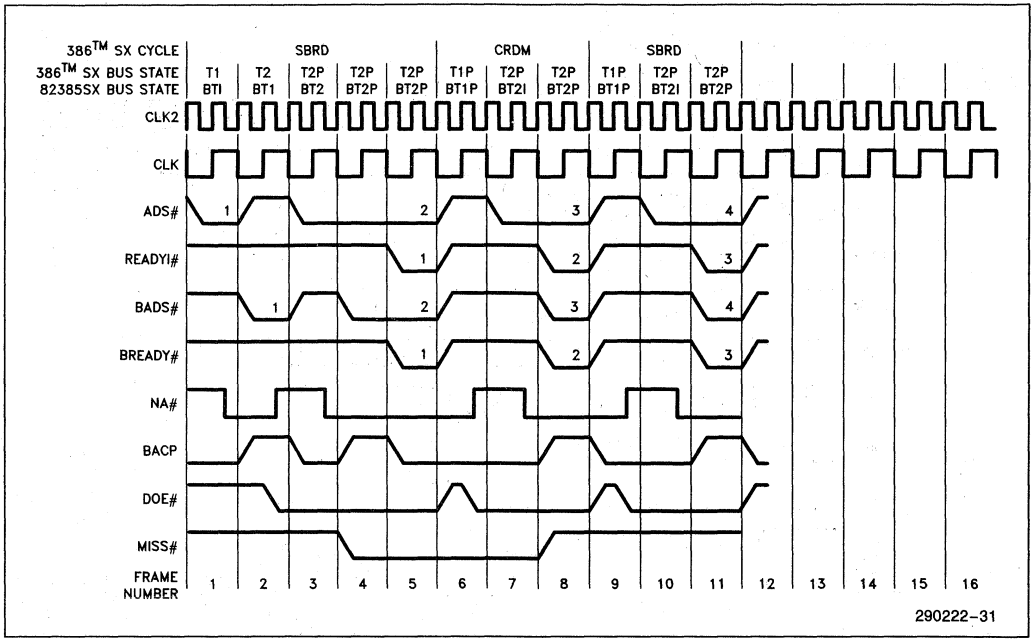


Figure 5-3C. SBRD, CRDM, SBRD—(N = 2)

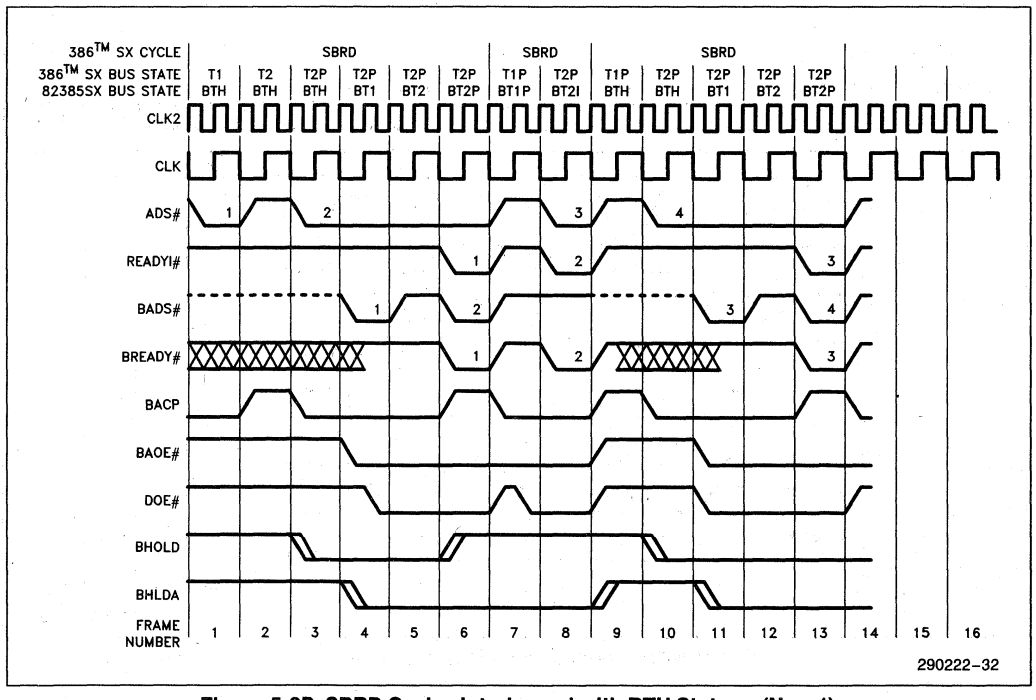


Figure 5-3D. SBRD Cycles Interleaved with BTH States—(N = 1)

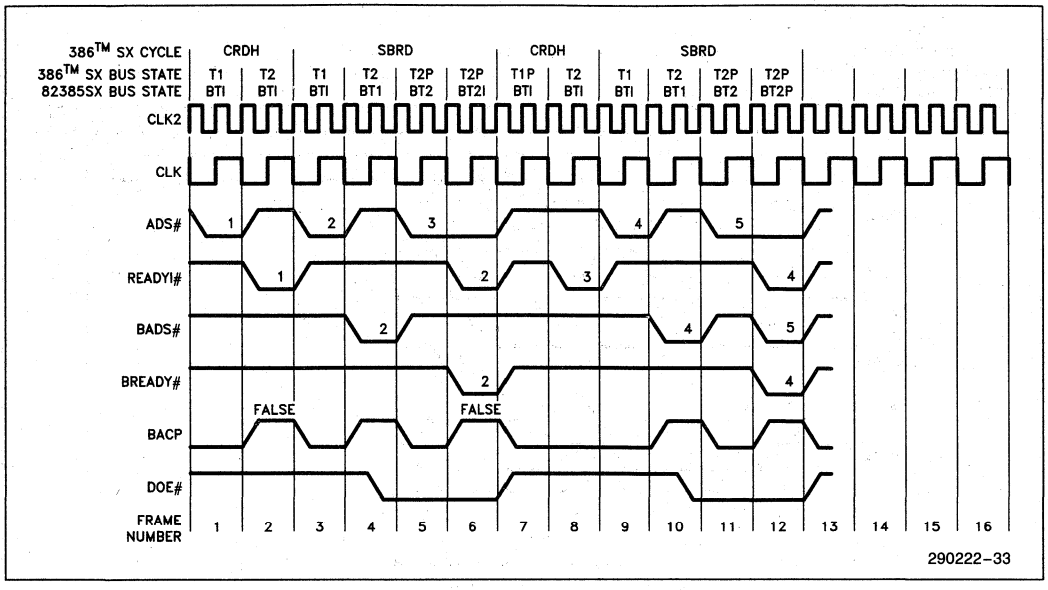


Figure 5-3E. Interleaved SBRD/CDRH Cycles—(N = 1)

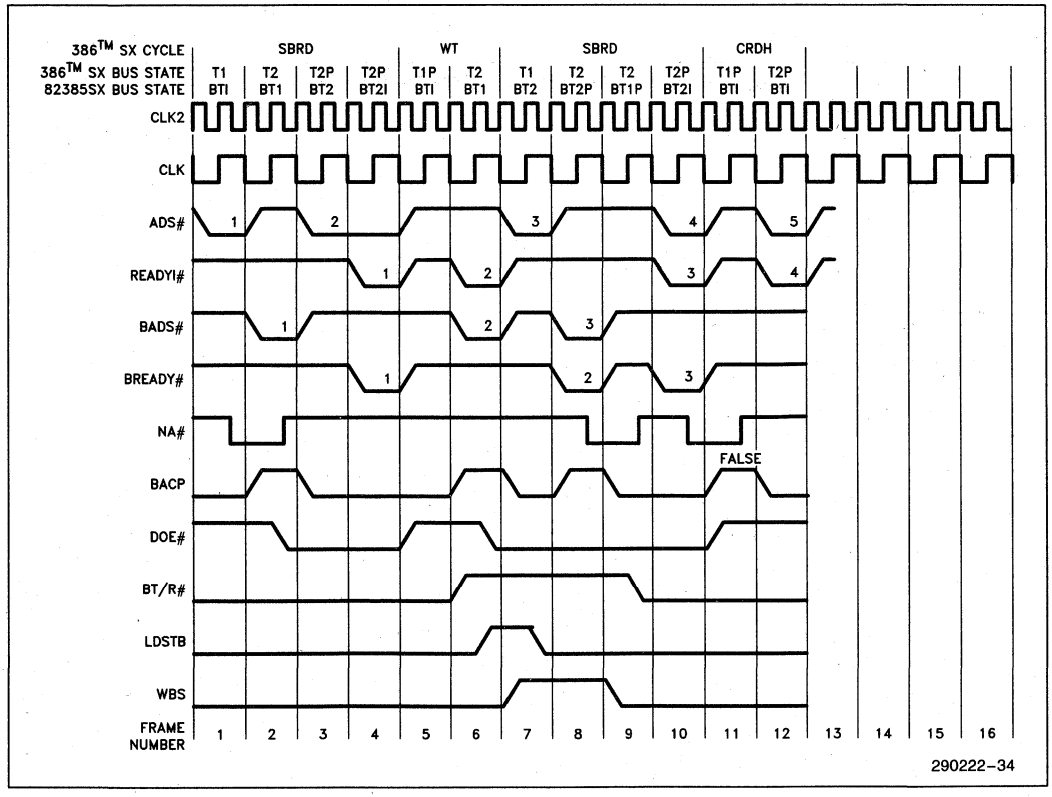


Figure 5-3F. SBRD, WT, SBRD, CRDH—(N = 1)

5

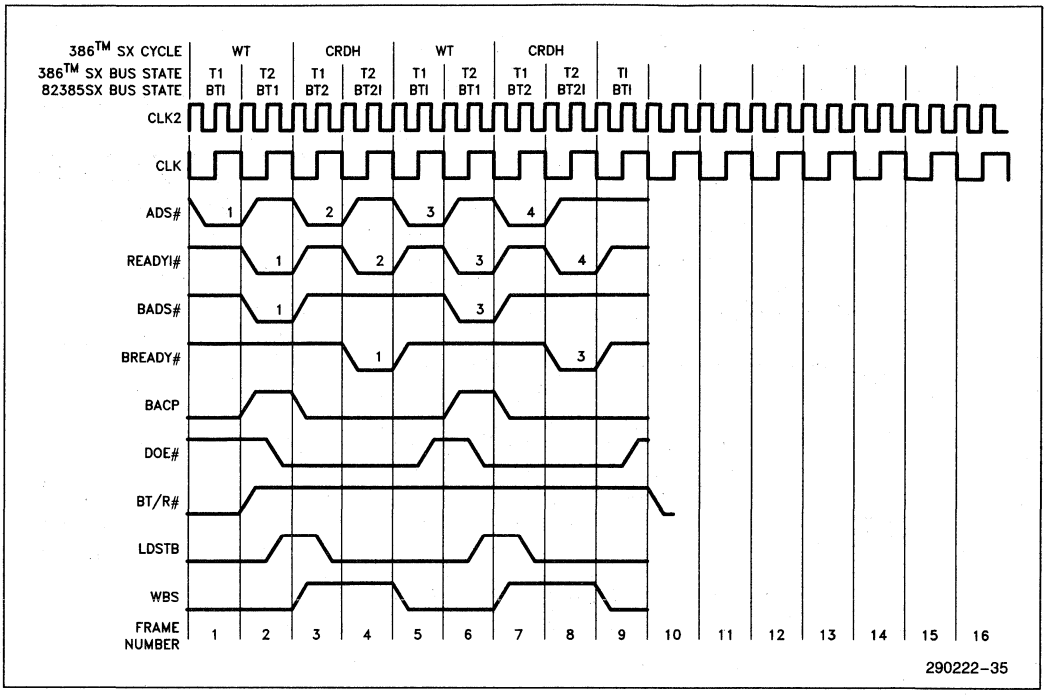


Figure 5-3G. Interleaved WT/CRDH Cycles—(N = 1)

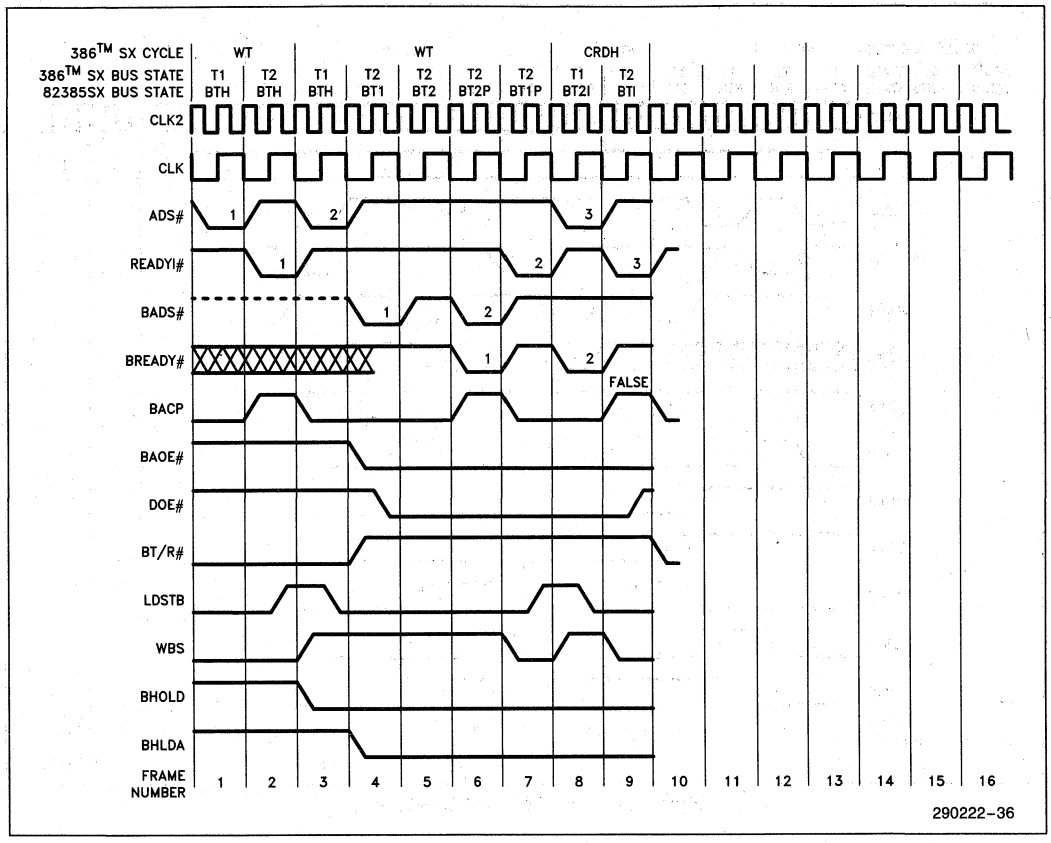


Figure 5-3H. WT, WT, CRDH—(N = 1)

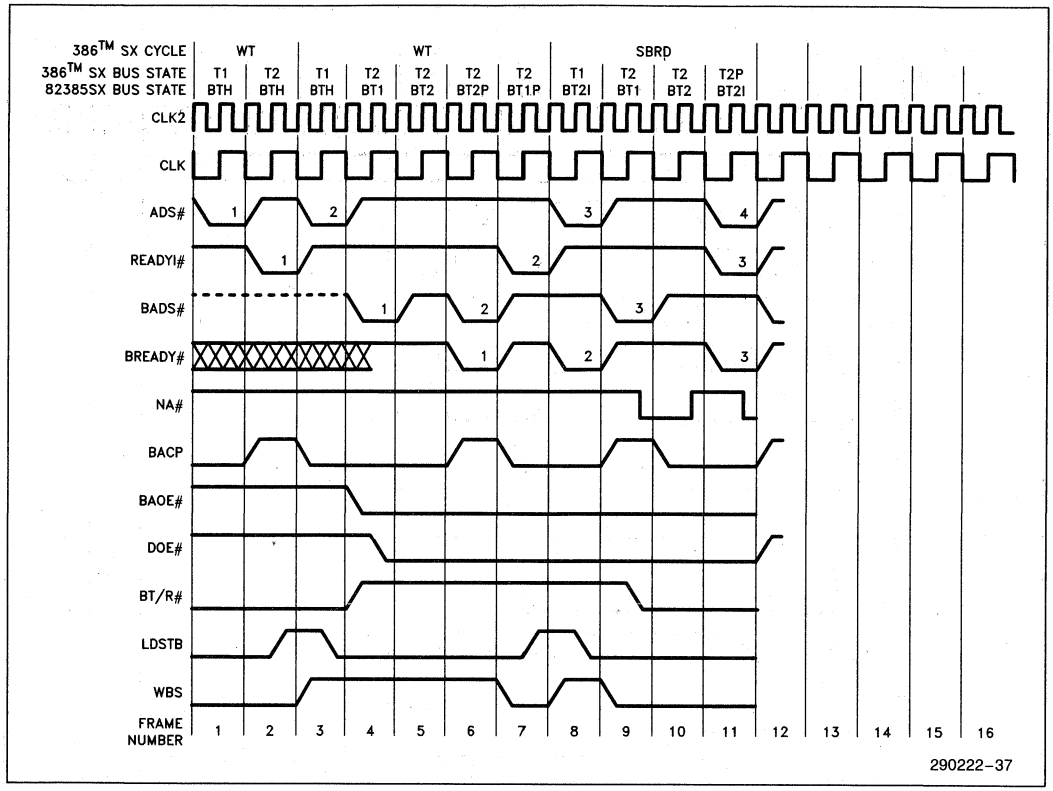


Figure 5-3I. WT, WT, SBRD—(N = 1)

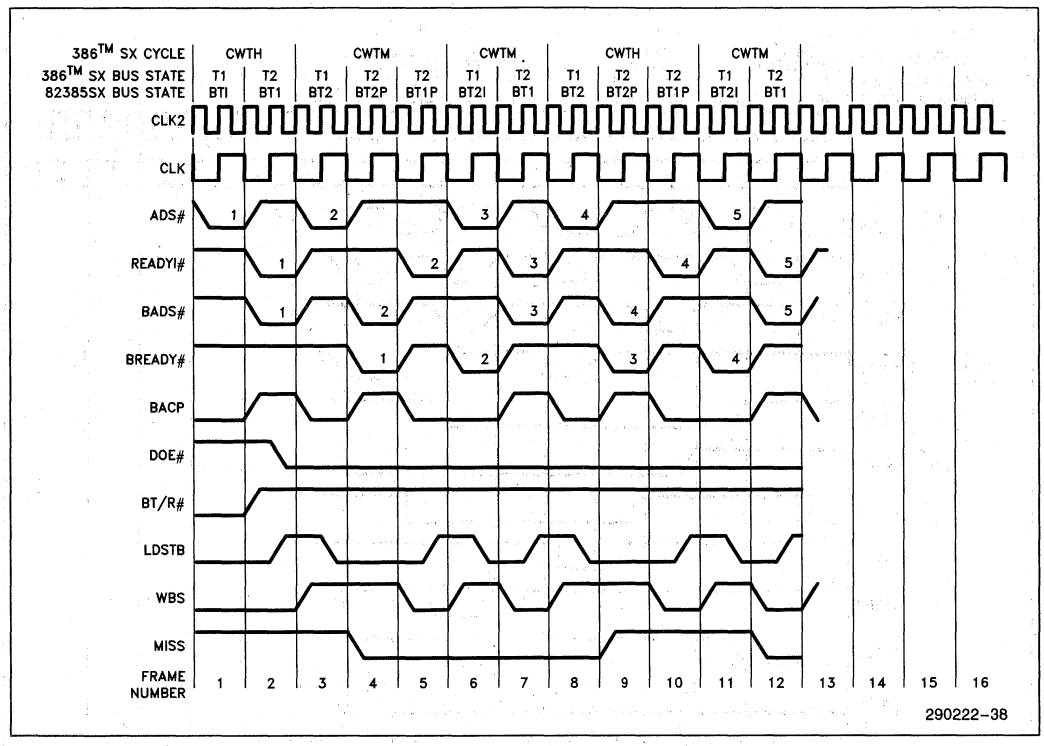


Figure 5-3J. Consecutive Write Cycles—(N = 1)

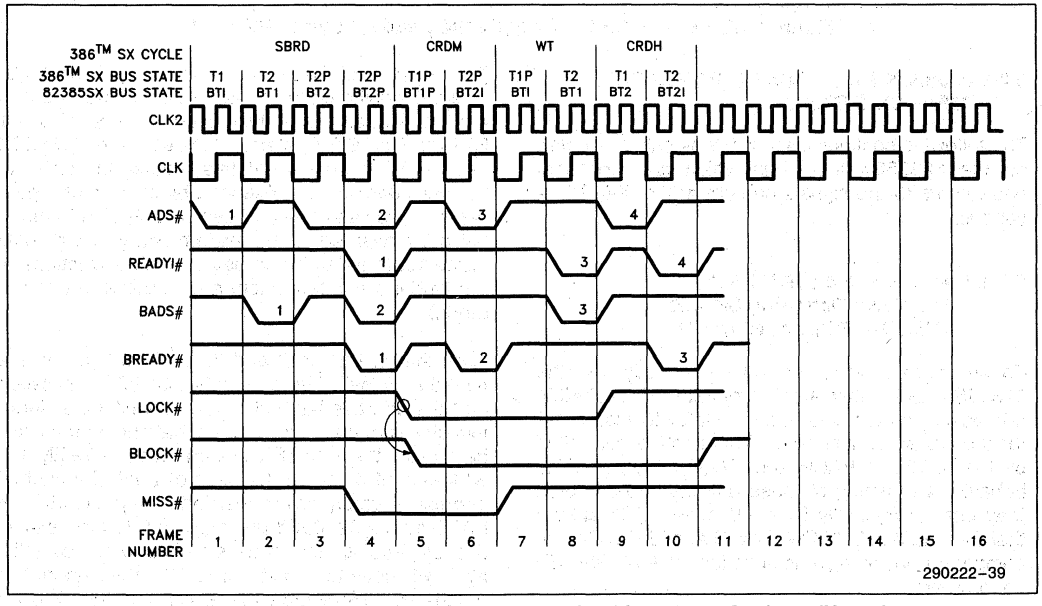


Figure 5-3K. LOCK#/BLOCK# in Non-Cacheable or Miss Cycles—(N = 1)

5

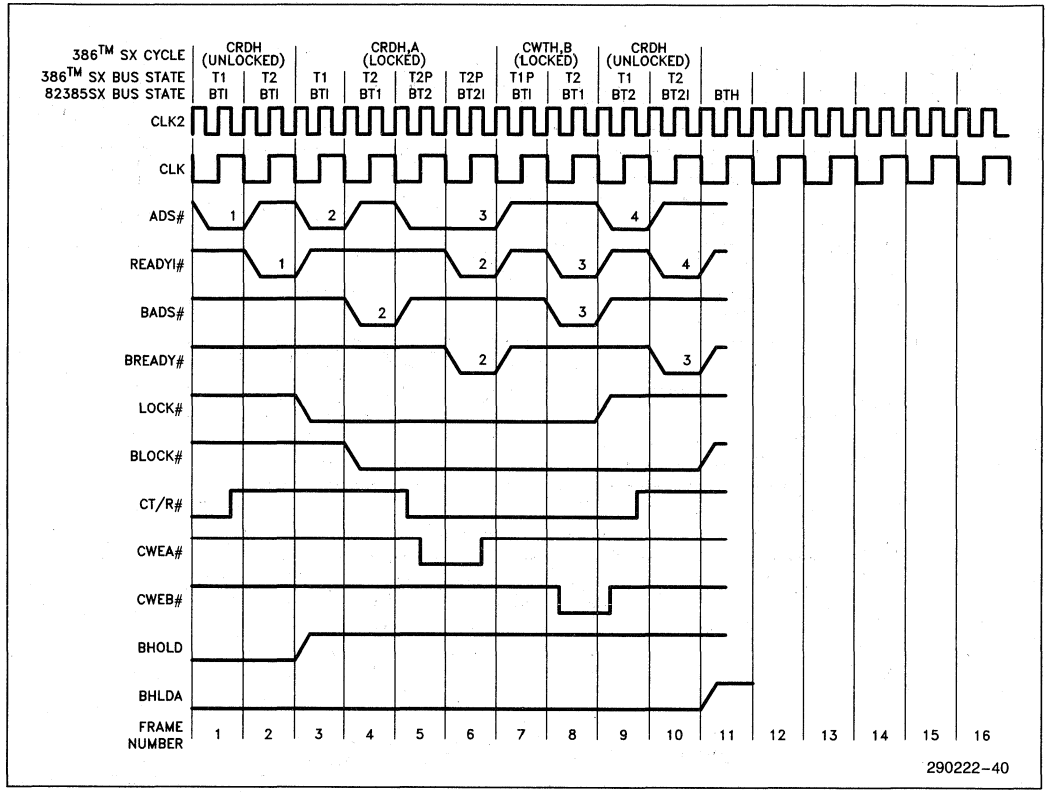


Figure 5-3L. LOCK#/BLOCK# in Cache Read Hit Cycle—(N = 1)

5.2.1 82385SX BUS COUNTERPARTS TO 386™ SX SIGNALS

The following sections discuss the signals presented on the 82385SX local bus which are functional equivalents to the signals present at the 386 SX local bus.

5.2.1.1 Address Bus (BA1–BA23) and Cycle Definition Signals (BM/IO#, BD/C#, BW/R#)

These signals are not driven directly by the 82385SX, but rather are the outputs of the 74374 address/cycle definition latch. (Refer to Figure 4-1 for the hardware interface.) This latch is controlled by the 82385SX BACP and BAOE# outputs. The behavior and timing of these outputs and the latch they control (typically F or AS series TTL) ensure that BA1–BA23, BM/IO#, BW/R#, and BD/C# are compatible in timing and function to their 386 SX counterparts.

The behavior of BACP can be seen in Figure 5-3B, where the rising edge of BACP latches and forwards the 386 SX address and cycle definition signals in a BT1 or first BT2P state. However, the 82385SX need not be the current bus master to latch the 386 SX address, as evidenced by cycle 4 of Figure 5-3A. In this case, the address is latched in frame 8, but not forwarded to the system (via BAOE#) until frame 10. (The latch and output enable functions of the 74374 are independent and invisible to one another.)

Note that in frames 2 and 6 the BACP pulses are marked "False". The reason is that BACP is issued and the address latched before the hit/miss determination is made. This ensures that should the cycle be a miss, the 82385SX bus can move directly into BT1 without delay. In the case of a hit, the latched address is simply never qualified by the assertion of BADS#. The 82385SX bus stays in BTI if there is no access pending (new cycle is a hit) and no bus activity. It will move to and stay in BT2I if the system has requested a pipelined cycle and the 82385SX does not have a pending bus access (new cycle is a hit).

5.2.1.2 Data Bus (BD0–BD15)

The 82385SX data bus is the system side of the 74646 latching transceiver. (See Figure 4-1.) This device is controlled by the 82385SX outputs LDSTB, DOE#, and BT/R#. LDSTB latches data in write cycles, DOE# enables the transceiver outputs, and BT/R# controls the transceiver direction. The interaction of these signals and the transceiver is such that BD0–BD15 behave just like their 386 SX counterparts. The transceiver is configured such that data flow in write cycles (A to B) is latched, and data flow in read cycles (B to A) is flow-through.

Although BD0–BD15 function just like their 386 SX counterparts, there is a timing difference that must be accommodated for in a system design. As mentioned above, the transceiver is transparent during read cycles, so the transceiver propagation delay must be added to the 386 SX data setup. In addition, the cache SRAM setup must be accommodated for in cache read miss cycles.

For non-cacheable reads the data setup is given by:

$$\text{Min BD0–BD15 Read Data Setup} \geq \text{386SX Min Data Setup} + \text{74646 B-to-A Max Propagation Delay}$$

The required BD0–BD15 setup in a cache read miss is given by:

$$\text{Min BD0–BD15 Read Data Setup} = \text{74646 B-to-A Max Propagation Delay} + \text{Cache SRAM Min Write Setup} + \text{One CLK2 Period} - \text{82385SX CWEA\# or CWEB\# Min Delay}$$

If a data buffer is located between the 386 SX data bus and the cache SRAMs, then its maximum propagation delay must be added to the above formula as well. A design analysis should be completed for every new design to determine actual margins.

A design can accommodate the increased data setup by choosing appropriately fast main memory DRAMs and data buffers. Alternatively, a designer may deal with the longer setup by inserting an extra wait state into cache read miss cycles. If an additional state is to be inserted, the system bus controller should sample the 82385SX MISS# output to distinguish read misses from cycles that do not require the longer setup. Tips on using the 82385SX MISS# signal are presented later in this chapter.

The behavior of LDSTB, DOE#, and BT/R# can be understood via Figures 5-3A through 5-3L. Note that in cycle 1 of Figure 5-3A (A non-cacheable system read), DOE# is activated midway through BT1, but in cycle 1 of Figure 5-3B (a cache read miss), DOE# is not activated until midway through BT2. The reason is that in a cacheable read cycle, the cache SRAMs are enabled to drive the 386 SX data bus before the outcome of the hit/miss decision (in anticipation of a hit.) In cycle 1 of Figure 5-3B, the assertion of DOE# must be delayed until after the 82385SX has disabled the cache output buffers. The result is that N=0 main memory should not be mapped into the cache.

5.2.1.3 Byte Enables (BBHE#, BBLE#)

These outputs are driven directly by the 82385SX, and are completely compatible in timing and function with their 386 SX counterparts. When a 386 SX cycle is forwarded to the 82385SX bus, the 386 SX byte enables are duplicated on BBHE# and BBLE#. The one exception is a cache read miss, during which BBHE# and BBLE# are both active regardless of the status of the 386 SX byte enables. This ensures that the cache is updated with a valid 16-bit entry.

5.2.1.4 Address Status (BADS#)

BADS# is identical in function and timing to its 386 SX counterpart. It is asserted in BT1 and BT2P states, and indicates that valid address and cycle definition (BA1–BA23, BBHE#, BBLE#, BM/IO#, BW/R#, BD/C#) information is available on the 82385SX bus.

5.2.1.5 Ready (BREADY#)

The 82385SX BREADY# input terminates 82385SX bus cycles just as the 386 SX READY# input terminates 386 SX bus cycles. The behavior of BREADY# is the same as that of READY#, but note in the A.C timing specifications that a cache read miss requires a longer BREADY# setup than do other cycles. This must be accommodated for in ready logic design.

5.2.1.6 Next Address (BNA#)

BNA# is identical in function and timing to its 386 SX counterpart. Note that in Figures 5-3A through 5-3L, BNA# is assumed asserted in every BT1P or first BT2 state. Along with the 82385SX's pipelining of the 386 SX, this ensures that the timing diagrams accurately reflect the full pipelined nature of the dual bus structure.

5.2.1.7 Bus Lock (BLOCK#)

The 386 SX flags a locked sequence of cycles by asserting LOCK#. During a locked sequence, the 386 SX does not acknowledge hold requests, so the

sequence executes without interruption by another master. The 82385SX forces all locked 386 SX cycles to run on the 82385SX bus (unless LBA# is active), regardless of whether or not the referenced location resides in the cache. In addition, a locked sequence of 386 SX cycles is run as a locked sequence on the 82385SX bus; BLOCK# is asserted and the 82385SX does not allow the sequence to be interrupted. Locked writes (hit or miss) and locked read misses affect the cache and cache directory just as their unlocked counterparts do. A locked read hit, however, is handled differently. The read is necessarily forced to run on the 82385SX local bus, and as the data returns from main memory, it is "re-copied" into the cache. (See Figure 5-3L.) The directory is not changed as it already indicates that this location exists in the cache. This activity is invisible to the system and ensures that semaphores are properly handled.

BLOCK# is asserted during locked 82385SX bus cycles just as LOCK# is asserted during locked 386 SX cycles. The BLOCK# maximum valid delay, however, differs from that of LOCK#, and this must be accounted for in any circuitry that makes use of BLOCK#. The difference is due to the fact that LOCK#, unlike the other 386 SX cycle definition signals, is not pipelined. The situation is clarified in Figure 5-3K. In cycle 2 the state of LOCK# is not known before the corresponding system read starts (Frame 4 and 5). In this case, LOCK# is asserted at the beginning of T1P, and the delay for BLOCK# to become active is the delay of LOCK# from the 386 SX plus the propagation delay through the 82385SX. This occurs because T1P and the corresponding BT1P are concurrent (Frame 5). The result is that BLOCK# should not be sampled at the end of BT1P. The first appropriate sampling point is midway through the next state, as shown in Frame 6. In Figure 5-3L, the maximum delay for BLOCK# to become valid in Frame 4 is the same as the maximum delay for LOCK# to become valid from the 386 SX. This is true since the pipelining issue discussed above does not occur.

The 82385 should negate BLOCK# after: BREADY# of the last 82385 Locked Cycle was asserted AND LOCK# turns inactive.

This means that in a sequence of cycles which begins with a 82385 Locked Cycle and goes on with all the possible Locked Cycles (other 82385 cycles, idles, and local cycles), while LOCK# is continuously active, the 82385 will maintain BLOCK# active continuously. Another implication is that in a Locked Posted Write Cycle followed by non-locked sequence, BLOCK# is negated one CLK after BREADY# of the write cycle. In other 82385 Locked Cycles, followed by non-locked sequences, BLOCK# is negated one CLK after LOCK# is negated, which occurs two CLKs after BREADY# is asserted. In the last case BLOCK# active moves by one CLK to the non-locked sequence.

The arbitration rules of Locked Cycles are:

MASTER MODE:

BHOLD input signal is ignored when BLOCK# or internal lock (16-bit non-aligned cycle) are active. BHLDA output signal remains inactive, and BAOE# output signal remains active at that time interval.

SLAVE MODE:

The 82385 does not relinquish the system bus if BLOCK# or internal lock are active. The BHOLD output signal remains active when BLOCK# or internal lock is active plus one CLK. The BHLDA input signal is ignored when BLOCK# or the internal lock is active plus one CLK. This means the 82385 slave does not respond to BHLDA inactivation. The BAOE# output signal remains active during the same time interval.

5.2.2 ADDITIONAL 82385SX BUS SIGNALS

The 82385SX bus provides two status outputs and one control input that are unique to cache operation and thus have no 386 SX counterparts. The outputs are MISS# and WBS, and the input is FLUSH.

5.2.2.1 Cache Read/Write Miss Indication (MISS#)

MISS# can be thought of as an extra 82385SX bus cycle definition signal similar to BM/IO#, BW/R#, and BD/C#, that distinguishes cacheable read and write misses from other cycles. MISS#, like the other definition signals, becomes valid with BADS# (BT1 or first BT2P). The behavior of MISS# is illustrated in Figures 5-3B, 5-3C, and 5-3J. The 82385SX floats MISS# when another master owns the bus, allowing multiple 82385SXs to share the same node in multi-cache systems. MISS# should thus be lightly pulled up (~20K) to keep it negated during hold (BTH) states.

MISS# can serve several purposes. As discussed previously, the BD0–BD15 and BREADY# setup times in a cache read miss are longer than in other cycles. A bus controller can distinguish these cycles by gating MISS# with BW/R#. MISS# may also prove useful in gathering 82385SX system performance data.

5.2.2.2 WRITE BUFFER STATUS (WBS)

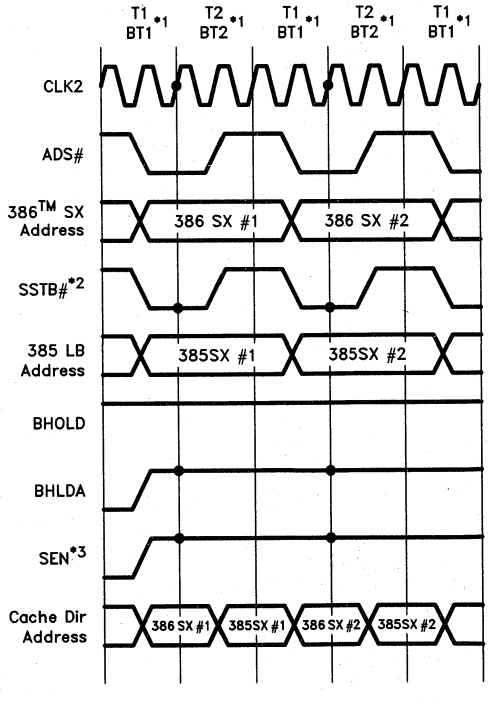
WBS is activated when 386 SX write cycle data is latched into the 74676 latching transceiver (via LDSTB). It is deactivated upon completion of the write cycle on the 82385SX bus when the 82385SX sees the BREADY# signal. WBS behavior is illustrated in Figures 5-3F through 5-3J, and potential applications are discussed in Chapter 3.

5.2.2.3 Cache Flush (FLUSH)

FLUSH is an 82385SX input which is used to reset all tag valid bits within the cache directory. The FLUSH input must be kept active for at least 4 CLK (8 CLK2) periods to complete the directory flush. Flush is generally used in diagnostics but can also be used in applications where snooping cannot guarantee coherency.

5.3 Bus Watching (Snoop) Interface

The 82385SX's bus watching interface consists of the snoop address (SA1–SA23), snoop strobe (SSTB#), and snoop enable (SEN) inputs. If masters reside at the system bus level, then the SA1–SA23 inputs are connected to the system address lines and SEN the system bus memory write command. SSTB# indicates that a valid address is present on the system bus. Note that the snoop bus inputs are synchronous, so care must be taken to ensure that they are stable during their sample windows. If no master resides beyond the 82385 bus level, then the 82385 inputs SA1–SA23, SEN, and SSTB# can respectively tie directly to BA1–BA23, BW/R#, and BADS# of the other system bus master (see Figure 5.5). However, it is recommended that SEN be driven by the logical "AND" of BW/R# and BM/IO# so as to prevent I/O writes from unnecessarily invalidating cache data.



NOTE:
*1. These states are induced by another System Bus master.
*2. SSTB# on the 82385SX is tied directly to BADS# of the System Bus master.
*3. SEN on the 82385SX is tied directly to BW/R# of the System Bus master.

Figure 5.4. Interleaved Snoop and 386™ SX Accesses to the Cache Directory

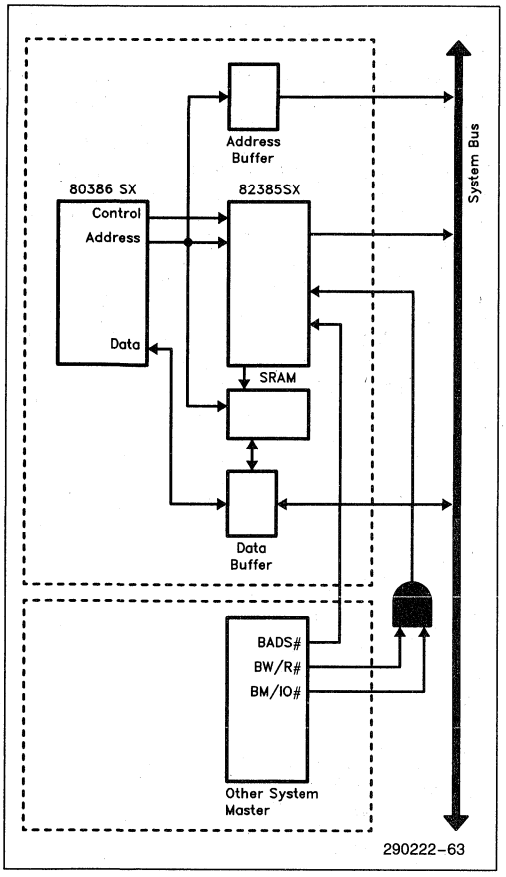


Figure 5.5. Snooping Connections in a Multi Master Environment

When the 82385SX detects a system write by another master and the conditions in Figure 5.4 are met: CLK2 PHI1 rising (CLK falling), BHLDA asserted, SEN asserted, SSTB# asserted, it internally latches SA1–SA23 and runs a cache look-up to see if the altered main memory location is duplicated in the cache. If yes (a snoop hit), the line valid bit associated with that cache entry is cleared. An important feature of the 82385SX is that even the 386 SX is running zero wait state hits out of the cache, all snoops are serviced. This is accomplished by time multiplexing the cache directory between the 386 SX address and the latched system address. If the SSTB# signal occurs during an 82385SX comparison cycle (for the 386 SX), the 386 SX cycle has the highest priority in accessing the cache directory. This takes the first of the two 386 SX states. The other state is then used for the snoop comparison. This worst case example, depicted in Figure 5.4, shows the 386 SX running zero wait state hits on the 386 SX local bus, and another master running zero wait state writes on the 82385SX bus. No snoops are missed, and no performance penalty incurred.

5.4 Reset Definition

Table 5-1 summarizes the states of all 82385SX outputs during reset and initialization. A slave mode 82385SX tri-states its "386 SX-like" front end. A master mode 82385SX emits a pulse stream on its BACP output. As the 386 SX address and cycle definition lines reach their reset values, this stream will latch the reset values through to the 82385SX bus.

Table 5-1. Pin State during RESET and Initialization

Output Name	Signal Level during RESET and Initialization	
	Master Mode	Slave Mode
NA#	High	High
READY0#	High	High
BRDYEN#	High	High
CALEN	High	High
CWEA#–CWEB#	High	High
CS0#, CS1#	Low	Low
CT/R#	High	High
COEA#–COEB#	High	High
BADS#	High	High Z
BBHE#, BBLE#	386 BE#	High Z
BLOCK#	High	High Z
MISS#	High	High Z
BACP	Pulse(1)	Pulse
BAOE#	Low	High
BT/R#	Low	Low
DOE#	High	High
LDSTB	Low	Low
BHOLD	—	Low
BHLDA	Low	—
WBS	Low	Low

NOTE:
1. In Master Mode, BAOE# is low and BACP emits a pulse stream during reset. As the 386 SX address and cycle definition signals reach their reset values, the pulse stream on BACP will latch these values through to the 82385SX local bus.

6.0 82385SX SYSTEM DESIGN CONSIDERATIONS

6.1 Introduction

This chapter discusses techniques which should be implemented in an 82385SX system. Because of the high frequencies and high performance nature of the 386 SX CPU/82385SX system, good design and layout techniques are necessary. It is always recommended to perform a complete design analysis of new system designs.

6.2 Power and Grounding

6.2.1 POWER CONNECTIONS

The 82385SX utilizes 8 power (V_{CC}) and 10 ground (V_{SS}) pins. All V_{CC} and V_{SS} pins must be connected to their appropriate plane. On a printed circuit board, all V_{CC} pins must be connected to the power plane and all V_{SS} pins must be connected to the ground plane.

6.2.2 POWER DECOUPLING

Although the 82385SX itself is generally a "passive" device in that it has a few output signals, the cache subsystem as a whole is quite active. Therefore, many decoupling capacitors should be placed around the 82385SX cache subsystem.

Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the decoupling capacitors and their respective devices as much as possible. Capacitors specifically for PGA packages are also commercially available, for the lowest possible inductance.

6.2.3 RESISTOR RECOMMENDATIONS

Because of the dual structure of the 82385SX subsystem (386 SX Local Bus and 82385SX Local Bus), any signals which are recommended to be pulled up will be respective to one of the busses. The following sections will discuss signals for both busses.

6.2.3.1 386 SX LOCAL BUS

For typical designs, the pullup resistors shown in Table 6-1 are recommended. This table correlates to Chapter 7 of the 386 SX Data Sheet. However, particular designs may have a need to differ from the listed values. Design analysis is recommended to determine specific requirements.

6.2.3.2 82835SX Local Bus

Pullup resistor recommendations for the 82385SX Local Bus signals are shown in Table 6-2. Design analysis is necessary to determine if deviations to the typical values given are needed.

Table 6-1. Recommended Resistor Pullups to V_{CC} (386™ SX Local Bus)

Pin and Signal	Pullup Value	Purpose
ADS# PGA E13 PQFP 123	20 KΩ ±10%	Lightly Pull ADS# Negated for 386 SX Hold States
LOCK# PGA F13 PQFP 118	20 KΩ ±10%	Lightly Pull LOCK# Negated for 386 SX Hold States

Table 6-2. Recommended Resistor Pullups to V_{CC} (82385SX Local Bus)

Signal and Pin	Pullup Value	Purpose
BADS# PGA N9 PQFP 89	20 KΩ ±10%	Lightly Pull BADS# Negated for 82385SX Hold States
BLOCK# PGA P9 PQFP 86	20 KΩ ±10%	Lightly Pull BLOCK# Negated for 82385SX Hold States
MISS# PGA N8 PQFP 85	20 KΩ ±10%	Lightly Pull MISS# Negated for 82385SX Hold States

6.3 82385SX Signal Connections

6.3.1 CONFIGURATION INPUTS

The 82835 configuration signals (M/S#, 2W/D#, DEFOE#) must be connected (pulled up) to the appropriate logic level for the system design. There is also a reserved 82385SX input which must be tied to the appropriate level. Refer to Table 6-3 for the signals and their required logic level.

Table 6-3. 82385SX Configuration Inputs Logic Levels

Pin and Signal	Logic Level	Purpose
M/S# PGA B13 PQFP 124	High	Master Mode Operation
	Low	Slave Mode Operation
2W/D# PGA D12 PQFP 127	High	2-Way Set Associative
	Low	Direct Mapped
Reserved PGA L14 PQFP 102	High	Must be tied to V_{CC} via a pull-up for proper functionality
DEFOE# PGA A14 PQFP 128	N/A	Define Cache Output Enable. Allows use of any SRAM.

NOTE:
The listed 82385SX pins which need to be tied high should use a pull-up resistor in the range of 5 KΩ to 20 KΩ.

6.3.2 CLK2 and RESET

The 82385SX has two inupts to which the 386 SX CLK2 signal must be connected. One is labeled CLK2 (82385SX pin C13) and the other is labeled BCLK2 (82385SX pin L13). These two inputs must be tied together on the printed circuit board.

The 82385SX also has two reset inputs. RESET (82385SX pin D13) and BRESET (82385SX pin K12) must be connected on the printed circuit board.

6.4 Unused Pin Requirements

For reliable operation, ALWAYS connect unused inputs to a valid logic level. As is the case with most other CMOS processes, a floating input will increase the current consumption of the component and give an indeterminate state to the component.

6.5 Cache SRAM Requirements

The 82385SX offers the option of using SRAMs with or without an output enable pin. This is possible by inserting a transceiver between the SRAMs and the 386 SX local data bus and strapping DEFOE# to the appropriate logic level for a given system configuration. This transceiver may also be desirable in a system which has a very heavily loaded 386 SX local data bus. The following sections discuss the SRAM requirements for all cache configurations.

6.5.1 CACHE MEMORY WITHOUT TRANSCEIVERS

As discussed in Section 3.2, the 82385SX presents all of the control signals necessary to access the cache memory. The SRAM chip selects, write enables, and output enables are driven directly by the 82385SX. Table 6-4 lists the required SRAM specifications. These specifications allow for zero margins. They should be used as guides for the actual system design.

Table 6-4. SRAM Specs for Non-Buffered Cache Memory

SRAM Spec Requirements				
	Direct Mapped		2-Way Set Associative	
	16 MHz	20 MHz	16 MHz	20 MHz
Read Cycle Requirements				
Address Access (MAX)	64 ns	44 ns	62 ns	42 ns
Chip Select Access (MAX)	76	56	76	56
OE# to Data Valid (MAX)	25	19	19	14
OE# to Data Float (MAX)	20	20	20	20
Write Cycle Requirements				
Chip Select to End of Write (MIN)	40	30	40	30
Address Valid to End of Write (MIN)	58	42	56	40
Write Pulse Width (MIN)	40	30	40	30
Data Setup (MAX)	—	—	—	—
Data Hold (MIN)	4	4	4	4

6.5.2 CACHE MEMORY WITH TRANSCEIVERS

To implement an 82385SX subsystem using cache memory transceivers, COEA# or COEB# must be used as output enable signals for the transceivers and DEFOE# must be appropriately strapped for proper COEx# functionality (since the cache SRAM transceivers must be enabled for writes as well as reads). DEFOE# must be tied high when using cache SRAM transceivers. In a 2-way set associative organization, COEA# enables the transceiver for bank A and COEB# enables the bank B transceiver. A direct mapped cache may use either COEA# or COEB# to enable the transceiver. Table 6-5 lists the required SRAM specifications. These specifications allow for zero margin. They should be used as guides for the actual system design.

7.0 SYSTEM TEST CONSIDERATIONS

7.1 Introduction

Power On Self Testing (POST) is performed by most systems after a reset. This chapter discusses the requirements for properly testing an 82385SX based system after power up.

7.2 Main Memory (DRAM) Testing

Most systems perform a memory test by writing a data pattern and then reading and comparing the data. This test may also be used to determine the total available memory within the system. Without properly taking into account the 82385SX cache memory, the memory test can give erroneous results. This will occur if the cache responds with read hits during the memory test routine.

7.2.1 MEMORY TESTING ROUTINE

In order to properly test main memory, the test routine must not read from the same block consecutively. For instance, if the test routine writes a data pattern to the first 16 Kbytes of memory (0000–3FFFH), reads from the same block, writes a new pattern to the same locations (0000–3FFFH), and read the new pattern, the second pattern tested would have had data returned from the 82385SX cache memory. Therefore, it is recommended that the test routine work with a memory block of at least 32 Kbytes. This will guarantee that no 16 Kbyte block will be read twice consecutively.

7.3 82385SX Cache Memory Testing

With the addition of SRAMs for the cache memory, it may be desirable for the system to be able to test the cache SRAMs during system diagnostics. This requires the test routine to access only the cache memory. The requirements for this routine are based on where it resides within the memory map. This can

5

Table 6-5. SRAM Specs for Buffered Cache Memory

SRAM Spec Requirements				
	Direct Mapped		2-Way Set Associative	
	16 MHz	20 MHz	16 MHz	20 MHz
Read Cycle Requirements				
Address Access (MAX)	57 ns	37 ns	55 ns	35 ns
Chip Select Access (MAX)	68	48	68	48
OE# to Data Valid (MAX)	N/A	N/A	N/A	N/A
OE# to Data Float (MAX)	N/A	N/A	N/A	N/A
Write Cycle Requirements				
Chip Select to End of Write (MIN)	40	30	40	30
Address Valid to End of Write (MIN)	58	42	56	40
Write Pulse Width (MIN)	40	30	40	30
Data Setup (MAX)	25	15	25	15
Data Hold (MIN)	3	3	3	3

be broken into two areas: the routine residing in cacheable memory space or the routine residing in either non-cacheable memory or on the 386 SX local bus (using the LBA# input).

7.3.1 TEST ROUTINE IN THE NCA# OR LBA# MEMORY MAP

In this configuration, the test routine will never be cached. The recommended method is code which will access a single 16 Kbyte block during the test. Initially, a 16 Kbyte read (assume 0000–3FFFH) must be executed. This will fill the cache directory with the address information which will be used in the diagnostic procedure. Then, a 16 Kbyte write to the same address locations (0000–3FFFH) will load the cache with the desired test pattern (due to write hits). The comparison can be made by completing another 16 Kbyte read (same locations, 0000–3FFFH), which will be cache read hits. Subsequent writes and reads to the same addresses will enable various patterns to be tested.

7.3.2 TEST ROUTINE IN CACHEABLE MEMORY

In this case, it must be understood that the diagnostic routine must reside in the cache memory before the actual data testing can begin. Otherwise, when the 386 SX performs a code fetch, a location within the cache memory which is to be tested will be altered due to the read miss (code fetch) update.

The first task is to load the diagnostic routine into the top of the cache memory. It must be known how much memory is required for the code as the rest of the cache memory will be tested as in the earlier method. Once the diagnostics have been cached (via read updates), the code will perform the same type of read/write/read/compare as in the routine explained in the above section. The difference is that now the amount of cache memory to be tested is 16 Kbytes minus the length of the test routine.

7.4 82385SX Cache Directory Testing

Since the 82385SX does not directly access the data bus, it is not possible to easily complete a comparison of the cache directory. (The 82385SX can serially transmit its directory contents. See Section 7.5.) However, the cache memory tests described in Section 7.3 will indicate if the directory is working properly. Otherwise, the data comparison within the diagnostics will show locations which fail.

There is a slight possibility that the cache memory comparison could pass even if locations within the directory gave false hit/miss results. This could cause the comparison to always be performed to main memory instead of the cache and give a proper comparison to the 386 SX. The solution here is to use the MISS# output of the 82385SX as an indicator to a diagnostic port which can be read by the 386 SX. It could also be used to flag an interrupt if a failure occurs.

The implementation of these techniques in the diagnostics will assure proper functionality of the 82385SX subsystem.

7.5 Special Function Pins

As mentioned in Chapter 3, there are three 82385SX pins which have reserved functions in addition to their normal operational functions. These pins are MISS#, WBS, and FLUSH.

As discussed previously, the 82385SX performs a directory flush when the FLUSH input is held active for at least 4 CLK (8 CLK2) cycles. However, the FLUSH pin also serves as a diagnostic input to the 82385SX. The 82385SX will enter a reserved mode if the FLUSH pin is high at the falling edge of RESET.

If, during normal operation, the FLUSH input is active for only one CLK (2 CLK2) cycle/s, the 82385SX will enter another reserved mode. Therefore it must be guaranteed that FLUSH is active for at least the 4 CLK (8 CLK2) cycle specification.

WBS and MISS# serve as outputs in the 82385SX reserved modes.

8.0 MECHANICAL DATA

8.1 Introduction

This chapter discusses the physical package and its connections in detail.

8.2 Pin Assignment

The 82385SX PGA pinout as viewed from the top side of the component is shown by Figure 8-1. Its pinout as viewed from the Pin side of the component is shown in Figure 8-2.

The 82385SX Plastic Quad Flat Pack (PQFP) pinout from the top side of the component is shown by Figure 8-3.

V_{CC} and V_{SS} connections must be made to multiple V_{CC} and V_{SS} (GND) pins. Each V_{CC} and V_{SS} must be connected to the appropriate voltage level. The circuit board should include V_{CC} and GND planes for power distribution and all V_{CC} and V_{SS} pins must be connected to the appropriate plane.

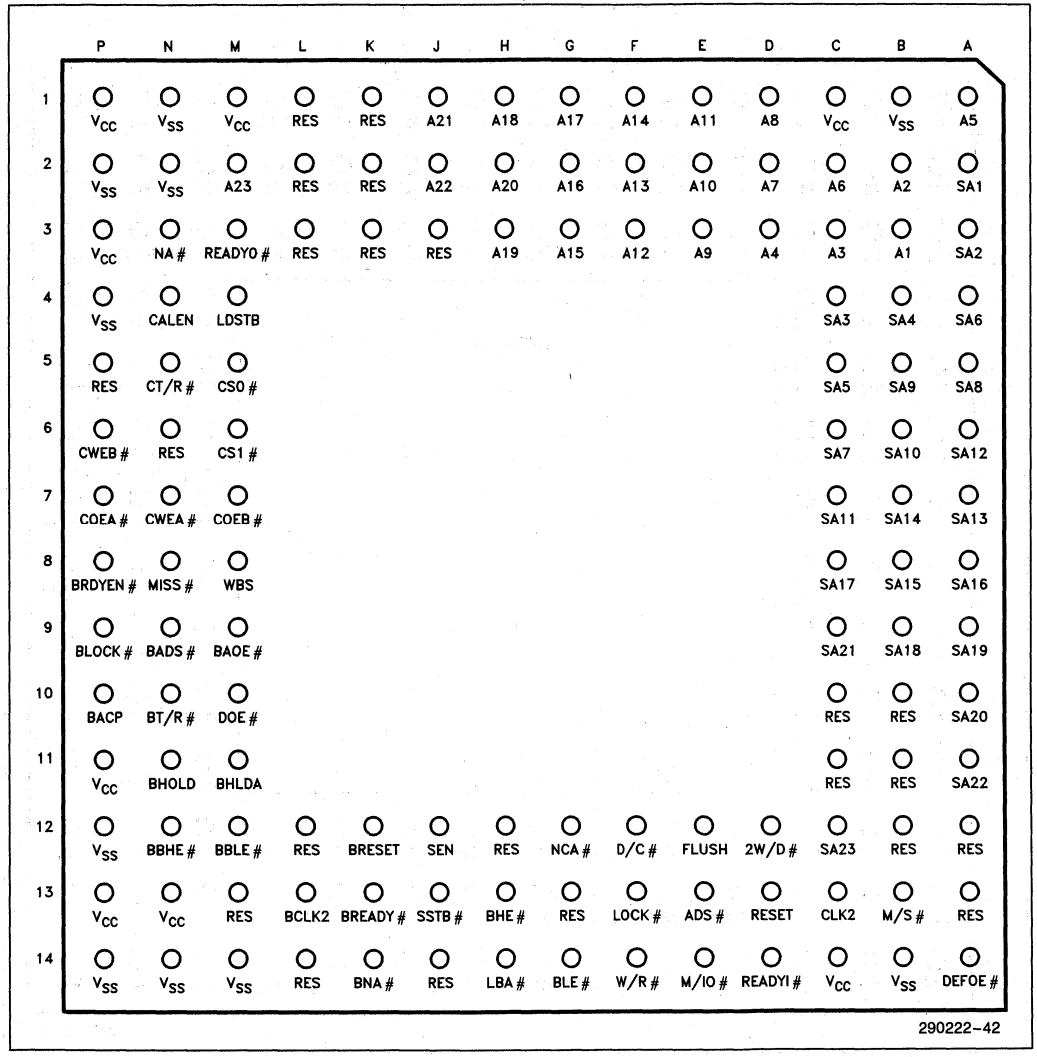


Figure 8-1. 82385SX PGA Pinout—View from TOP Side

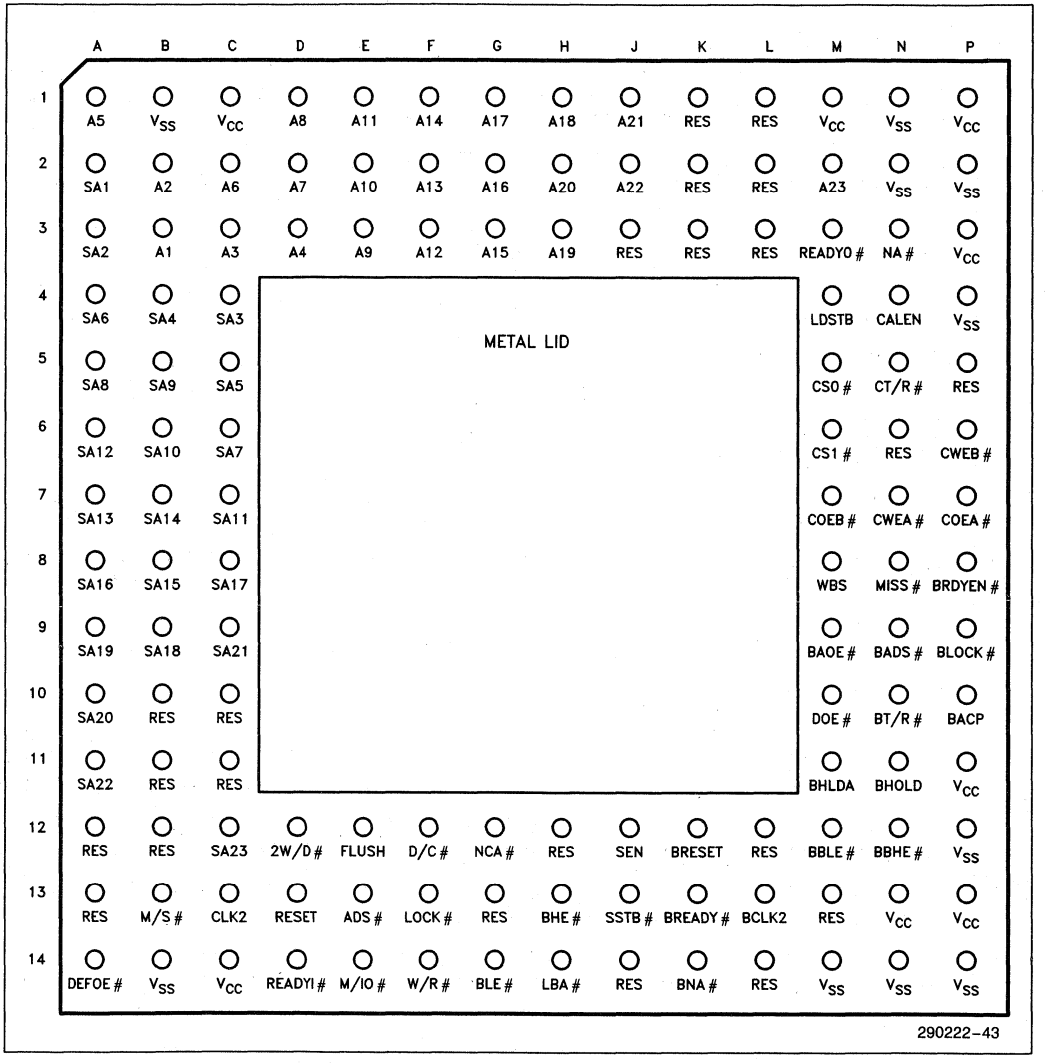


Figure 8-2. 82385SX PGA Pinout—View from PIN Side

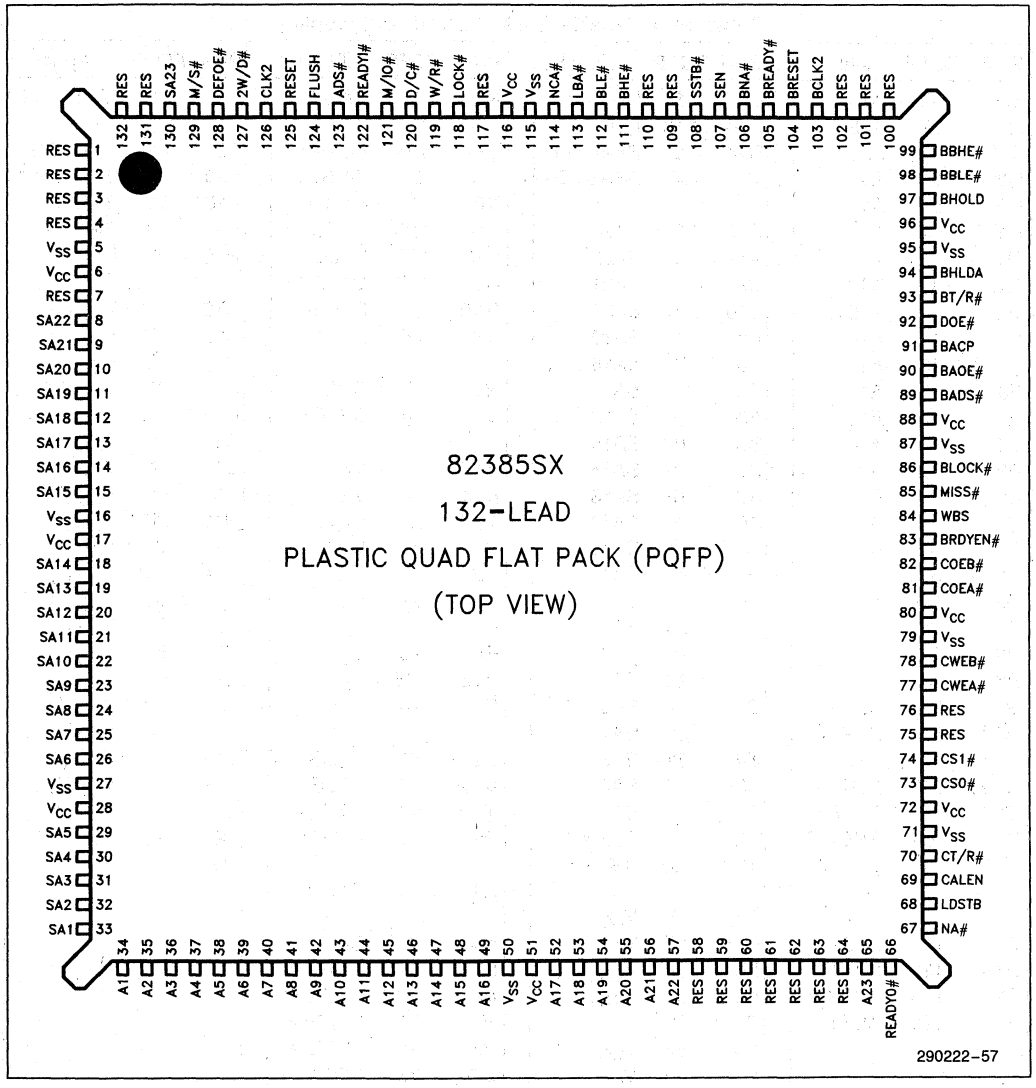


Figure 8-3. 82385SX PQFP Pinout—View from TOP Side

Table 8-1. 82385SX Pinout—Functional Grouping

PGA	PQFP	Signal
M2	65	A23
J2	57	A22
J1	56	A21
H2	55	A20
H3	54	A19
H1	53	A18
G1	52	A17
G2	49	A16
G3	48	A15
F1	47	A14
F2	46	A13
F3	45	A12
E1	44	A11
E2	43	A10
E3	42	A9
D1	41	A8
D2	40	A7
C2	39	A6
A1	38	A5
D3	37	A4
C3	36	A3
B2	35	A2
B3	34	A1
G14	112	BLE#
H13	111	BHE#
C13	126	CLK2
D13	125	RESET
K12	104	BRESET
L13	103	BCLK2
F14	119	W/R#
F12	120	D/C#
E14	121	M/IO#
F13	118	LOCK#
E13	123	ADS#

PGA	PQFP	Signal
G12	114	NCA#
H14	113	LBA#
D14	122	READYI#
M3	66	READYO#
C12	130	SA23
A11	8	SA22
C9	9	SA21
A10	10	SA20
A9	11	SA19
B9	12	SA18
C8	13	SA17
A8	14	SA16
B8	15	SA15
B7	18	SA14
A7	19	SA13
A6	20	SA12
C7	21	SA11
B6	22	SA10
B5	23	SA9
A5	24	SA8
C6	25	SA7
A4	26	SA6
C5	29	SA5
B4	30	SA4
C4	31	SA3
A3	32	SA2
A2	33	SA1
J12	107	SEN#
J13	108	SSTB#
C1	6	V_{CC}
C14	17	V_{CC}
M1	28	V_{CC}
N13	51	V_{CC}
P1	72	V_{CC}
P3	80	V_{CC}
P11	88	V_{CC}
P13	96	V_{CC}
—	116	V_{CC}

PGA	PQFP	Signal
N3	67	NA#
E12	124	FLUSH
M8	84	WBS
N8	85	MISS#
A14	128	DEFOE#
B13	129	M/S#
D12	127	2W/D#
M10	92	DOE#
M4	68	LDSTB
N11	97	BHOLD
M11	94	BHLDA
B1	5	V_{SS}
B14	16	V_{SS}
M14	27	V_{SS}
N1	50	V_{SS}
N2	71	V_{SS}
N14	79	V_{SS}
P2	87	V_{SS}
P4	95	V_{SS}
P12	115	V_{SS}
P14	—	V_{SS}
N9	89	BADS#
M12	98	BBLE#
N12	99	BBHE#
P9	86	BLOCK#
K14	106	BNA#
N4	69	CALEN
P7	81	COEA#
M7	82	COEB#
N7	77	CWEA#
P6	78	CWEB#
M5	73	CS0#
M6	74	CS1#

PGA	PQFP	Signal
N5	70	CT/R#
P8	83	BRDYEN#
K13	105	BREADY#
P10	91	BACP
M9	90	BAOE#
N10	93	BT/R#
A12	1	V_{CC} (*)
A13	131	V_{CC} (*)
B10	7	V_{CC} (*)
B11	3	V_{CC} (*)
B12	132	V_{CC} (*)
C10	4	V_{CC} (*)
C11	2	V_{CC} (*)
G13	117	V_{CC} (*)
H12	110	V_{CC} (*)
J3	60	V_{CC} (*)
J14	109	V_{CC} (*)
K1	58	V_{CC} (*)
K2	59	V_{CC} (*)
K3	62	V_{CC} (*)
L1	61	V_{CC} (*)
L2	63	V_{CC} (*)
L3	64	V_{CC} (*)
L12	100	N.C. (*)
L14	102	V_{CC} (*)
M13	101	N.C. (*)
N6	75	N.C. (*)
P5	76	N.C. (*)

*Reserved pins, N.C. indicates a no connect.

8.3 Package Dimensions and Mounting

The 82385SX PGA package is a 132-pin ceramic Pin Grid Array. The pins are arranged 0.100 inch (2.54 mm) center-to-center, in a 14 × 14 matrix, three rows around.

A wide variety of available PGA sockets allow low insertion force or zero insertion force mounting. These come in a choice of terminals such as solder-tail, surface mount, or wire wrap.

The 82385SX PQFP is a 132-lead Plastic Quad Flat Pack. The pins are "fine pitch", 0.025 inches (0.635 mm) center to center.

The PQFP device is intended to be surface mounted directly to the printed board although sockets are available for this device.

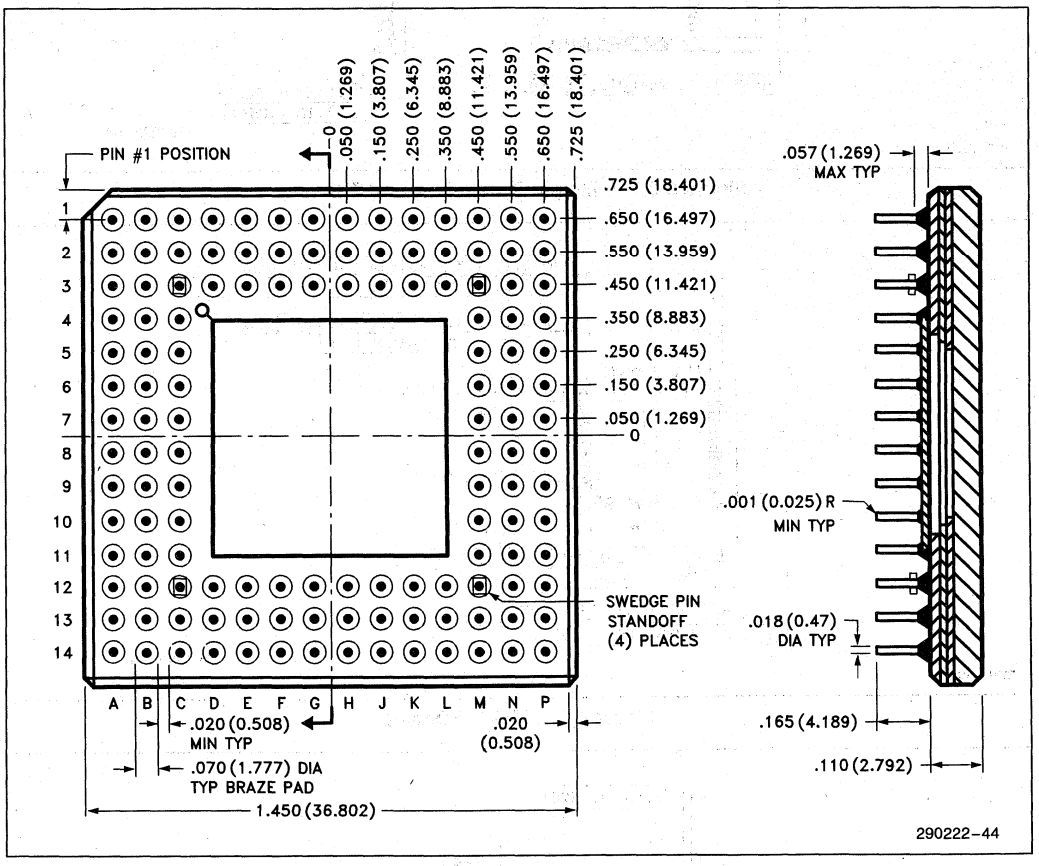


Figure 8-3.1. 132-Pin PGA Package Dimensions

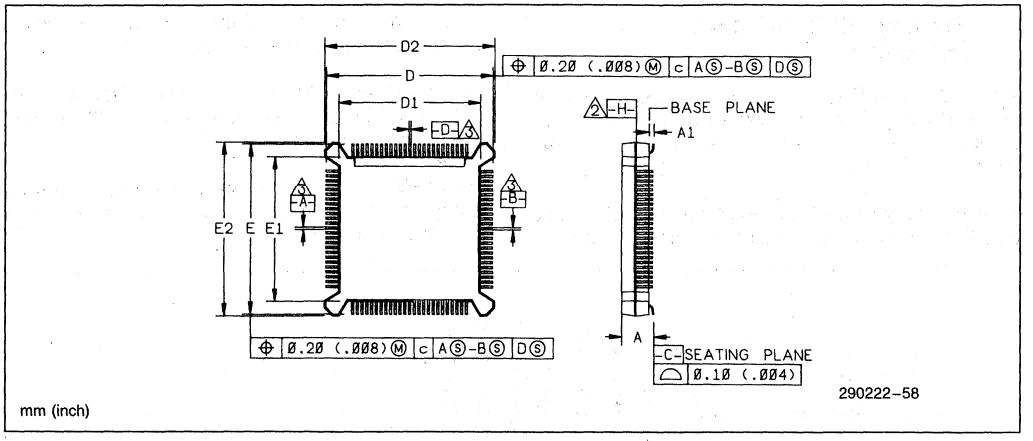


Figure 8-3.2. Principal Dimensions and Datums

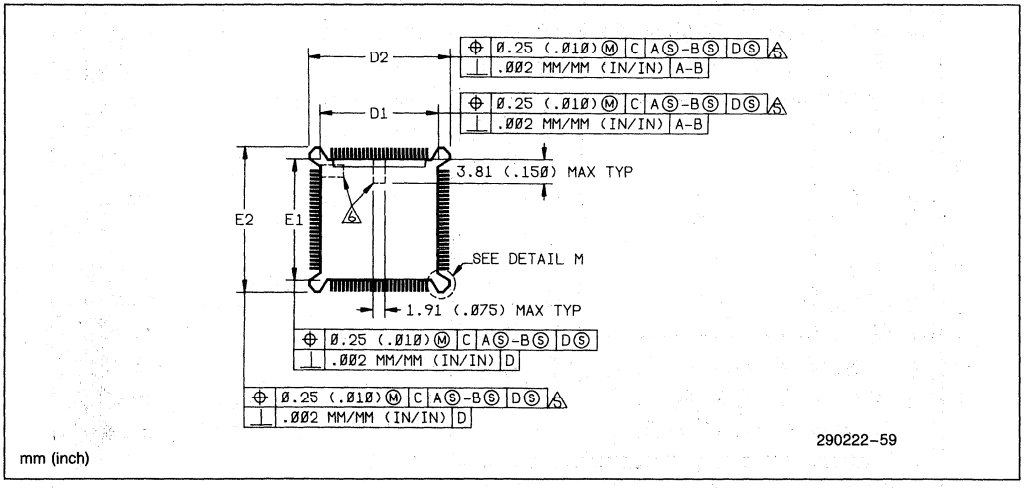


Figure 8-3.3. Molded Details

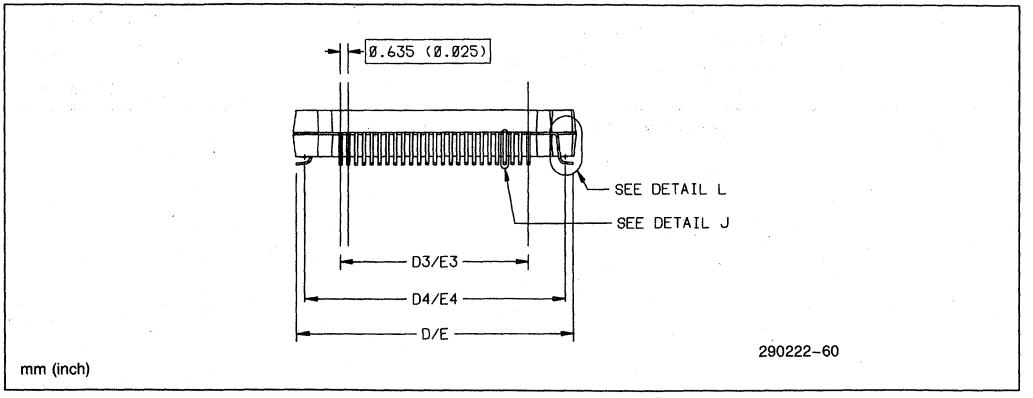


Figure 8-3.4. Terminal Details

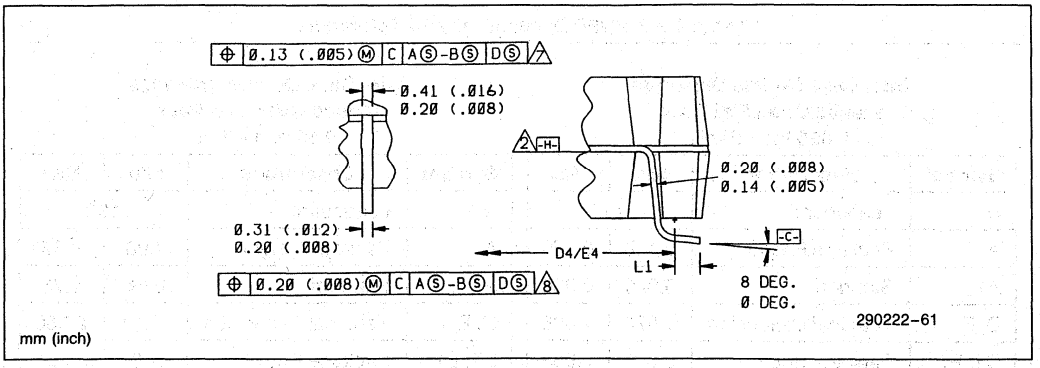


Figure 8-3.5. Typical Lead

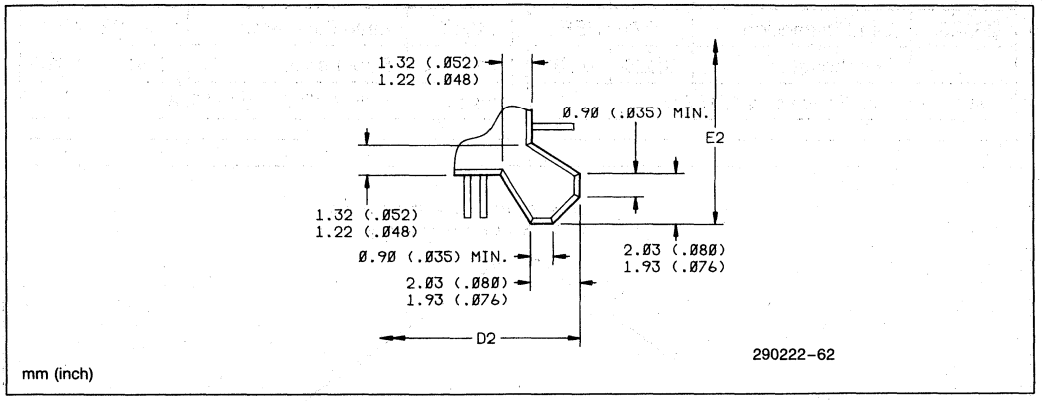


Figure 8-3.6. Detail M

PLASTIC QUAD FLAT PACK

Table 8-3.1. Symbol List for Plastic Quad Flat Pack

Letter or Symbol	Description of Dimensions
A	Package height: distance from seating plane to highest point of body
A1	Standoff: Distance from seating plane to base plane
D/E	Overall package dimension: lead tip to lead tip
D1/E1	Plastic body dimension
D2/E2	Bumper Distance
D3/E3	Footprint
L1	Foot length
N	Total number of leads

NOTES:
1. All dimensions and tolerances conform to ANSI Y14.5M-1982.
2. Datum plane -H- located at the mold parting line and coincident with the bottom of the lead where lead exits plastic body.
3. Datums A-B and -D- to be determined where center leads exit plastic body at datum plane -H-.
4. Controlling Dimension, Inch.
5. Dimensions D1, D2, E1 and E2 are measured at the mold parting line and do not include mold protrusion. Allowable mold protrusion of 0.18 mm (0.007 in) per side.
6. Pin 1 identifier is located within one of the two zones indicated.
7. Measured at datum plane -H-.
8. Measured at seating plane datum -C-.

Table 8-3.2. PQFP Dimensions and Tolerances

Intel Case Outline Drawings Plastic Quad Flat Pack 0.025 Inch Pitch			
Symbol	**Description**	**Min**	**Max**
N	Leadcount	132	
A	Package Height	0.160	0.170
A1	Standoff	0.020	0.030
D, E	Terminal Dimension	1.075	1.085
D1, E1	Package Body	0.947	0.953
D2, E2	Bumper Distance	1.097	1.103
D3, E3	Lead Dimension	0.800 REF	
L1	Foot Length	0.020	0.030
Issue	IWS Preliminary 1/15/87		

Intel Case Outline Drawings Plastic Quad Flat Pack 0.64 mm Pitch			
Symbol	**Description**	**Min**	**Max**
N	Leadcount	132	
A	Package Height	4.06	4.32
A1	Standoff	0.51	0.76
D, E	Terminal Dimension	27.31	27.56
D1, E1	Package Body	24.05	24.21
D2, E2	Bumper Distance	27.86	28.02
D3, E3	Lead Dimension	20.32 REF	
L1	Foot Length	0.51	0.76
Issue	IWS Preliminary 1/15/87		

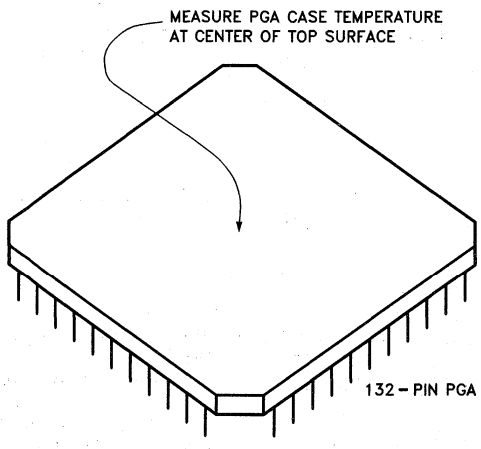
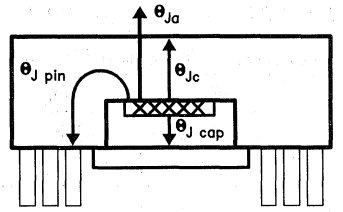


Figure 8-3.7. Measuring 82385SX PGA Case Temperature

Table 8-3.3. 82385SX PGA Package Typical Thermal Characteristics

Parameter	Thermal Resistance—°C/Watt						
	Airflow—f³/min (m³/sec)						
	0 (0)	50 (0.25)	100 (0.50)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)
θ Junction-to-Case (Case Measured as Figure 8-3.7)	2	2	2	2	2	2	2
θ Case-to-Ambient (No Heatsink)	19	18	17	15	12	10	9
θ Case-to-Ambient (with Omnidirectional Heatsink)	16	15	14	12	9	7	6
θ Case-to-Ambient (with Unidirectional Heatsink)	15	14	13	11	8	6	5

NOTES:
1. Table 8-3.4 applies to 82385SX PGA plugged into socket or soldered directly onto board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.
3. $\theta_{J\text{-}CAP} = 4°C/W$ (approx.)
   $\theta_{J\text{-}PIN} = 4°C/W$ (inner pins) (approx.)
   $\theta_{J\text{-}PIN} = 8°C/W$ (outer pins) (approx.)



290222-46

Table 8-3.3. 82385 PQFP Package Typical Thermal Characteristics

Parameter	Thermal Resistance—°C/Watt						
	Airflow—/LFM						
	0 (0)	50 (0.25)	100 (0.50)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)
θ Junction-to-Case (Case Measured as Figure 8-3.7)	5	5	5	5	5	5	5
θ Case-to-Ambient (No Heatsink)	23.5	22.0	20.5	17.5	14.0	11.5	9.5
θ Case-to-Ambient (with Omnidirectional Heatsink)	TO BE DEFINED						
θ Case-to-Ambient (with Unidirectional Heatsink)							

NOTES:
1. Table 8-3.3 applies to 82385SX PQFP plugged into socket or soldered directly onto board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.
3. $\theta_{J\text{-}CAP} = 4°C/W$ (approx.)
   $\theta_{J\text{-}PIN} = 4°C/W$ (inner pins) (approx.)
   $\theta_{J\text{-}PIN} = 8°C/W$ (outer pins) (approx.)

8.4 Package Thermal Specification

The case temperature should be measured at the center of the top surface as in Figure 8-3.7 for PGA or Table 8-3.3 for PQFP. The case temperature may be measured in any environment to determine whether or not the 82385SX is within the specified operating range.

9.0 ELECTRICAL DATA

9.1 Introduction

This chapter presents the A.C. and D.C specifications for the 82385SX.

9.2 Maximum Ratings

Storage Temperature ..... −65°C to +150°C

Case Temperature under Bias ... −65°C to +110°C

Supply Voltage
with Respect to V_{SS} ..... −0.5V to +6.5V

Voltage on Any Other Pin .... −0.5V to V_{CC} + 0.5V

NOTE:
Stress above those listed may cause permanent damage to the device. This is a stress rating only and functional operation at these or any other conditions above those listed in the operational sections of this specification is not implied.

Exposure to absolute maximum rating conditions for extended periods may affect device reliability. Although the 82385SX contains protective circuitry to resist damage from static electric discharges, always take precautions against high static voltages or electric fields.

9.3 D.C. Specifications T_{CASE} = 0°C to +85°C; V_{CC} = 5V ±5%; V_{SS} = 0V

Table 9-1. D.C. Specifications (16 MHz and 20 MHz)

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{IL}	Input Low Voltage	−0.3	0.8	V	(Noe 1)
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{CL}	CLK2, BCLK2 Input Low	−0.3	0.8	V	(Note 1)
V_{CH}	CLK2, BCLK2 Input High	V_{CC} − 0.8	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage		0.45	V	I_{OL} = 4 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = −1 mA
I_{CC}	Power Supply Current		275	mA	(Note 2)
I_{LI}	Input Leakage Current		±15	μA	$0V < V_{IN} < V_{CC}$
I_{LO}	Output Leakage Current		±15	μA	$0.45V < V_{OUT} < V_{CC}$
C_{IN}	Input Capacitance		10	pF	(Note 3)
C_{CLK}	CLK2 Input Capacitance		20	pF	(Note 3)

NOTES:
1. Minimum value is not 100% tested.
2. I_{CC} is specified with inputs driven to CMOS levels. I_{CC} may be higher if driven to TTL levels.
3. Sampled only.

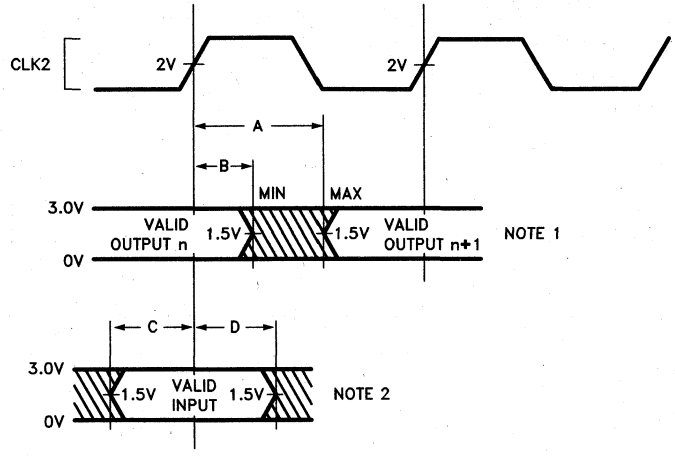
9.4 A.C. Specifications

The A.C. specifications given in the following tables consist of output delays and input setup requirements. The A.C. diagram's purpose is to illustrate the clock edges from which the timing parameters are measured. The reader should not infer any other timing relationships from them. For specific information on timing relationships between signals, refer to the appropriate functional section.

A.C. spec measurement is defined in Figure 9-1. Inputs must be driven to the levels shown when A.C. specifications are measured. 82385SX output delays are specified with minimum and maximum limits, which are measured as shown. 82385SX input setup and hold times are specified as minimums and define the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct 82385SX operation.

9.4.1 FREQUENCY DEPENDENT SIGNALS

The 82385SX has signals whose output valid delays are dependent on the clock frequency. These signals are marked in the A.C. Specification Tables with a Note 1.



LEGEND:
A—Maximum output delay specification
B—Minimum output delay specification
C—Minimum input setup specification
D—Minimum input hold specification

NOTES:
1. Under rated loading 82385SX output (t_r and t_f) is typically ≤ 4.0 ns from 0.8V to 2.0V.
2. Input waveforms have t_r ≤ 2.0 ns from 0.8V to 2.0V.

Figure 9-1. Drive Levels and Measurement Points for A.C. Specification

5

A.C. SPECIFICATION TABLES

Functional operating range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C

Table 4.1. A.C. Specifications at 16 MHz

Symbol	Parameter	Min	Max	Units	Notes
t1	Operating Frequency	15.4	16	MHz	
t2	CLK2, BCLK2 Period	31.25	32.5	ns	
t3a	CLK2, BCLK2 High Time @ 2V	10		ns	
t3b	CLK2, BCLK2 High Time @ 3.7V	7		ns	3
t4a	CLK2, BCLK2 Low Time @ 2V	10		ns	
t4b	CLK2, BCLK2 Low Time @ 0.8V	7		ns	3
t5	CLK2, BCLK2 Fall Time		8	ns	3, 9
t6	CLK2, BCLK2 Rise Time		8	ns	3, 9
t7a	A4–A12 Setup Time	30		ns	1
t7b	LOCK# Setup Time	19		ns	1
t7c	BLE#, BHE# Setup Time	21		ns	1
t7d	A1–A3, A13–A23 Setup Time	23		ns	1
t8	A1–A23, BLE#, BHE#, LOCK# Hold	3		ns	
t9a	M/IO#, D/C# Setup Time	30		ns	1
t9b	W/R# Setup Time	30		ns	1
t9c	ADS# Setup Time	30		ns	1
t10	M/IO#, D/C#, W/R#, ADS# Hold Time	5		ns	
t11	READYI# Setup Time	19		ns	1
t12	READYI# Hold Time	4		ns	
t13a1	NCA# Setup Time (See t55b2)	27		ns	6
t13a2	NCA# Setup Time (See t55b3)	20		ns	6
t13b	LBA# Setup Time	16		ns	
t14a	NCA# Hold Time	4		ns	
t14b	LBA# Hold Time	4		ns	
t15	RESET, BRESET Setup Time	13		ns	
t16	RESET, BRESET Hold Time	4		ns	
t17	NA# Valid Delay	12	42	ns	1 (25 pF Load)
t18	READYO# Valid Delay	3	31	ns	1 (25 pF Load)
t19	BRDYEN# Valid Delay	3	31	ns	
t21a1	CALEN Rising, PHI1	3	30	ns	
t21a2	CALEN Falling, PHI1	3	30	ns	
t21a3	CALEN Falling in T1P, PHI2	3	30	ns	
t21b	CALEN Rising Following CWTH	3	39	ns	1
t21c	CALEN Pulse Width	10		ns	
t21d	CALEN Rising to CS# Falling	13		ns	

A.C. SPECIFICATION TABLES (Continued)

Functional operating range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C

Table 4.1. A.C. Specifications at 16 MHz (Continued)

Symbol	Parameter	Min	Max	Units	Notes
t22a1	CWEx# Falling, PHI1 (CWTH)	4	31	ns	1
t22a2	CWEx# Falling, PHI2 (CRDM)	4	31	ns	1
t22b	CWEx# Pulse Width	40		ns	1, 2
t22c1	CWEx# Rising, PHI1 (CWTH)	4	31	ns	1
t22c2	CWEx# Rising, PHI2 (CRDM)	4	31	ns	1
t23a1	CS1#, CS2# Rising, PHI1 (CRDM)	6	41	ns	1
t23a2	CS1#, CS2# Rising, PHI2 (CWTH)	6	41	ns	1
t23a3	CS1#, CS2# Falling, PHI1 (CWTH)	6	41	ns	1
t23a4	CS1#, CS2# Falling, PHI2 (CRDM)	6	41	ns	1
t24a1	CT/R# Rising, PHI2 (CRDH)	6	43	ns	1
t24a2	CT/R# Falling, PHI1 (CRDH)	6	43	ns	1
t24a3	CT/R# Falling, PHI2 (CRDH)	6	43	ns	1
t25a	COEA#, COEB# Falling (Direct)	4	33	ns	(25 pF Load)
t25b	COEA#, COEB# Falling (2-Way)	4	34	ns	1 (25 pF Load)
t25c1	COEx# Rising Delay @ T_{CASE} = 0C	4	20	ns	(25 pF Load)
t25c2	COEx# Rising Delay @ T_{CASE} = T_{MAX}	4	20	ns	(25 pF Load)
t23b	COEx# Falling to CSx# Rising	0		ns	
t25d	CWEx# Falling to COEx# Falling or CWEx# Rising to COEx# Rising	0	10	ns	(25 pF Load)
t26	CS0#, CS1# Falling to CWEx# Rising	40		ns	1, 2
t27	CWEx# Falling to CS0#, CS1# Falling	0		ns	
t28a	CWEx# Rising to CALEN Rising	0		ns	
t28b	CWEx# Rising to CS0#, CS1# Falling	0		ns	
t31	SA(1-23) Setup Time	25		ns	
t32	SA(1-23) Hold Time	3		ns	
t33	BADS# Valid Delay	4	33	ns	1
t34	BADS# Float Delay	4	33	ns	3
t35	BNA# Setup Time	11		ns	
t36	BNA# Hold Time	15		ns	
t37	BREADY# Setup Time	31		ns	1
t38	BREADY# Hold Time	4		ns	
t40a	BACP Rising Delay	0	26	ns	
t40b	BACP Falling Delay	0	28	ns	
t41	BAOE# Valid Delay	3	23	ns	

A.C. SPECIFICATION TABLES (Continued)

Functional operating range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C

Table 4.1. A.C. Specifications at 16 MHz (Continued)

Symbol	Parameter	Min	Max	Units	Notes
t43a	BT/R# Valid Delay	2	27	ns	
t43b1	DOE# Falling Delay	2	30	ns	
t43b2	DOE# Rising Delay @ T_{CASE} = 0C	3	23	ns	
t43b3	DOE# Rising Delay @ T_{CASE} = T_{MAX}	3	26	ns	
t43c	LDSTB Valid Delay	2	33	ns	
t44a	SEN Setup Time	15		ns	
t44b	SSTB# Setup Time	15		ns	
t45	SEN, SSTB# Hold Time	5		ns	
t46	BHOLD Setup Time	26		ns	
t47	BHOLD Hold Time	5		ns	
t48	BHLDA Valid Delay	3	33	ns	
t55a	BLOCK# Valid Delay	3	36	ns	1, 5
t55b1	BBxE# Valid Delay	3	36	ns	1, 7
t55b2	BBxE# Valid Delay	3	36	ns	1, 7
t55b3	BBxE# Valid Delay	3	43	ns	1, 7
t55c	LOCK# Falling to BLOCK# Falling	0	36	ns	1, 5
t56	MISS# Valid Delay	3	43	ns	1
t57	MISS#, BBxE#, BLOCK# Float Delay	4	40	ns	3
t58	WBS Valid Delay	3	39	ns	1
t59	FLUSH Setup Time	21		ns	
t60	FLUSH Hold Time	5		ns	
t61	FLUSH Setup to RESET Low	31		ns	
t62	FLUSH Hold from RESET Low	31		ns	

A.C. SPECIFICATION TABLES

Functional operating range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C

A.C. Specifications at 20 MHz

Symbol	Parameter	Min	Max	Units	Notes
t1	Operating Frequency	15.4	20	MHz	
t2	CLK2, BCLK2 Period	25	32.5	ns	
t3a	CLK2, BCLK2 High Time @ 2V	10		ns	
t3b	CLK2, BCLK2 High Time @ 3.7V	7		ns	3
t4a	CLK2, BCLK2 Low Time @ 2V	10		ns	
t4b	CLK2, BCLK2 Low Time @ 0.8V	7		ns	3
t5	CLK2, BCLK2 Fall Time		8	ns	3, 9
t6	CLK2, BCLK2 Rise Time		8	ns	3, 9
t7a1	A4–A12 Setup Time	20		ns	1
t7a2	A1–A3, A13–A19, A21–A23 Setup Time	18		ns	1
t7a3	A20 Setup Time	16		ns	1
t7b	LOCK# Setup Time	16		ns	1
t7c	BLE#, BHE# Setup Time	18		ns	1
t8	A1–A23, BLE#, BHE#, LOCK# Hold	3		ns	
t9a	M/IO#, D/C# Setup Time	20		ns	1
t9b	W/R# Setup Time	20		ns	1
t9c	ADS# Setup Time	22		ns	1
t10	M/IO#, D/C#, W/R#, ADS# Hold Time	5		ns	
t11	READYI# Setup Time	12		ns	1
t12	READYI# Hold Time	4		ns	
t13a1	NCA# Setup Time (See t55b2)	21		ns	6
t13a2	NCA# Setup Time (See t55b3)	16		ns	6
t13b	LBA# Setup Time	10		ns	
t14a	NCA# Hold Time	4		ns	
t14b	LBA# Hold Time	4		ns	
t15	RESET, BRESET Setup Time	12		ns	
t16	RESET, BRESET Hold Time	4		ns	
t17	NA# Valid Delay	12	34	ns	1 (25 pF Load)
t18	READYO# Valid Delay	3	26	ns	1 (25 pF Load)
t19	BRDYEN# Valid Delay	3	26	ns	
t21a1	CALEN Rising, PHI1	3	24	ns	
t21a2	CALEN Falling, PHI1	3	24	ns	
t21a3	CALEN Falling in T1P, PHI2	3	24	ns	
t21b	CALEN Rising Following CWTH	3	34	ns	1
t21c	CALEN Pulse Width	10		ns	

A.C. SPECIFICATION TABLES (Continued)

Functional operating range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C

A.C. Specifications at 20 MHz (Continued)

Symbol	Parameter	Min	Max	Units	Notes
t21d	CALEN Rising to CS# Falling	13		ns	
t22a1	CWEx# Falling, PHI1 (CWTH)	4	27	ns	1
t22a2	CWEx# Falling, PHI2 (CRDM)	4	27	ns	1
t22b	CWEx# Pulse Width	30		ns	1, 2
t22c1	CWEx# Rising, PHI1 (CWTH)	4	27	ns	1
t22c2	CWEx# Rising, PHI2 (CRDM)	4	27	ns	1
t23a1	CS1#, CS2# Rising, PHI1 (CRDM)	6	37	ns	1
t23a2	CS1#, CS2# Rising, PHI2 (CWTH)	6	37	ns	1
t23a3	CS1#, CS2# Falling, PHI1 (CWTH)	6	37	ns	1
t23a4	CS1#, CS2# Falling, PHI2 (CRDM)	6	37	ns	1
t24a1	CT/R# Rising, PHI2 (CRDH)	6	38	ns	1
t24a2	CT/R# Falling, PHI1 (CRDH)	6	38	ns	1
t24a3	CT/R# Falling, PHI2 (CRDH)	6	38	ns	1
t25a	COEA#, COEB# Falling (Direct)	4	22	ns	(25 pF Load)
t25b	COEA#, COEB# Falling (2-Way)	4	24.5	ns	1 (25 pF Load)
t25c	COEx# Rising Delay	5	17	ns	(25 pF Load)
CACHE SRAM WRITE CYCLES					
t23b	COEx# Falling to CSx# Rising	0		ns	8
t25d	CWEx# Falling to COEx# Falling or CWEx# Rising to COEx# Rising	0	10	ns	8 (25 pF Load)
t26	CS0#, CS1# Falling to CWEx# Rising	30		ns	1, 2
t27	CWEx# Falling to CS0#, CS1# Falling	0		ns	
t28a	CWEx# Rising to CALEN Rising	0		ns	
t28b	CWEx# Rising to CS0#, CS1# Falling	0		ns	
t31	SA(1-23) Setup Time	19		ns	
t32	SA(1-23) Hold Time	3		ns	
t33	BADS# Valid Delay	4	28	ns	1
t34	BADS# Float Delay	4	30	ns	3
t35	BNA# Setup Time	9		ns	
t36	BNA# Hold time	15		ns	
t37	BREADY# Setup Time	26		ns	1
t38	BREADY# Hold Time	4		ns	
t40a	BACP Rising Delay	0	20	ns	
t40b	BACP Falling Delay	0	22	ns	

A.C. SPECIFICATION TABLES (Continued)

Functional operating range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C

A.C. Specifications at 20 MHz (Continued)

Symbol	Parameter	Min	Max	Units	Notes
t41	BAOE# Valid Delay	3	18	ns	
t43a	BT/R# Valid Delay	2	19	ns	
t43b1	DOE# Falling Delay	2	23	ns	
t43b2	DOE# Rising Delay @ T_{CASE} = 0C	4	17	ns	
t43b3	DOE# Rising Delay @ T_{CASE} = T_{MAX}	4	19	ns	
t43c	LDSTB Valid Delay	2	26	ns	
t44a	SEN Setup Time	11		ns	
t44b	SSTB# Setup Time	11		ns	
t45	SEN, SSTB# Hold Time	5		ns	
t46	BHOLD Setup Time	17		ns	
t47	BHOLD Hold Time	5		ns	
t48	BHLDA Valid Delay	3	28	ns	
t55a	BLOCK# Valid Delay	3	30	ns	1, 5
t55b1	BBxE# Valid Delay	3	30	ns	1, 7
t55b2	BBxE# Valid Delay	3	30	ns	1, 7
t55b3	BBxE# Valid Delay	3	36	ns	1, 7
t55c	LOCK# Falling to BLOCK# Falling	0	30	ns	1, 5
t56	MISS# Valid Delay	3	35	ns	1
t57	MISS#, BBxE#, BLOCK# Float Delay	4	32	ns	3
t58	WBS Valid Delay	3	37	ns	1
t59	FLUSH Setup Time	16		ns	
t60	FLUSH Hold Time	5		ns	
t61	FLUSH Setup to RESET Low	26		ns	
t62	FLUSH Hold from RESET Low	26		ns	

82385SX A.C. Specification Notes:

1. Frequency dependent specifications.
2. Used for cache data memory (SRAM) specifications.
3. This parameter is sampled, not 100% tested. Guaranteed by design.
5. BLOCK# delay is either from BPHI1 or from 386 LOCK#. Refer to Figures 5-3K and 5-3L in the 82385SX data sheet.
6. NCA# setup time is now specified to the rising edge of BPHI2 in the state after 386 SX addresses become valid (either the state after the first T2 or after the first T2P).
7. BBxE# Valid Delay is a function of NCA# setup.
   BBxE# valid delay:
   - t55b1 For cacheable system bus accesses
   - t55b2 For NCA# setup < t13a1
   - t55b3 For t13a2 < NCA# setup < t13a1

8. t23b and t25d are only valid specifications when DEFOE# = V_{CC}. Otherwise, if DEFOE# = V_{SS}, COEx# is never asserted during cache SRAM write cycles. If DEFOE# = V_{SS}, t23b and t25d are Not Applicable.
9. t5 is measured from 0.8V to 3.7V. t6 is measured from 3.7V to 0.8V.

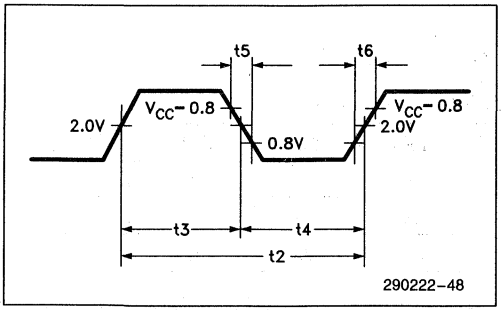


Figure 9-2. CLK2, BCLK2 Timing

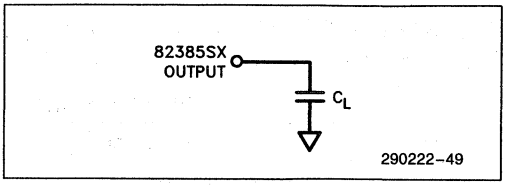
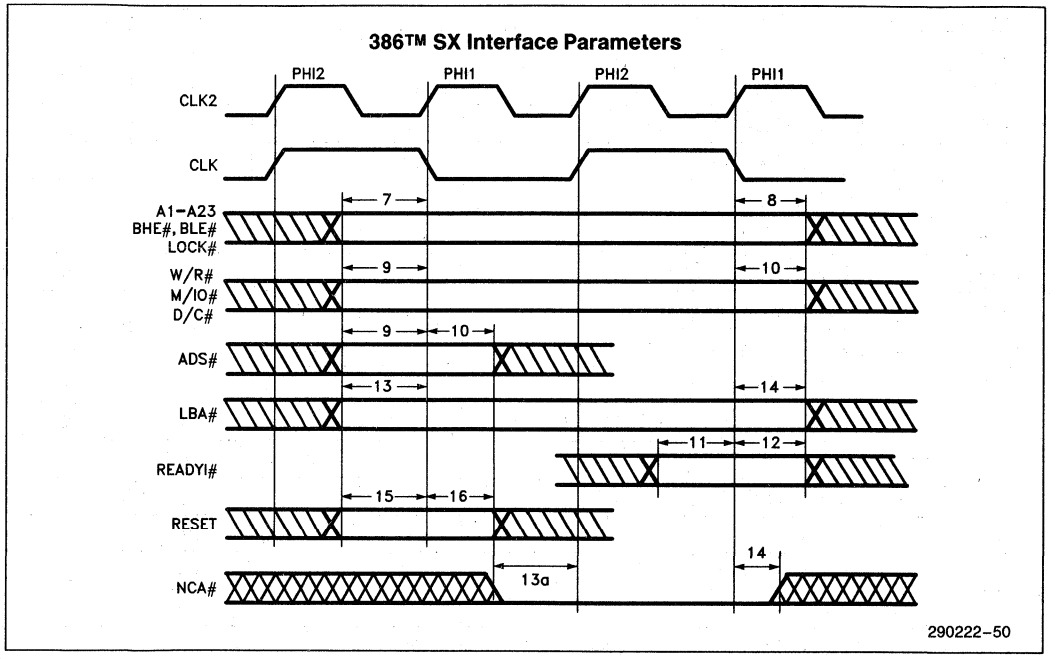
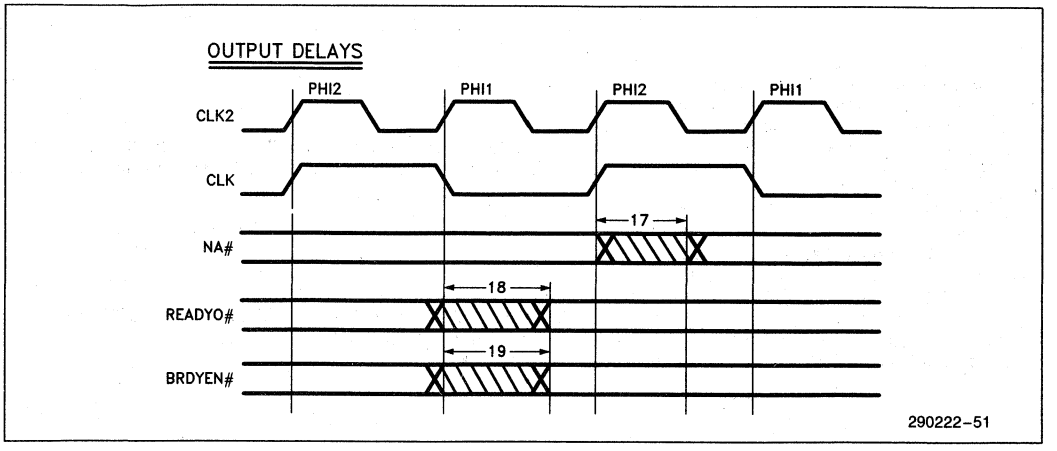
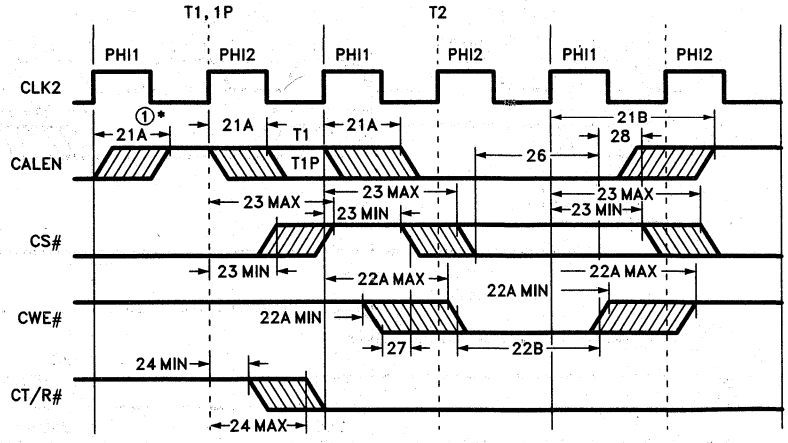


Figure 9-3. A.C. Test Load





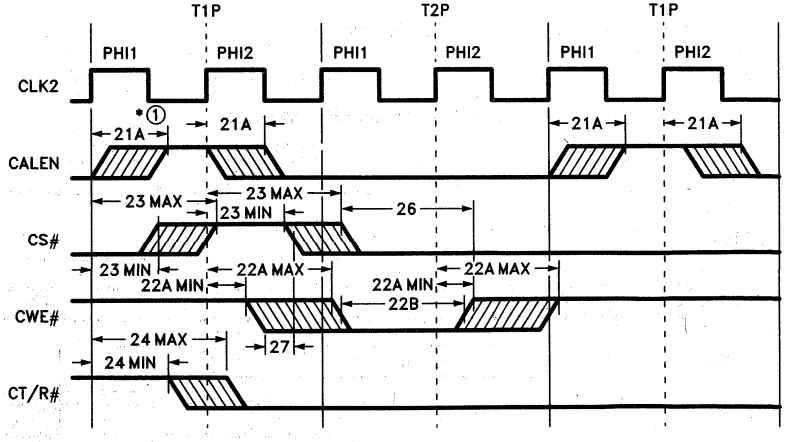
Cache Write Hit Cycle



290222-52

①*. This would be 21B if previous bus cycle was Cache Write Hit cycle.

Cache Read Miss (Cache Update Cycle)

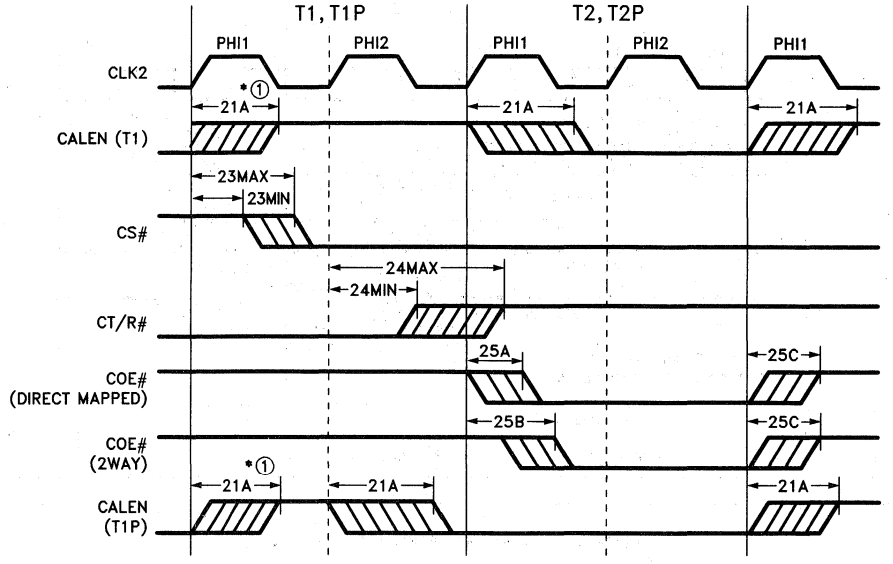


290222-53

①*. This would be 21B if previous bus cycle was Cache Write Hit cycle.

5

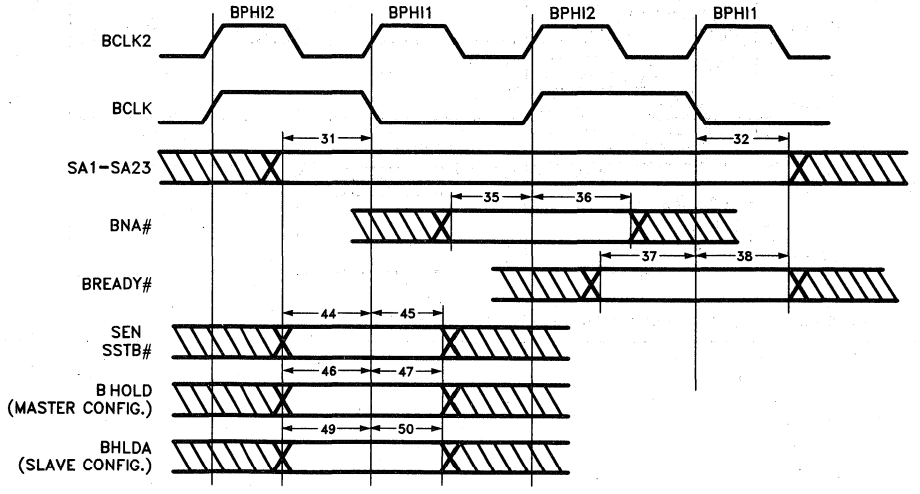
Cache Read Cycle



290222-54

①*. This would be 21B if previous bus cycle was Cache Write Hit cycle.

System Bus Interface Parameters

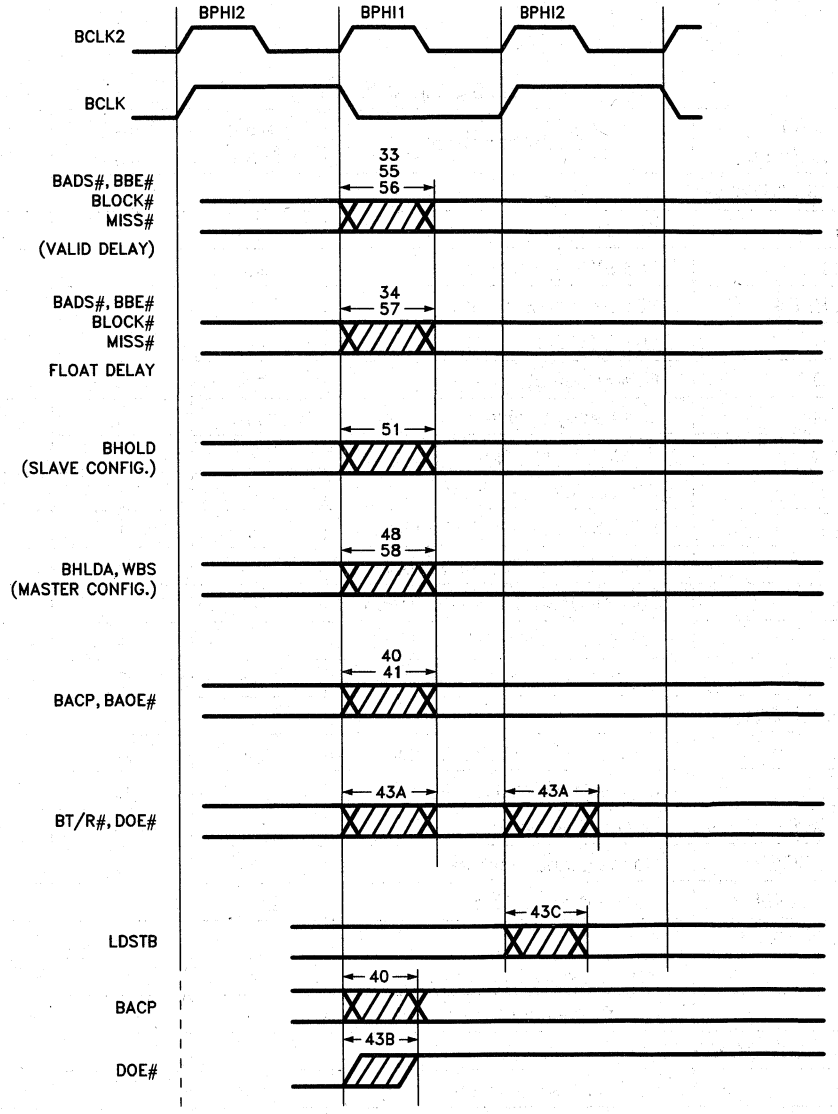


290222-55

①*. This would be 21B if previous bus cycle was Cache Write Hit cycle.

System Bus Interface Parameters (Continued)

OUTPUT DELAYS



290222-56

5

APPENDIX A

82385SX Signal Summary

Signal Group/Name	Signal Function	Active State	Input/ Output	Tri-State Output?
386 SX INTERFACE				
RESET	386 SX Reset	High	I	—
A1–A23	386 SX Address Bus	High	I	—
BHE#, BLE#	386 SX Byte Enables	Low	I	—
CLK2	386 SX Clock	—	I	—
READYO#	Ready Output	Low	O	No
BRDYEN#	Bus Ready Enable	Low	O	No
READYI#	386 SX Ready Input	Low	I	—
ADS#	386 SX Address Status	Low	I	—
M/IO#	386 SX Memory / I/O Indication	—	I	—
W/R#	386 SX Write/Read Indication	—	I	—
D/C#	386 SX Data/Control Indication	—	I	—
LOCK#	386 SX Lock Indication	Low	I	—
NA#	386 SX Next Address Request	Low	O	No
CACHE CONTROL				
CALEN	Cache Address Latch Enable	High	O	No
CT/R#	Cache Transmit/Receive	—	O	No
CS0#, CS1#	Cache Chip Selects	Low	O	No
COEA#, COEB#	Cache Output Enables	Low	O	No
CWEA#, CWEB#	Cache Write Enables	Low	O	No
LOCAL DECODE				
LBA#	386 SX Local Bus Access	Low	I	—
NCA#	Non-Cacheable Access	Low	I	—
STATUS AND CONTROL				
MISS#	Cache Miss Indication	Low	O	Yes
WBS	Write Buffer Status	High	O	No
FLUSH	Cache Flush	High	I	—

82385SX Signal Summary (Continued)

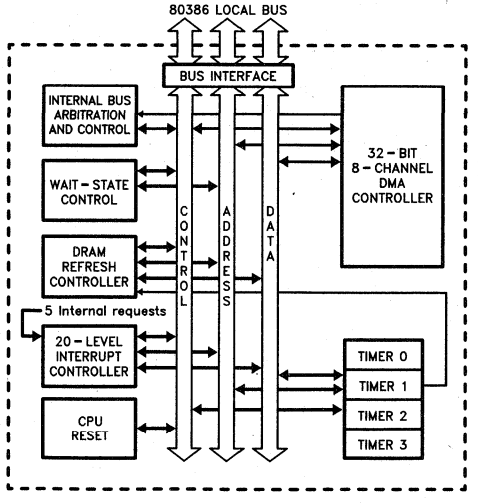
Signal Group/Name	Signal Function	Active State	Input/ Output	Tri-State Output?
82385SX INTERFACE				
BREADY#	82385SX Ready Input	Low	I	—
BNA#	82385SX Next Address Request	Low	I	—
BLOCK#	82385SX Lock Indication	Low	O	Yes
BADS#	82385SX Address Status	Low	O	Yes
BBHE#, BBLE#	82385SX Byte Enables	Low	O	yes
DATA/ADDR CONTROL				
LDSTB	Local Data Strobe	Pos. Edge	O	No
DOE#	Data Output Enable	Low	O	No
BT/R#	Bus Transmit/Receive	—	O	No
BACP	Bus Address Clock Pulse	Pos. Edge	O	No
BAOE#	Bus Address Output Enable	Low	O	No
CONFIGURATION				
2W/D#	2-Way/Direct Map Select	—	I	—
M/S#	Master/Slave Select	—	I	—
DEFOE#	Define Cache Output Enable	—	I	—
COHERENCY				
SA1–SA23	Snoop Address Bus	High	I	—
SSTB#	Snoop Strobe	Low	I	—
SEN	Snoop Enable	High	I	—
ARBITRATION				
BHOLD	Hold	High	I/O	No
BHLDA	Hold Acknowledge	High	I/O	No

82380
HIGH PERFORMANCE 32-BIT DMA CONTROLLER WITH INTEGRATED SYSTEM SUPPORT PERIPHERALS

- **High Performance 32-Bit DMA Controller**
  - **— 50 MBytes/sec Maximum Data Transfer Rate at 25 MHz**
  - **— 8 Independently Programmable Channels**
- **20-Source Interrupt Controller**
  - **— Individually Programmable Interrupt Vectors**
  - **— 15 External, 5 Internal Interrupts**
  - **— 82C59A Superset**
- **Four 16-Bit Programmable Interval Timers**
  - **— 82C54 Compatible**
- **Programmable Wait State Generator**
  - **— 0 to 15 Wait States Pipelined**
  - **— 1 to 16 Wait States Non-Pipelined**
- **DRAM Refresh Controller**
- **80386 Shutdown Detect and Reset Control**
  - **— Software/Hardware Reset**
- **High Speed CHMOS III Technology**
- **132-Pin PGA Package**
- **Optimized for use with the 80386 Microprocessor**
  - **— Resides on Local Bus for Maximum Bus Bandwidth**

The 82380 is a multi-function support peripheral that integrates system functions necessary in an 80386 environment. It has eight channels of high performance 32-bit DMA with the most efficient transfer rates possible on the 80386 bus. System support peripherals integrated into the 82380 provide Interrupt Control, Timers, Wait State generation, DRAM Refresh Control, and System Reset logic.

The 82380's DMA Controller can transfer data between devices of different data path widths using a single channel. Each DMA channel operates independently in any of several modes. Each channel has a temporary data storage register for handling non-aligned data without the need for external alignment logic.



290128-1

82380 Internal Block Diagram

October 1990
Order Number: 290128-005

TABLE OF CONTENTS

CONTENTS

PAGE

1.0 FUNCTIONAL OVERVIEW 5-1085
1.1 82380 Architecture 5-1085
1.1.1 DMA Controller 5-1086
1.1.2 Programmable Interval Timers 5-1087
1.1.3 Interrupt Controller 5-1088
1.1.4 Wait State Generator 5-1089
1.1.5 DRAM Refresh Controller 5-1089
1.1.6 CPU Reset Function 5-1090
1.1.7 Register Map Relocation 5-1090
1.2 Host Interface 5-1090
1.3 IBM-PC System Compatibility 5-1091

2.0 80386 HOST INTERFACE 5-1091
2.1 Master and Slave Modes 5-1092
2.2 80386 Interface Signals 5-1092
2.2.1 Clock (CLK2) 5-1092
2.2.2 Data Bus (D0–D31) 5-1092
2.2.3 Address Bus (A31–A2) 5-1093
2.2.4 Byte Enable (BE3#–BE0#) 5-1093
2.2.5 Bus Cycle Definition Signals (D/C#, W/R#, M/IO#) 5-1094
2.2.6 Address Status (ADS#) 5-1094
2.2.7 Transfer Acknowledge (READY#) 5-1094
2.2.8 Next Address Request (NA#) 5-1094
2.2.9 Reset (RESET, CPURST) 5-1094
2.2.10 Interrupt Out (INT) 5-1095
2.3 82380 Bus Timing 5-1095
2.3.1 Address Pipelining 5-1096
2.3.2 Master Mode Bus Timing 5-1096
2.3.3 Slave Mode Bus Timing 5-1099

3.0 DMA CONTROLLER 5-1100
3.1 Functional Description 5-1101
3.2 Interface Signals 5-1102
3.2.1 DREQn and EDACK (0–2) 5-1103
3.2.2 HOLD and HLDA 5-1103
3.2.3 EOP# 5-1103
3.3 Modes of Operation 5-1103
3.3.1 Target/Requester Definition 5-1104
3.3.2 Buffer Transfer Processes 5-1104
3.3.3 Data Transfer Modes 5-1105
3.3.4 Channel Priority Arbitration 5-1109
3.3.5 Combining Priority Modes 5-1111
3.3.6 Bus Operation 5-1112
3.3.6.1 Fly-By Transfers 5-1112
3.3.6.2 Two-Cycle Transfers 5-1112
3.3.6.3 Data Path Width and Data Transfer Rate Considerations 5-1113
3.3.6.4 Read, Write, and Verify Cycles 5-1113
3.4 Bus Arbitration and Handshaking 5-1113
3.4.1 Synchronous and Asynchronous Sampling of DREQn and EOP# 5-1116
3.4.2 Arbitration of Cascaded Master Requests 5-1118
3.4.3 Arbitration of Refresh Requests 5-1120

5

CONTENTS

PAGE

3.0 DMA CONTROLLER (Continued)
3.5 DMA Controller Register Overview 5-1120
3.5.1 Control/Status Registers 5-1120
3.5.2 Channel Registers 5-1121
3.5.3 Temporary Registers 5-1122
3.6 DMA Controller Programming 5-1123
3.6.1 Buffer Processes 5-1123
3.6.1.1 Single Buffer Process 5-1123
3.6.1.2 Buffer Auto-Initialize Process 5-1123
3.6.1.3 Buffer Chaining Process 5-1123
3.6.2 Data Transfer Modes 5-1124
3.6.3 Cascaded Bus Masters 5-1124
3.6.4 Software Commands 5-1124
3.7 Register Definitions 5-1125
3.8 8237A Compatibility 5-1131

4.0 PROGRAMMABLE INTERRUPT CONTROLLER 5-1132
4.1 Functional Description 5-1132
4.1.1 Internal Block Diagram 5-1132
4.1.2 Interrupt Controller Banks 5-1133
4.2 Interface Signals 5-1134
4.2.1 Interrupt Inputs 5-1134
4.2.2 Interrupt Output (INT) 5-1135
4.3 Bus Functional Description 5-1135
4.4 Modes of Operation 5-1136
4.4.1 End-Of-Interrupt 5-1136
4.4.2 Interrupt Priorities 5-1137
4.4.2.1 Fully Nested Mode 5-1137
4.4.2.2 Automatic Rotation—Equal Priority Devices 5-1138
4.4.2.3 Specific Rotation—Specific Priority 5-1139
4.4.2.4 Interrupt Priority Mode Summary 5-1139
4.4.3 Interrupt Masking 5-1140
4.4.4 Edge Or Level Interrupt Triggering 5-1140
4.4.5 Interrupt Cascading 5-1140
4.4.5.1 Special Fully Nested Mode 5-1141
4.4.6 Reading Interrupt Status 5-1141
4.4.6.1 Poll Command 5-1141
4.4.6.2 Reading Interrupt Registers 5-1141
4.5 Register Set Overview 5-1141
4.5.1 Initialization Command Words (ICW) 5-1143
4.5.2 Operation Control Words (OCW) 5-1143
4.5.3 Poll/Interrupt Request/In-Service Status Register 5-1144
4.5.4 Interrupt Mask Register (IMR) 5-1144
4.5.5 Vector Register (VR) 5-1144
4.6 Programming 5-1144
4.6.1 Initialization (ICW) 5-1144
4.6.2 Vector Registers (VR) 5-1145
4.6.3 Operation Control Words (OCW) 5-1145
4.6.3.1 Read Status And Poll Commands (OCW3) 5-1145
4.7 Register Bit Definition 5-1146
4.8 Register Operational Summary 5-1149

CONTENTS

PAGE

5.0 PROGRAMMABLE INTERVAL TIMER 5-1150
5.1 Functional Description 5-1150
5.1.1 Internal Architecture 5-1151
5.2 Interface Signals 5-1152
5.2.1 CLKIN 5-1152
5.2.2 TOUT1, TOUT2#, TOUT3# 5-1152
5.2.3 GATE 5-1152
5.3 Modes of Operation 5-1153
5.3.1 Mode 0—Interrupt On Terminal Count 5-1153
5.3.2 Mode 1—GATE Retriggerable One-Shot 5-1153
5.3.3 Mode 2—Rate Generator 5-1155
5.3.4 Mode 3—Square Wave Generator 5-1156
5.3.5 Mode 4—Initial Count Triggered Strobe 5-1158
5.3.6 Mode 5—GATE Retriggerable Strobe 5-1159
5.3.7 Operation Common to All Modes 5-1160
5.3.7.1 GATE 5-1160
5.3.7.2 Counter 5-1160
5.4 Register Set Overview 5-1160
5.4.1 Counter 0, 1, 2, 3 Registers 5-1161
5.4.2 Control Word Register I & II 5-1161
5.5 Programming 5-1161
5.5.1 Initialization 5-1161
5.5.2 Read Operation 5-1161
5.6 Register Bit Definitions 5-1163

6.0 WAIT STATE GENERATOR 5-1165
6.1 Functional Description 5-1165
6.2 Interface Signals 5-1166
6.2.1 READY# 5-1166
6.2.2 READYO# 5-1166
6.2.3 WSC(0–1) 5-1166
6.3 Bus Function 5-1167
6.3.1 Wait States in Non-Pipelined Cycle 5-1167
6.3.2 Wait States in Pipelined Cycle 5-1168
6.3.3 Extending and Early Terminating Bus Cycle 5-1169
6.4 Register Set Overview 5-1170
6.5 Programming 5-1171
6.6 Register Bit Definition 5-1171
6.7 Application Issues 5-1171
6.7.1 External 'READY' Control Logic 5-1171

7.0 DRAM REFRESH CONTROLLER 5-1173
7.1 Functional Description 5-1173
7.2 Interface Signals 5-1173
7.2.1 TOUT1/REF# 5-1173
7.3 Bus Function 5-1174
7.3.1 Arbitration 5-1174
7.4 Modes of Operation 5-1175
7.4.1 Word Size and Refresh Address Counter 5-1175
7.5 Register Set Overview 5-1175
7.6 Programming 5-1175
7.7 Register Bit Definition 5-1175

CONTENTS

PAGE

8.0 RELOCATION REGISTER AND ADDRESS DECODE 5-1175
8.1 Relocation Register 5-1175
8.1.1 I/O-Mapped 82380 5-1176
8.1.2 Memory-Mapped 82380 5-1176
8.2 Address Decoding 5-1176

9.0 CPU RESET AND SHUTDOWN DETECT 5-1176
9.1 Hardware Reset 5-1176
9.2 Software Reset 5-1176
9.3 Shutdown Detect 5-1177

10.0 INTERNAL CONTROL AND DIAGNOSTIC PORTS 5-1177
10.1 Internal Control Port 5-1177
10.2 Diagnostic Ports 5-1177

11.0 INTEL RESERVED I/O PORTS 5-1178

12.0 MECHANICAL DATA 5-1178
12.1 Introduction 5-1178
12.2 Pin Assignment 5-1179
12.3 Package Dimensions and Mounting 5-1181
12.4 Package Thermal Specification 5-1183

13.0 ELECTRICAL DATA 5-1184
13.1 Power and Grounding 5-1184
13.2 Power Decoupling 5-1184
13.3 Unused Pin Recommendations 5-1184
13.4 ICE-386 Support 5-1184
13.5 Maximum Ratings 5-1185
13.6 D.C. Specifications 5-1185
13.7 A.C. Specifications 5-1187

APPENDIX A—Ports Listed by Address 5-1196

APPENDIX B—Ports Listed by Function 5-1200

APPENDIX C—Pin Descriptions 5-1204

APPENDIX D—System Notes 5-1207

1.0 FUNCTIONAL OVERVIEW

The 82380 contains several independent functional modules. The following is a brief discussion of the components and features of the 82380. Each module has a corresponding detailed section later in this data sheet. Those sections should be referred to for design and programming information.

1.1 82380 Architecture

The 82380 is comprised of several computer system functions that are normally found in separate LSI and VLSI components. These include: a high-performance, eight-channel, 32-bit Direct Memory Access Controller; a 20-level Programmable Interrupt Controller which is a superset of the 82C59A; four 16-bit Programmable Interval Timers which are functionally equivalent to the 82C54 timers; a DRAM Refresh Controller; a Programmable Wait State Generator; and system reset logic. The interface to the 82380 is optimized for high-performance operation with the 80386 microprocessor.

The 82380 operates directly on the 80386 bus. In the Slave mode, it monitors the state of the processor at all times and acts or idles according to the commands of the host. It monitors the address pipeline status and generates the programmed number of wait states for the device being accessed. The 82380 also has logic to reset the 80386 via hardware or software reset requests and processor shutdown status.

After a system reset, the 82380 is in the Slave mode. It appears to the system as an I/O device. It becomes a bus master when it is performing DMA transfers.

To maintain compatibility with existing software, the registers within the 82380 are accessed as bytes. If the internal logic of the 82380 requires a delay before another access by the processor, wait states are automatically inserted into the access cycle. This allows the programmer to write initialization routines, etc. without regard to hardware recovery times.

Figure 1-1 shows the basic architectural components of the 82380. The following sections briefly discuss the architecture and function of each of the distinct sections of the 82380.

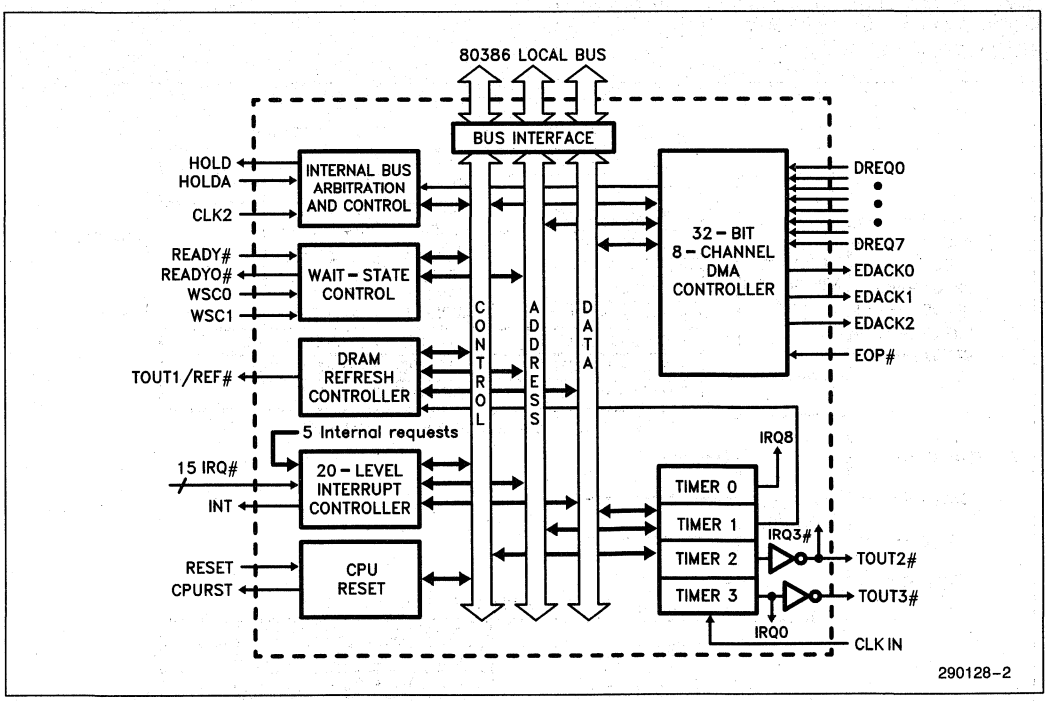


Figure 1-1. Architecture of the 82380

1.1.1 DMA CONTROLLER

The 82380 contains a high-performance, 8-channel, 32-bit DMA controller. It is capable of transferring any combination of bytes, words, and double words. The addresses of both source and distination can be independently incremented, decremented or held constant, and cover the entire 32-bit physical address space of the 80386. It can disassemble and assemble misaligned data via a 32-bit internal temporary data storage register. Data transferred between devices of different data path widths can also be assembled and disassembled using the internal temporary data storage register. The DMA Controller can also transfer aligned data between I/O and memory on the fly, allowing data transfer rates up to 32 megabytes per second for an 82380 operating at 16 MHz. Figure 1-2 illustrates the functional components of the DMA Controller.

There are twenty-four general status and command registers in the 82380 DMA Controller. Through these registers any of the channels may be programmed into any of the possible modes. The operating modes of any one channel are independent of the operation of the other channels.

Each channel has three programmable registers which determine the location and amount of data to be transferred:

Byte Count Register—Number of bytes to transfer. (24-bits)

Requester Register—Address of memory or peripheral which is requesting DMA service. (32-bits)

Target Register—Address of peripheral or memory which will be accessed. (32-bits)

There are also port addresses which, when accessed, cause the 82380 to perform specific functions. The actual data written does not matter, the act of writing to the specific address causes the command to be executed. The commands which operate in this mode are: Master Clear, Clear Terminal Count Interrupt Request, Clear Mask Register, and Clear Byte Pointer Flip-Flop.

DMA transfers can be done between all combinations of memory and I/O; memory-to-memory, memory-to-I/O, I/O-to-memory, and I/O-to-I/O. DMA service can be requested through software and/or hardware. Hardware DMA acknowledge signals are available for all channels (except channel 4) through an encoded 3-bit DMA acknowledge bus (EDACK0-2).

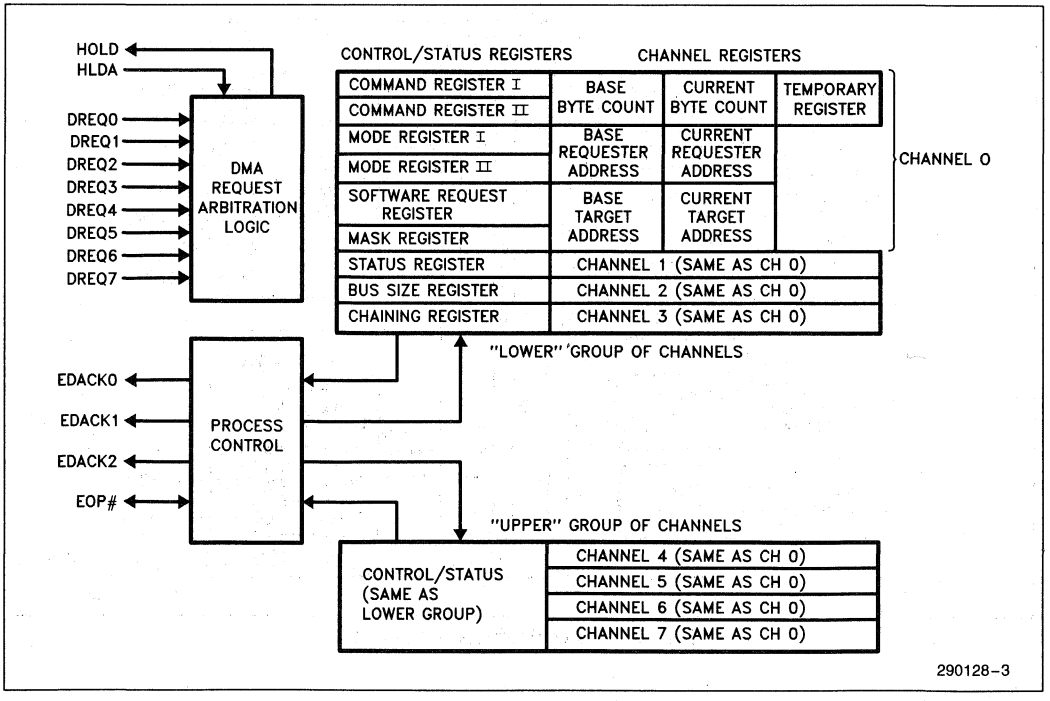


Figure 1-2. 82380 DMA Controller

The 82380 DMA controller transfers blocks of data (buffers) in three modes: Single Buffer, Buffer Auto-Initialize, and Buffer Chaining. In the Single Buffer Process, the 82380 DMA Controller is programmed to transfer one particular block of data. Successive transfers then require reprogramming of the DMA channel. Single Buffer transfers are useful in systems where it is known at the time the transfer begins what quantity of data is to be transferred, and there is a contiguous block of data area available.

The Buffer Auto-Initialize Process allows the same data area to be used for successive DMA transfers without having to reprogram the channel.

The Buffer Chaining Process allows a program to specify a list of buffer transfers to be executed. The 82380 DMA Controller, through interrupt routines, is reprogrammed from the list. The channel is reprogrammed for a new buffer before the current buffer transfer is complete. This pipelining of the channel programming process allows the system to allocate non-contiguous blocks of data storage space, and transfer all of the data with one DMA process. The buffers that make up the chain do not have to be in contiguous locations.

Channel priority can be fixed or rotating. Fixed priority allows the programmer to define the priority of DMA channels based on hardware or other fixed parameters. Rotating priority is used to provide peripherals access to the bus on a shared basis.

With fixed priority, the programmer can set any channel to have the current lowest priority. This allows the user to reset or manually rotate the priority schedule without reprogramming the command registers.

1.1.2 PROGRAMMABLE INTERVAL TIMERS

Four 16-bit programmable interval timers reside within the 82380. These timers are identical in function to the timers in the 82C54 Programmable Interval Timer. All four of the timers share a common clock input which can be independent of the system clock. The timers are capable of operating in six different modes. In all of the modes, the current count can be latched and read by the 80386 at any time, making these very versatile event timers. Figure 1-3 shows the functional components of the Programmable Interval Timers.

The outputs of the timers are directed to key system functions, making system design simpler. Timer 0 is routed directly to an interrupt input and is not available externally. This timer would typically be used to generate time-keeping interrupts.

Timers 1 and 2 have outputs which are available for general timer/counter purposes as well as special functions. Timer 1 is routed to the refresh control logic to provide refresh timing. Timer 2 is connected to an interrupt request input to provide other timer functions. Timer 3 is a general purpose timer/counter whose output is available to external hardware. It is also connected internally to the interrupt request which defaults to the highest priority (IRQ0).

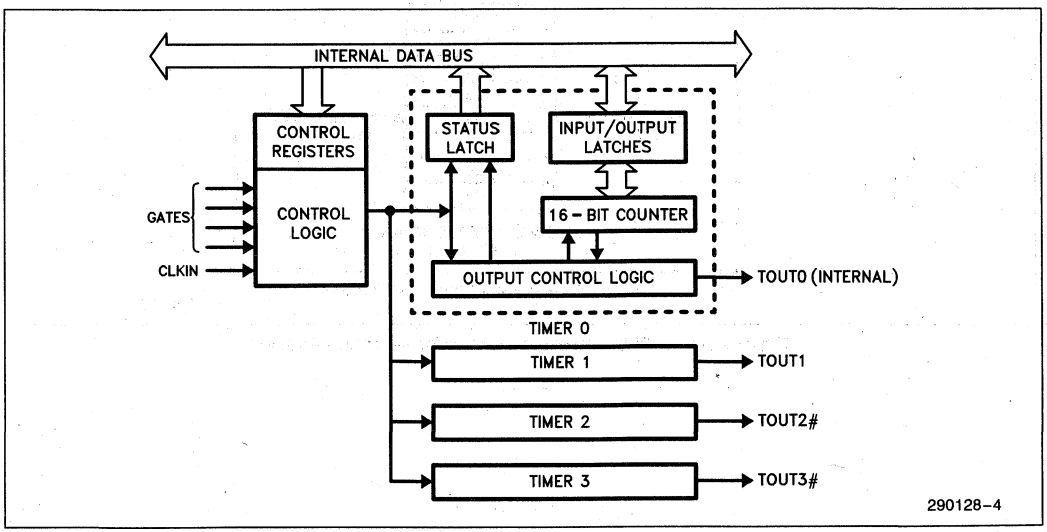


Figure 1-3. Programmable Interval Timers—Block Diagram

1.1.3 INTERRUPT CONTROLLER

The 82380 has the equivalent of three enhanced 82C59A Programmable Interrupt Controllers. These controllers can all be operated in the Master mode, but the priority is always as if they were cascaded. There are 15 interrupt request inputs provided for the user, all of which can be inputs from external slave interrupt controllers. Cascading 82C59As to these request inputs allows a possible total of 120 external interrupt requests. Figure 1-4 is a block diagram of the 82380 Interrupt Controller.

Each of the interrupt request inputs can be individually programmed with its own interrupt vector, allowing more flexibility in interrupt vector mapping than was available with the 82C59A. An interrupt is provided to alert the system that an attempt is being made to program the vectors in the method of the 82C59A. This provides compatibility of existing software that used the 82C59A or 8259A with new designs using the 82380.

In the event of an unrequested or otherwise erroneous interrupt acknowledge cycle, the 82380 Interrupt Controller issues a default vector. This vector, programmed by the system software, will alert the system of unsolicited interrupts of the 80386.

The functions of the 82380 Interrupt Controller are identical to the 82C59A, except in regards to programming the interrupt vectors as mentioned above. Interrupt request inputs are programmable as either edge or level triggered and are software maskable. Priority can be either fixed or rotating and interrupt requests can be nested.

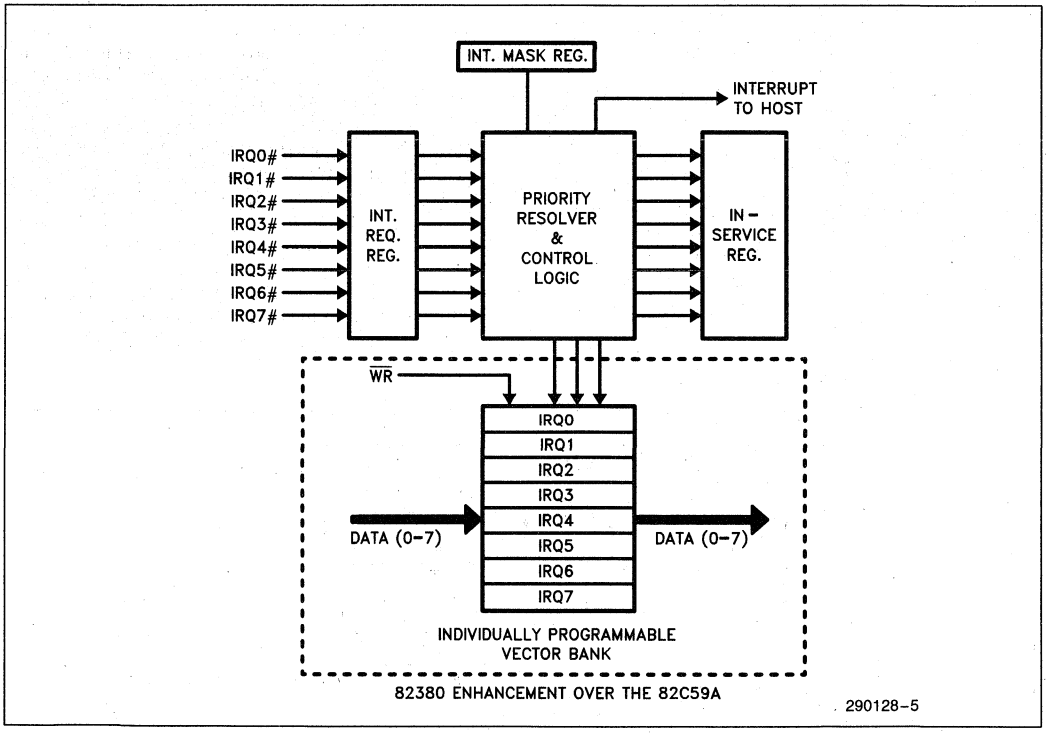


Figure 1-4. 82380 Interrupt Controller—Block Diagram

Enhancements are added to the 82380 for cascading external interrupt controllers. Master to Slave handshaking takes place on the data bus, instead of dedicated cascade lines.

1.1.4 WAIT STATE GENERATOR

The Wait State Generator is a programmable READY generation circuit for the 80386 bus. A peripheral requiring wait states can request the Wait State Generator to hold the processor's READY input inactive for a predetermined number of bus states. Six different wait state counts can be programmed into the Wait State Generator by software; three for memory accesses and three for I/O accesses. A block diagram of the 82380 Wait State Generator is shown in Figure 1-5.

The peripheral being accessed selects the required wait state count by placing a code on a 2-bit wait state select bus. This code along with the M/IO# signal from the bus master is used to select one of six internal 4-bit wait state registers which has been programmed with the desired number of wait states. From zero to fifteen wait states can be programmed into the wait state registers. The Wait State Generator tracks the state of the processor or current bus master at all times, regardless of which device is the current bus master and regardless of whether or not the Wait State Generator is currently active.

The 82380 Wait State Generator is disabled by making the select inputs both high. This allows hardware which is intelligent enough to generate its own ready signal to be accessed without penalty. As previously mentioned, deselecting the Wait State Generator does not disable its ability to determine the proper number of wait states due to pipeline status in subsequent bus cycles.

The number of wait states inserted into a pipelined bus cycle is the value in the selected wait state register. If the bus master is operating in the non-pipelined mode, the Wait State Generator will increase the number of wait states inserted into the bus cycle by one.

On reset, the Wait State Generator's registers are loaded with the value FFH, giving the maximum number of wait states for any access in which the wait state select inputs are active.

1.1.5 DRAM REFRESH CONTROLLER

The 82380 DRAM Refresh Controller consists of a 24-bit refresh address counter and bus arbitration logic. The output of Timer 1 is used to periodically request a refresh cycle. When the controller receives the request, it requests access to the system bus through the HOLD signal. When bus control is acknowledged by the processor or current bus master, the refresh controller executes a memory read operation at the address currently in the Refresh Address Register. At the same time, it activates a refresh signal (REF#) that the memory uses to force a refresh instead of a normal read. Control of the bus is transferred to the processor at the completion of this cycle. Typically a refresh cycle will take six clock cycles to execute on an 80386 bus.

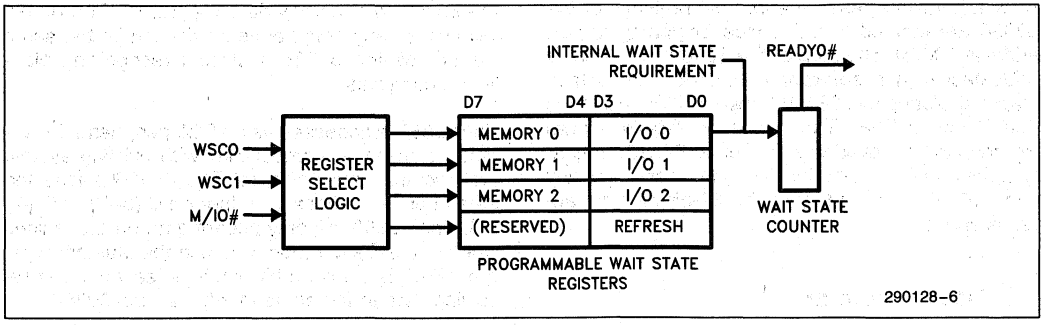


Figure 1-5. 82380 Wait State Generator—Block Diagram

The 82380 DRAM Refresh Controller has the highest priority when requesting bus access and will interrupt any active DMA process. This allows large blocks of data to be moved by the DMA controller without affecting the refresh function. Also the DMA controller is not required to completely relinquish the bus, the refresh controller simply steals a bus cycle between DMA accesses.

The amount by which the refresh address is incremented is programmable to allow for different bus widths and memory bank arrangements.

1.1.6 CPU RESET FUNCTION

The 82380 contains a special reset function which can respond to hardware reset signals from the 82384, as well as a software reset command. The circuit will hold the 80386's RESET line active while an external hardware reset signal is present at its RESET input. It can also reset the 80386 processor as the result of a software command. The software reset command causes the 82380 to hold the processor's RESET line active for a minimum of 62 CLK2 cycles; enough time to allow an 80386 to re-initialize.

The 82380 can be programmed to sense the shutdown detect code on the status lines from the 80386. If the Shutdown Detect function is enabled, the 82380 will automatically reset the processor. A diagnostic register is available which can be used to determine the cause of reset.

1.1.7 REGISTER MAP RELOCATION

After a hardware reset, the internal registers of the 82380 are located in I/O space beginning at port address 0000H. The map of the 82380's registers is relocatable via a software command. The default mapping places the 82380 between I/O addresses 0000H and 00DBH. The relocation register allows this map to be moved to any even 256-byte boundary in the processor's 16-bit I/O address space or any even 16-Mbyte boundary in the 32-bit memory address space.

1.2 Host Interface

The 82380 is designed to operate efficiently on the local bus of an 80386 microprocessor. The control signals of the 82380 are identical in function to those of the 80386. As a slave, the 82380 operates with all of the features available on the 80386 bus. When the 82380 is in the Master mode, it looks identical to the 80386 to the connected devices.

The 82380 monitors the bus at all times, and determines whether the current bus cycle is a pipelined or non-pipelined access. All of the status signals of the processor are monitored.

The control, status, and data registers within the 82380 are located at fixed addresses relative to each other, but the group can be relocated to either memory or I/O space and to different locations within those spaces.

As a Slave device, the 82380 monitors the control/status lines of the CPU. The 82380 will generate all of the wait states it needs whenever it is accessed. This allows the programmer the freedom of accessing 82380 registers without having to insert NOPs in the program to wait for slower 82380 internal registers.

The 82380 can determine if a current bus cycle is a pipelined or a non-pipelined cycle. It does this by monitoring the ADS# and READY# signals and thereby keeping track of the current state of the 80386.

As a bus master, the 82380 looks like an 80386 to the rest of the system. This enables the designer greater flexibility in systems which include the 82380. The designer does not have to alter the interfaces of any peripherals designed to operate with the 80386 to accommodate the 82380. The 82380 will access any peripherals on the bus in the same manner as the 80386, including recognizing pipelined bus cycles.

The 82380 is accessed as an 8-bit peripheral. This is done to maintain compatibility with existing system architectures and software. The 80386 places the data of all 8-bit accesses either on D (0–7) or D (8–15). The 82380 will only accept data on these lines when in the Slave mode. When in the Master mode, the 82380 is a full 32-bit machine, sending and receiving data in the same manner as the 80386.

1.3 IBM PC* System Compatibility

The 82380 is an 80386 companion device designed to provide an enhancement of the system functions common to most small computer systems. It is modeled after and is a superset of the Intel peripheral products found in the IBM PC, PC-AT, and other popular small computers.

2.0 80386 HOST INTERFACE

The 82380 contains a set of interface signals to operate efficiently with the 80386 host processor. These signals were designed so that minimal hardware is needed to connect the 82380 to the 80386.

Figure 2-1 depicts a typical system configuration with the 80386 processor. As shown in the diagram, the 82380 is designed to interface directly with the 80386 bus.

*IBM PC and IBM PC-AT are registered trademarks of International Business Machines Inc.

Since the 82380 is residing on the opposite side of the data bus transceiver (with respect to the rest of the peripherals in the system), it is important to note that the transceiver should be controlled so that contention between the data bus transceiver and the 82380 will not occur. In order to do this, port address decoding logic should be included in the direction and enable control logic of the transceiver. When any of the 82380 internal registers is read, the data bus transceiver should be disabled so that only the 82380 will drive the local bus.

This section describes the basic bus functions of the 82380 to show how this device interacts with the 80386 processor. Other signals which are not directly related to the host interface will be discussed in their associated functional block description.

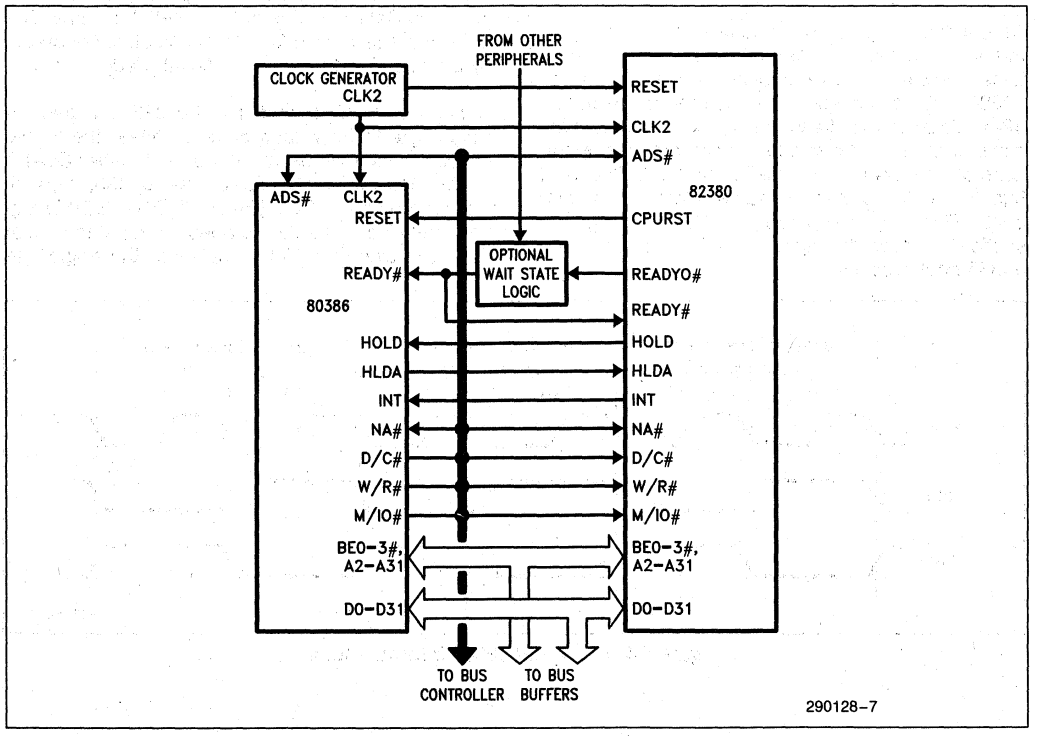


Figure 2-1. 80386/82380 System Configuration

2.1 Master and Slave Modes

At any time, the 82380 acts as either a Slave device or a Master device in the system. Upon reset, the 82380 will be in the Slave Mode. In this mode, the 80386 processor can read/write into the 82380 internal registers. Initialization information may be programmed into the 82380 during Slave Mode.

When DMA service (including DRAM Refresh Cycles generated by the 82380) is requested, the 82380 will request and subsequently get control of the 80386 local bus. This is done through the HOLD and HLDA (Hold Acknowledge) signals. When the 80386 processor responds by asserting the HLDA signal, the 82380 will switch into Master Mode and perform DMA transfers. In this mode, the 82380 is the bus master of the system. It can read/write data from/to memory and peripheral devices. The 82380 will return to the Slave Mode upon completion of DMA transfers, or when HLDA is negated.

2.2 80386 Interface Signals

As mentioned in the Architecture section, the Bus Interface module of the 82380 (see Figure 1-1) contains signals that are directly connected to the 80386 host processor. This module has separate 32-bit Data and Address busses. Also, it has additional control signals to support different bus operations on the system. By residing on the 80386 local bus, the 82380 shares the same address, data and control lines with the processor. The following subsections discuss the signals which interface to the 80386 host processor.

2.2.1 CLOCK (CLK2)

The CLK2 input provides fundamental timing for the 82380. It is divided by two internally to generate the 82380 internal clock. Therefore, CLK2 should be driven with twice the 80386's frequency. In order to maintain synchronization with the 80386 host processor, the 82380 and the 80386 should share a common clock source.

The internal clock consists of two phases: PHI1 and PHI2. Each CLK2 period is a phase of the internal clock. PHI2 is usually used to sample input and set up internal signals and PHI1 is for latching internal data. Figure 2-2 illustrates the relationship of CLK2 and the 82380 internal clock signals. The CPURST signal generated by the 82380 guarantees that the 80386 will wake up in phase with PHI1.

2.2.2 DATA BUS (D0–D31)

This 32-bit three-state bidirectional bus provides a general purpose data path between the 82380 and the system. These pins are tied directly to the corresponding Data Bus pins of the 80386 local bus. The Data Bus is also used for interrupt vectors generated by the 82380 in the Interrupt Acknowledge cycle.

During Slave I/O operations, the 82380 expects a single byte to be written or read. When the 80386 host processor writes into the 82380, either D0–D7 or D8–D15 will be latched into the 82380, depending upon how the Byte Enable (BE0#–BE#3) signals are driven. The 82380 does not need to look at D16–D31 since the 80386 duplicates the single byte

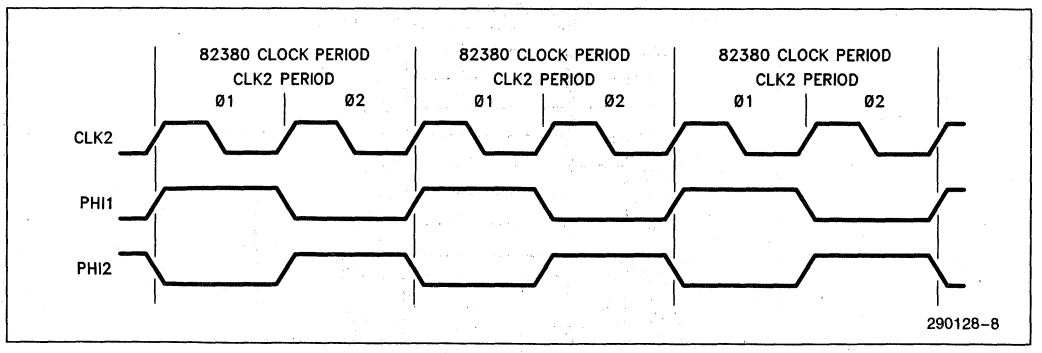


Figure 2-2. CLK2 and 82380 Internal Clock

data on both halves of the bus. When the 80386 host processor reads from the 82380, the single byte data will be duplicated four times on the Data Bus; i.e., on D0–D7, D8–D15, D16–D23 and D24–D31.

During Master Mode, the 82380 can transfer 32-, 16-, and 8-bit data between memory (or I/O devices) and I/O devices (or memory) via the Data Bus.

2.2.3 ADDRESS BUS (A31–A2)

These three-state bidirectional signals are connected directly to the 80386 Address Bus. In the Slave Mode, they are used as input signals so that the processor can address the 82380 internal ports/registers. In the Master Mode, they are used as output signals by the 82380 to address memory and peripheral devices. The Address Bus is capable of addressing 4 G-bytes of physical memory space (00000000H to FFFFFFFFH), and 64 K-bytes of I/O addresses (00000000H to 0000FFFFH).

2.2.4 BYTE ENABLE (BE3#–BE0#)

These bidirectional pins select specific byte(s) in the double word addressed by A31–A2. Similar to the Address Bus function, these signals are used as inputs to address internal 82380 registers during Slave Mode operation. During Master Mode operation, they are used as outputs by the 82380 to address memory and I/O locations.

NOTE:

In addition to the above function, BE3# is used to enable a production test mode and must be LOW during reset. The 80386 processor will automatically hold BE3# LOW during RESET.

The definitions of the Byte Enable signals depend upon whether the 82380 is in the Master or Slave Mode. These definitions are depicted in Table 2-1.

Table 2-1. Byte Enable Signals

As INPUTS (Slave Mode):

BE3#–BE0#	Implied A1, A0	Data Bits Written to 82380*
XXX0	00	D0–D7
XX01	01	D8–D15
X011	10	D0–D7
X111	11	D8–D15

X–DON'T CARE
*During READ, data will be duplicated on D0–D7, D8–D15, D16–D23, and D24–D31.
During WRITE, the 80386 host processor duplicates data on D0–D15, and D16–D31, so that the 82380 is concerned only with the lower half of the Data Bus.

As OUTPUTS (Master Mode):

BE3#–BE0#	Byte to be Accessed Relative to A31–A2	Logical Byte Presented On Data Bus During WRITE Only*			
		D24–31	D16–23	D8–15	D0–7
1110	0	U	U	U	A
1101	1	U	U	A	A
1011	2	U	A	U	A
0111	3	A	U	A	A
1001	1, 2	U	B	A	A
1100	0, 1	U	U	B	A
0011	2, 3	B	A	B	A
1000	0, 1, 2	U	C	B	A
0001	1, 2, 3	C	B	A	A
0000	0, 1, 2, 3	D	C	B	A

U = Undefined
A = Logical D0–D7
B = Logical D8–D15
C = Logical D16–D23
D = Logical D24–D31
*Actual number of bytes accessed depends upon the programmed data path width.

2.2.5 BUS CYCLE DEFINITION SIGNALS (D/C#, W/R#, M/IO#)

These three-state bidirectional signals define the type of bus cycle being performed. W/R# distinguishes between write and read cycles. D/C# distinguishes between processor data and control cycles. M/IO# distinguishes between memory and I/O cycles.

During Slave Mode, these signals are driven by the 80386 host processor; during Master Mode, they are driven by the 82380. In either mode, these signals will be valid when the Address Status (ADS#) is driven LOW. Exact bus cycle definitions are given in Table 2-2. Note that some combinations are recognized as inputs, but not generated as outputs. In the Master Mode, D/C# is always HIGH.

2.2.6 ADDRESS STATUS (ADS#)

This bidirectional signal indicates that a valid address (A2–A31, BE0#–BE3#) and bus cycle definition (W/R#, D/C#, M/IO#) is being driven on the bus. In the Master Mode, it is driven by the 82380 as an output. In the Slave Mode, this signal is monitored as an input by the 82380. By the current and past status of ADS# and the READY# input, the 82380 is able to determine, during Slave Mode, if the next bus cycle is a pipelined address cycle. ADS# is asserted during T1 and T2P bus states (see Bus State Definition).

Note that during the idle states at the beginning and the end of a DMA process, neither the 80386 nor the 82380 is driving the ADS# signal; i.e., the signal is left floated. Therefore, it is important to use a pull-up resistor (approximately 10 KΩ) on the ADS# signal.

2.2.7 TRANSFER ACKNOWLEDGE (READY#)

This input indicates that the current bus cycle is complete. In the Master Mode, assertion of this signal indicates the end of a DMA bus cycle. In the Slave Mode, the 82380 monitors this input and ADS# to detect a pipelined address cycles. This signal should be tied directly to the READY# input of the 80386 host processor.

2.2.8 NEXT ADDRESS REQUEST (NA#)

This input is used to indicate to the 82380 in the Master Mode that the system is requesting address pipelining. When driven LOW by either memory or peripheral devices during Master Mode, it indicates that the system is prepared to accept a new address and bus cycle definition signals from the 82380 before the end of the current bus cycle. If this input is active when sampled by the 82380, the next address is driven onto the bus, provided a bus request is already pending internally.

This input pin is monitored only in the Master Mode. In the Slave Mode, the 82380 uses the ADS# and READY# signals to determine address pipelining cycles, and NA# will be ignored.

2.2.9 RESET (RESET, CPURST)

RESET

This synchronous input suspends any operation in progress and places the 82380 in a known initial state. Upon reset, the 82380 will be in the Slave Mode waiting to be initialized by the 80386 host processor. The 82380 is reset by asserting RESET for 15 or more CLK2 periods. When RESET is asserted, all other input pins are ignored, and all other bus pins are driven to an idle bus state as shown in Table 2-3. The 82380 will determine the phase of its internal clock following RESET going inactive.

Table 2-2. Bus Cycle Definition

M/IO#	D/C#	W/R#	As INPUTS	As OUTPUTS
0	0	0	Interrupt Acknowledge	NOT GENERATED
0	0	1	UNDEFINED	NOT GENERATED
0	1	0	I/O Read	I/O Read
0	1	1	I/O Write	I/O Write
1	0	0	UNDEFINED	NOT GENERATED
1	0	1	HALT if BE(3–0) # = X011 SHUTDOWN if BE (3–0)# = XXX0	NOT GENERATED
1	1	0	Memory Read	Memory Read
1	1	1	Memory Write	Memory Write

Table 2-3. Output Signals Following RESET

Signal	Level
A2–A31, D0–D31, BE0#–BE3#	Float
D/C#, W/R#, M/IO#, ADS#	Float
READYO#	'1'
EOP#	'1' (Weak Pull-UP)
EDACK2–EDACK0	'100'
HOLD	'0'
INT	UNDEFINED*
TOUT1/REF#, TOUT2#/IRQ3#, TOUT3#	UNDEFINED*
CPURST	'0'

*The Interrupt Controller and Programmable Interval Timer are initialized by software commands.

RESET is level-sensitive and must be synchronous to the CLK2 signal. Therefore, this RESET input should be tied to the RESET output of the Clock Generator. The RESET setup and hold time requirements are shown in Figure 2.3.

CPURST

This output signal is used to reset the 80386 host processor. It will go active (HIGH) whenever one of the following events occurs: a) 82380's RESET input is active; b) a software RESET command is issued to the 82380; or c) when the 82380 detects a processor Shutdown cycle and when this detection feature is enabled (see CPU Reset and Shutdown Detect). When activated, CPURST will be held active for 62 CLK2 periods. The timing of CPURST is such that the 80386 processor will be in synchronization with the 82380. This timing is shown in Figure 2-4.

2.2.10 INTERRUPT OUT (INT)

This output pin is used to signal the 80386 host processor that one or more interrupt requests (either internal or external) are pending. The processor is expected to respond with an Interrupt Acknowledge cycle. This signal should be connected directly to the Maskable Interrupt Request (INTR) input of the 80386 host processor.

2.3 82380 Bus Timing

The 82380 internally divides the CLK2 signal by two to generate its internal clock. Figure 2-2 shows the relationship of CLK2 and the internal clock. The internal clock consists of two phases: PHI1 and PHI2. Each CLK2 period is a phase of the internal clock. In Figure 2-2, both PHI1 and PHI2 of the 82380 internal clock are shown.

5

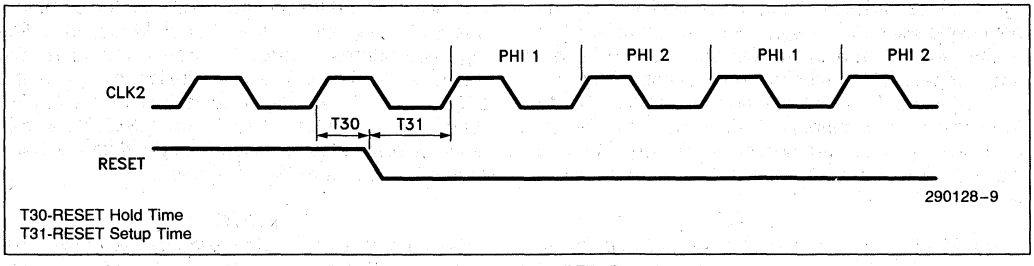


Figure 2-3. RESET Timing

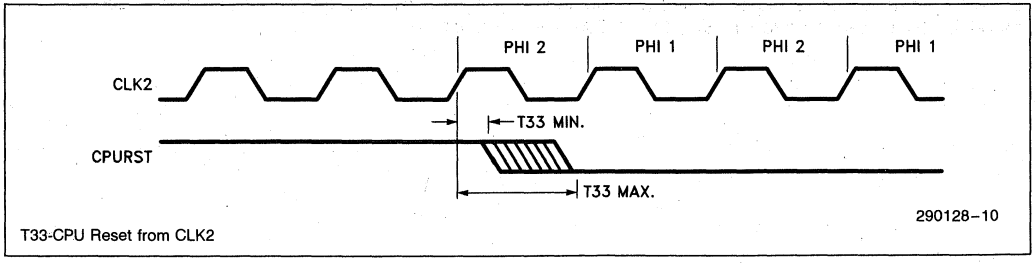


Figure 2-4. CPURST Timing

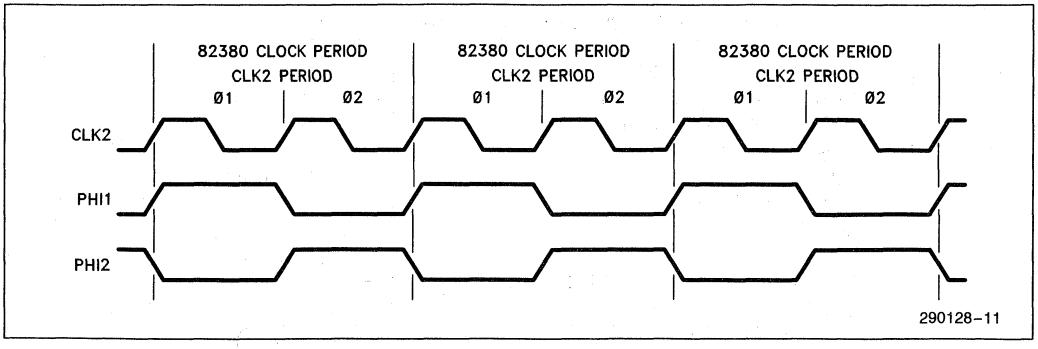


Figure 2-2. CLK2 and 82380 Internal Clock

In the 82380, whether it is in the Master or Slave Mode, the shortest time unit of bus activity is a bus state. A bus state, which is also referred as a 'T-state', is defined as one 82380 PHI2 clock period (i.e., two CLK2 periods). Recall in Table 2-2, there are six different types of bus cycles in the 82380 as defined by the M/IO#, D/C# and W/R# signals. Each of these bus cycles is composed of two or more bus states. The length of a bus cycle depends on when the READY# input is asserted (i.e., driven LOW).

2.3.1 ADDRESS PIPELINING

The 82380 supports Address Pipelining as an option in both the Master and Slave Mode. This feature typically allows a memory or peripheral device to operate with one less wait state than would otherwise be required. This is possible because during a pipelined cycle, the address and bus cycle definition of the next cycle will be generated by the bus master while waiting for the end of the current cycle to be acknowledged. The pipelined bus is especially well suited for interleaved memory environment. For 16 MHz interleaved memory designs with 100 ns access time DRAMs, zero wait state memory accesses can be achieved when pipelined addressing is selected.

In the Master Mode, the 82380 is capable of initiating, on a cycle-by-cycle basis, either a pipelined or non-pipelined access depending upon the state of the NA# input. If a pipelined cycle is requested (indicated by NA# being driven LOW), the 82380 will drive the address and bus cycle definition of the next cycle as soon as there is an internal bus request pending.

In the Slave Mode, the 82380 is constantly monitoring the ADS# and READY# signals on the processor local bus to determine if the current bus cycle is a pipelined cycle. If a pipelined cycle is detected, the 82380 will request one less wait state from the processor if the Wait State Generator feature is selected. On the other hand, during an 82380 internal register access in a pipelined cycle, it will make use of the advance address and bus cycle information. In all cases, Address Pipelining will result in a savings of one wait state.

2.3.2 MASTER MODE BUS TIMING

When the 82380 is in the Master Mode, it will be in one of six bus states. Figure 2-5 shows the complete bus state diagram of the Master Mode, including pipelined address states. As seen in the figure, the 82380 state diagram is very similar to that of the 80386. The major difference is that in the 82380, there is no Hold state. Also, in the 82380, the conditions for some state transitions depend upon whether it is the end of a DMA process*.

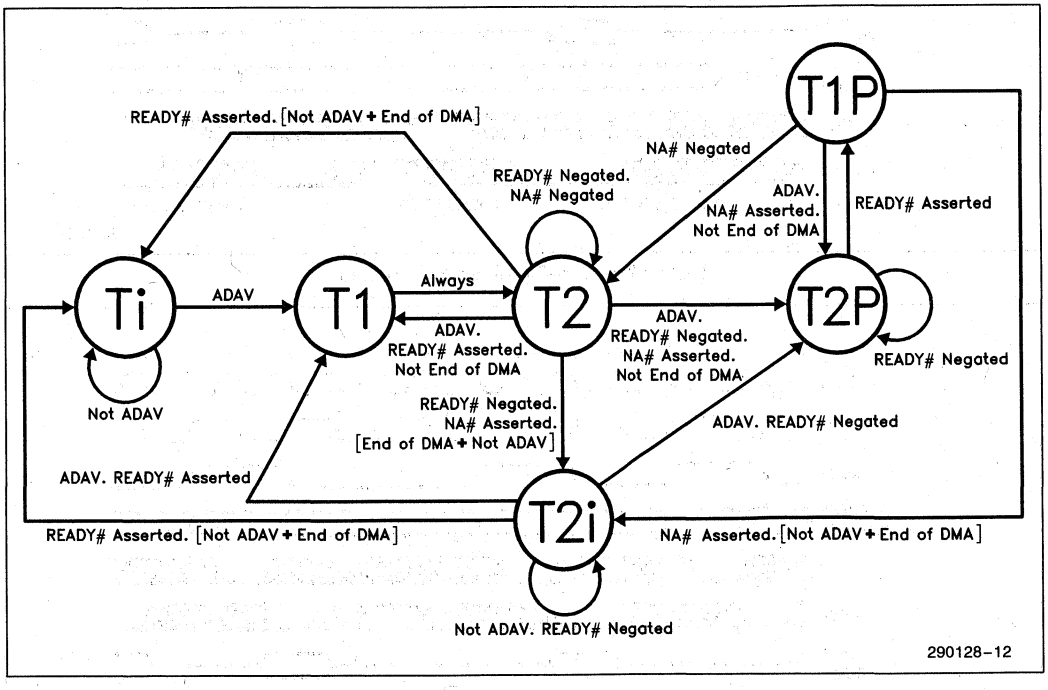
NOTE:

*The term 'end of a DMA process' is loosely defined here. It depends on the DMA modes of operation as well as the state of the EOP# and DREQ inputs. This is explained in detail in section 3—DMA Controller.

The 82380 will enter the idle state, Ti, upon RESET and whenever the internal address is not available at the end of a DMA cycle or at the end of a DMA process. When address pipelining is not used (NA# is not asserted), a new bus cycle always begins with state T1. During T1, address and bus cycle definition signals will be driven on the bus. T1 is always followed by T2.

If a bus cycle is not acknowledged (with READY#) during T2 and NA# is negated, T2 will be repeated. When the end of the bus cycle is acknowledged during T2, the following state will be T1 of the next bus cycle (if the internal address latch is loaded and if this is not the end of the DMA process). Otherwise, the Ti state will be entered. Therefore, if the memory or peripheral accessed is fast enough to respond within the first T2, the fastest non-pipelined cycle will take one T1 and one T2 state.

Use of the address pipelining feature allows the 82380 to enter three additional bus states: T1P, T2P, and T2i. T1P is the first bus state of a pipelined bus cycle. T2P follows T1P (or T2) if NA# is asserted when sampled. The 82380 will drive the bus with the address and bus cycle definition signals of the next cycle during T2P. From the state diagram, it can be seen that after an idle state Ti, the first bus cycle must begin with T1, and is therefore a non-pipelined bus cycle. The next bus cycle can be pipelined if NA# is asserted and the previous bus cycle ended in a T2P state. Once the 82380 is in a pipelined cycle and provided that NA# is asserted in subsequent cycles, the 82380 will be switching between T1P and T2P states. If the end of the current bus cycle is not acknowledged by the READY# input, the 82380 will extend the cycle by adding T2P states. The fastest pipelined cycle will consist of one T1P and one T2P state.



NOTE:
ADAV—Internal Address Available

Figure 2-5. Master Mode State Diagram

The 82380 will enter state T2i when NA# is asserted and when one of the following two conditions occurs. The first condition is when the 82380 is in state T2. T2i will be entered if READY# is not asserted and there is no next address available. This situation is similar to a wait state. The 82380 will stay in T2i for as long as this condition exists. The second condition which will cause the 82380 enter T2i is when the 82380 is in state T1P. Before going to state T2P, the 82380 needs to wait in state T2i until the next address is available. Also, in both cases, if the DMA process is complete, the 82380 will enter the T2i state in order to finish the current DMA cycle.

Figure 2-6 is a timing diagram showing non-pipelined bus accesses in the Master Mode. Figure 2-7 shows the timing of pipelined accesses in the Master Mode.

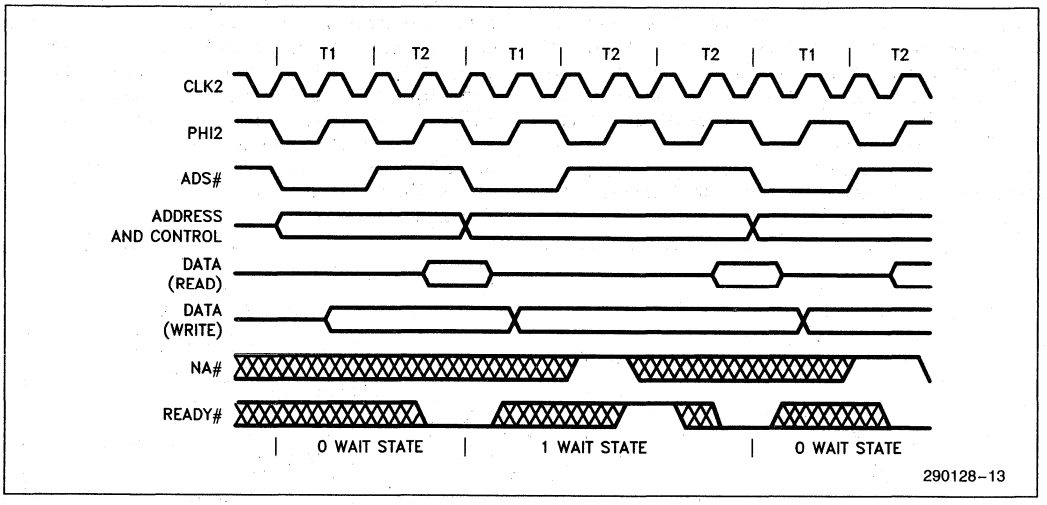


Figure 2-6. Non-Pipelined Bus Cycles

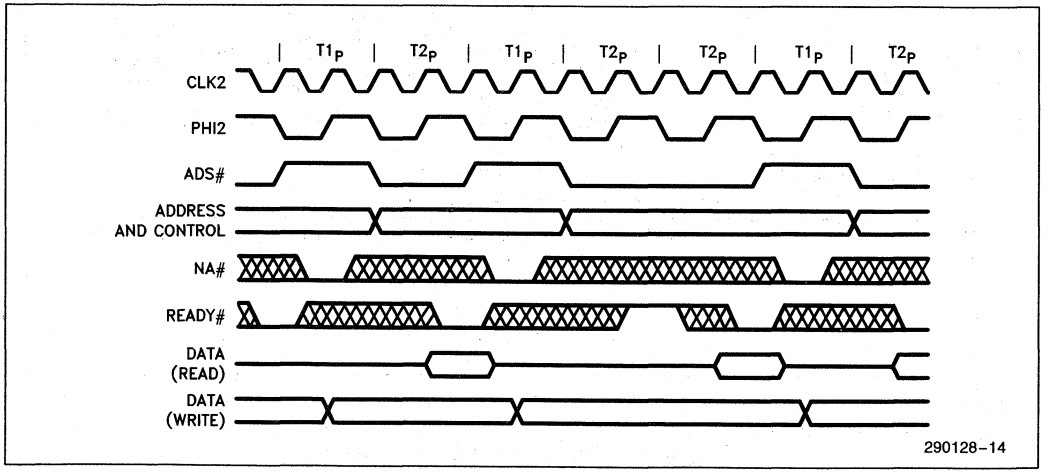
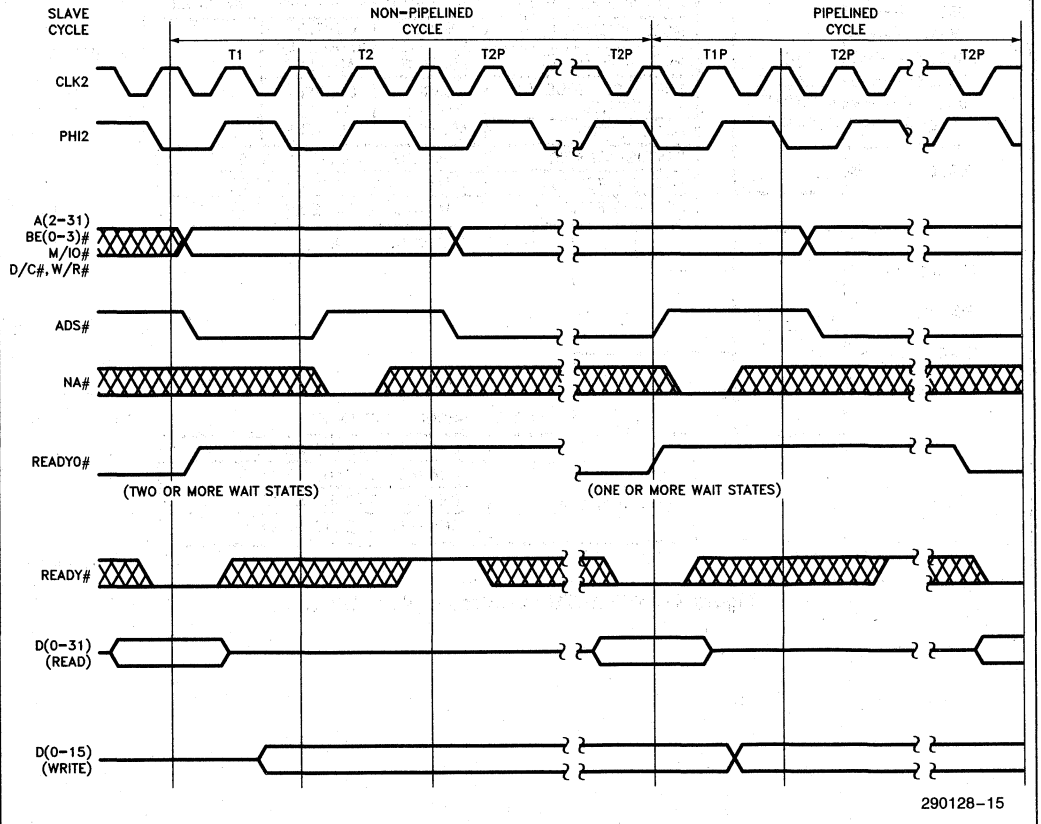


Figure 2-7. Pipelined Bus Cycles

2.3.3 SLAVE MODE BUS TIMING

Figure 2-8 shows the Slave Mode bus timing in both pipelined and non-pipelined cycles when the 82380 is being accessed. Recall that during Slave Mode, the 82380 will constantly monitor the ADS# and READY# signals to determine if the next cycle is pipelined. In Figure 2-8, the first cycle is non-pipelined and the second cycle is pipelined. In the pipelined cycle, the 82380 will start decoding the address and bus cycle signals one bus state earlier than in a non-pipelined cycle.

The READY# input signal is sampled by the 80386 host processor to determine the completion of a bus cycle. This occurs during the end of every T2 and T2P state. Normally, the output of the 82380 Wait State Generator, READYO#, is directly connected to the READY# input of the 80386 host processor and the 82380. In such case, READYO# and READY# will be identical (see Wait State Generator).



NOTE:
NA# is shown here only for timing reference. It is not sampled by the 82380 during Slave Mode.
When the 82380 registers are accessed, it will take one or more wait states in pipelined and two or more wait states in non-pipelined cycle to complete the internal access.

Figure 2-8. Slave Read/Write Timing

3.0 DMA Controller

The 82380 DMA Controller is capable of transferring data between any combination of memory and/or I/O, with any combination (8-, 16-, or 32-bits) of data path widths. Bus bandwidth is optimized through the use of an internal temporary register which can disassemble or assemble data to or from either an aligned or a non-aligned destination or source. Figure 3-1 is a block diagram of the 82380 DMA Controller.

The 82380 has eight channels of DMA. Each channel operates independently of the others. Within the operation of the individual channels, there are many different modes of data transfer available. Many of the operating modes can be intermixed to provide a very versatile DMA controller.

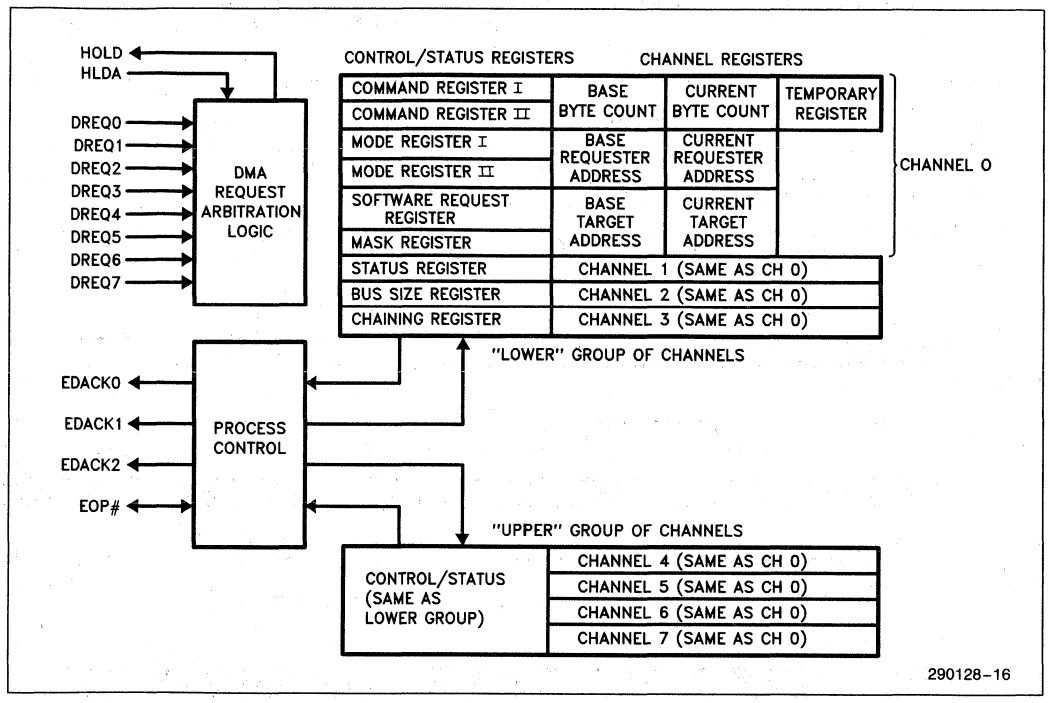


Figure 3-1. 82380 DMA Controller Block Diagram

3.1 Functional Description

In describing the operation of the 82380's DMA Controller, close attention to terminology is required. Before entering the discussion of the function of the 82380 DMA Controller, the following explanations of some of the terminology used herein may be of benefit. First, a few terms for clarification:

DMA PROCESS—A DMA process is the execution of a programmed DMA task from beginning to end. Each DMA process requires initial programming by the host 80386 microprocessor.

BUFFER—A contiguous block of data.

BUFFER TRANSFER—The action required by the DMA to transfer an entire buffer.

DATA TRANSFER—The DMA action in which a group of bytes, words, or double words are moved between devices by the DMA Controller. A data transfer operation may involve movement of one or many bytes.

BUS CYCLE—Access by the DMA to a single byte, word, or double word.

Each DMA channel consists of three major components. These components are identified by the contents of programmable registers which define the memory or I/O devices being serviced by the DMA. They are the Target, the Requester, and the Byte Count. They will be defined generically here and in greater detail in the DMA register definition section.

The Requester is the device which requires service by the 82380 DMA Controller, and makes the request for service. All of the control signals which the DMA monitors or generates for specific channels are logically related to the Requester. Only the Requester is considered capable of initiating or terminating a DMA process.

The Target is the device with which the Requester wishes to communicate. As far as the DMA process is concerned, the Target is a slave which is incapable of control over the process.

The direction of data transfer can be either from Requester to Target or from Target to Requester; i.e., each can be either a source or a destination.

The Requester and Target may each be either I/O or memory. Each has an address associated with it that can be incremented, decremented, or held constant. The addresses are stored in the Requester Address Registers and Target Address Registers, respectively. These registers have two parts: one which contains the current address being used in the DMA process (Current Address Register), and one which holds the programmed base address (Base Address Register). The contents of the Base Registers are never changed by the 82380 DMA Controller. The Current Registers are incremented or decremented according to the progress of the DMA process.

The Byte Count is the component of the DMA process which dictates the amount of data which must be transferred. Current and Base Byte Count Registers are provided. The Current Byte Count Register is decremented once for each byte transferred by the DMA process. When the register is decremented past zero, the Byte Count is considered 'expired' and the process is terminated or restarted, depending on the mode of operation of the channel. The point at which the Byte Count expires is called 'Terminal Count' and several status signals are dependent on this event.

Each channel of the 82380 DMA Controller also contains a 32-bit Temporary Register for use in assembling and disassembling non-aligned data. The operation of this register is transparent to the user, although the contents of it may affect the timing of some DMA handshake sequences. Since there is data storage available for each channel, the DMA Controller can be interrupted without loss of data.

The 82380 DMA Controller is a slave on the bus until a request for DMA service is received via either a software request command or a hardware request signal. The host processor may access any of the control/status or channel registers at any time the 82380 is a bus slave. Figure 3-2 shows the flow of operations that the DMA Controller performs.

At the time a DMA service request is received, the DMA Controller issues a bus hold request to the host processor. The 82380 becomes the bus master when the host relinquishes the bus by asserting a hold acknowledge signal. The channel to be serviced will be the one with the highest priority at the time the DMA Controller becomes the bus master. The DMA Controller will remain in control of the bus until the hold acknowledge signal is removed, or until the current DMA transfer is complete.

While the 82380 DMA Controller has control of the bus, it will perform the required data transfer(s). The type of transfer, source and destination addresses, and amount of data to transfer are programmed in the control registers of the DMA channel which received the request for service.

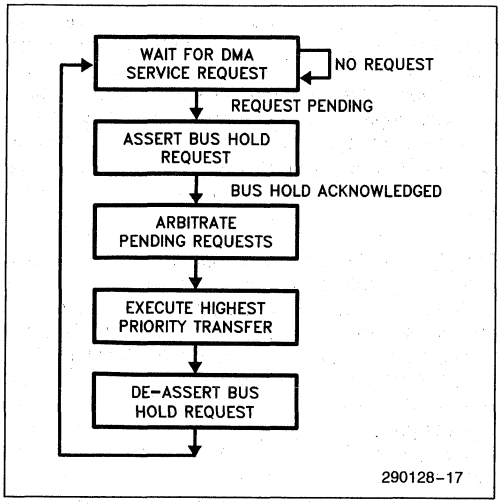


Figure 3-2. Flow of DMA Controller Operation

At completion of the DMA process, the 82380 will remove the bus hold request. At this time the 82380 becomes a slave again, and the host returns to being a master. If there are other DMA channels with requests pending, the controller will again assert the hold request signal and restart the bus arbitration and switching process.

3.2 Interface Signals

There are fourteen control signals dedicated to the DMA process. They include eight DMA Channel Requests (DREQn), three Encoded DMA Acknowledge signals (EDACKn), Processor Hold and Hold Acknowledge (HOLD, HLDA), and End-Of-Process (EOP#). The DREQn inputs and EDACK(0–2) outputs are handshake signals to the devices requiring DMA service. The HOLD output and HLDA input are handshake signals to the host processor. Figure 3-3 shows these signals and how they interconnect between the 82380 DMA Controller, and the Requester and Target devices.

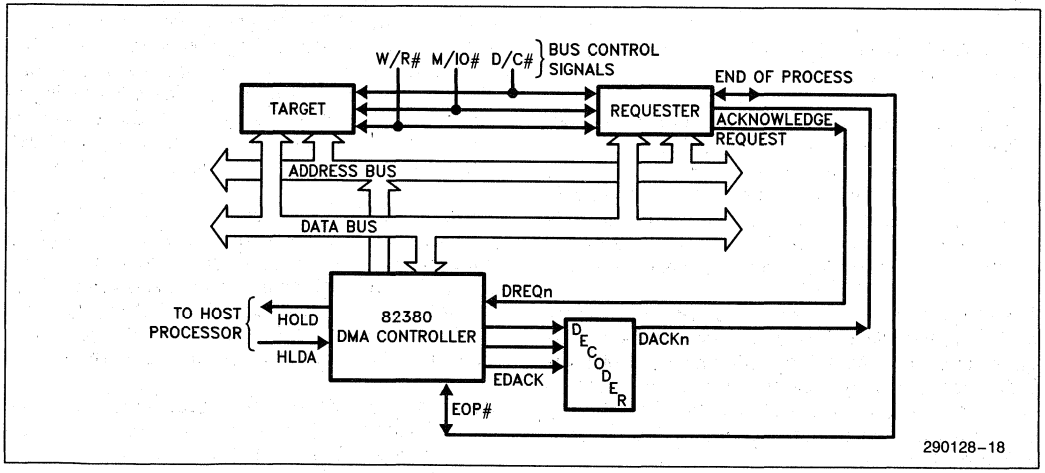


Figure 3-3. Requester, Target, and DMA Controller Interconnection (2-Cycle Configuration)

3.2.1 DREQn and EDACK(0–2)

These signals are the handshake signals between the peripheral and the 82380. When the peripheral requires DMA service, it asserts the DREQn signal of the channel which is programmed to perform the service. The 82380 arbitrates the DREQn against other pending requests and begins the DMA process after finishing other higher priority processes.

When the DMA service for the requested channel is in progress, the EDACK(0–2) signals represent the DMA channel which is accessing the Requester. The 3-bit code on the EDACK(0–2) lines indicates the number of the channel presently being serviced. Table 3-2 shows the encoding of these signals. Note that Channel 4 does not have a corresponding hardware acknowledge.

The DMA acknowledge (EDACK) signals indicate the active channel only during DMA accesses to the Requester. During accesses to the Target, EDACK(0–2) has the idle code (100). EDACK(0–2) can thus be used to select a Requester device during a transfer.

Table 3-2. EDACK Encoding During a DMA Transfer

EDACK2	EDACK1	EDACK0	Active Channel
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	Target Access
1	0	1	5
1	1	0	6
1	1	1	7

DREQn can be programmed as either an Asynchronous or Synchronous input. See section 3.4.1 for details on synchronous versus asynchronous operation of this pin.

The EDACKn signals are always active. They either indicate 'no acknowledge' or they indicate a bus access to the requester. The acknowledge code is either 100, for an idle DMA or during a DMA access to the Target, or 'n' during a Requester access, where n is the binary value representing the channel. A simple 3-line to 8-line decoder can be used to provide discrete acknowledge signals for the peripherals.

3.2.2 HOLD and HLDA

The Hold Request (HOLD) and Hold Acknowledge (HLDA) signals are the handshake signals between the DMA Controller and the host processor. HOLD is an output from the 82380 and HLDA is an input. HOLD is asserted by the DMA Controller when there is a pending DMA request, thus requesting the processor to give up control of the bus so the DMA process can take place. The 80386 responds by asserting HLDA when it is ready to relinquish control of the bus.

The 82380 will begin operations on the bus one clock cycle after the HLDA signal goes active. For this reason, other devices on the bus should be in the slave mode when HLDA is active.

HOLD and HLDA should not be used to gate or select peripherals requesting DMA service. This is because of the use of DMA-like operations by the DRAM Refresh Controller. The Refresh Controller is arbitrated with the DMA Controller for control of the bus, and refresh cycles have the highest priority. A refresh cycle will take place between DMA cycles without relinquishing bus control. See section 3.4.3 for a more detailed discussion of the interaction between the DMA Controller and the DRAM Refresh Controller.

3.2.3 EOP#

EOP# is a bi-directional signal used to indicate the end of a DMA process. The 82380 activates this as an output during the T2 states of the last Requester bus cycle for which a channel is programmed to execute. The Requester should respond by either withdrawing its DMA request, or interrupting the host processor to indicate that the channel needs to be programmed with a new buffer. As an input, this signal is used to tell the DMA Controller that the peripheral being serviced does not require any more data to be transferred. This indicates that the current buffer is to be terminated.

EOP# can be programmed as either an Asynchronous or a Synchronous input. See section 3.4.1 for details on synchronous versus asynchronous operation of this pin.

3.3 Modes of Operation

The 82380 DMA Controller has many independent operating functions. When designing peripheral interfaces for the 82380 DMA Controller, all of the functions or modes must be considered. All of the channels are independent of each other (except in priority of operation) and can operate in any of the modes. Many of the operating modes, though independently programmable, affect the operation of other modes. Because of the large number of com-

binations possible, each programmable mode is discussed here with its affects on the operation of other modes. The entire list of possible combinations will not be presented.

Table 3-1 shows the categories of DMA features available in the 82380. Each of the five major categories is independent of the others. The sub-categories are the available modes within the major function or mode category. The following sections explain each mode or function and its relation to other features.

Table 3-1. DMA Operating Modes

I. **Target/Requester Definition**
- a. Data Transfer Direction
- b. Device Type
- c. Increment/Decrement/Hold

II. **Buffer Processes**
- a. Single Buffer Process
- b. Buffer Auto-Initialize Process
- c. Buffer Chaining Process

III. **Data Transfer/Handshake Modes**
- a. Single Transfer Mode
- b. Demand Transfer Mode
- c. Block Transfer Mode
- d. Cascade Mode

IV. **Priority Arbitration**
- a. Fixed
- b. Rotating
- c. Programmable Fixed

V. **Bus Operation**
- a. Fly-By (Single-Cycle)/Two-Cycle
- b. Data Path Width
- c. Read, Write, or Verify Cycles

3.3.1 TARGET/REQUESTER DEFINITION

All DMA transfers involve three devices: the DMA Controller, the Requester, and the Target. Since the devices to be accessed by the DMA Controller vary widely, the operating characteristics of the DMA Controller must be tailored to the Requester and Target devices.

The Requester can be defined as either the source or the destination of the data to be transferred. This is done by specifying a Write or a Read transfer, respectively. In a Read transfer, the Target is the data source and the Requester is the destination for the data. In a Write transfer, the Requester is the source and the Target in the destination.

The Requester and Target addresses can each be independently programmed to be incremented, decremented, or held constant. As an example, the 82380 is capable of reversing a string or data by having a Requester address increment and the Target address decrement in a memory-to-memory transfer.

3.3.2 BUFFER TRANSFER PROCESSES

The 82380 DMA Controller allows three programmable Buffer Transfer Processes. These processes define the logical way in which a buffer of data is accessed by the DMA.

The three Buffer Transfer Processes include the Single Buffer Process, the Buffer Auto-Initialize Process, and the Buffer Chaining Process. These processes require special programming considerations. See the DMA Programming section for more details on setting up the Buffer Transfer Processes.

Single Buffer Process

The Single Buffer Process allows the DMA channel to transfer only one buffer of data. When the buffer has been completely transferred (Current Byte Count decremented past zero or EOP# input active), the DMA process ends and the channel becomes idle. In order for that channel to be used again, it must be reprogrammed.

The single Buffer Process is usually used when the amount of data to be transferred is known exactly, and it is also known that there is not likely to be any data to follow before the operating system can reprogram the channel.

Buffer Auto-Initialize Process

The Buffer Auto-Initialize Process allows multiple groups of data to be transferred to or from a single buffer. This process does not require reprogramming. The Current Registers are automatically reprogrammed from the Base Registers when the current process is terminated, either by an expired Byte Count or by an external EOP# signal. The data transferred will always be between the same Target and Requester.

The auto-initialization/process-execution cycle is repeated, with a HOLD/HLDA re-arbitration, until the channel is either disabled or re-programmed.

Buffer Chaining Process

The Buffer Chaining Process is useful for transferring large quantities of data into non-contiguous buffer areas. In this process, a single channel is used to process data from several buffers, while having to program the channel only once. Each new buffer is programmed in a pipelined operation that provides the new buffer information while the old buffer is being processed. The chain is created by loading new buffer information while the 82380 DMA Controller is processing the Current Buffer. When the Current Buffer expires, the 82380 DMA Controller automatically restarts the channel using the new buffer information.

Loading the new buffer information is done by an interrupt routine which is requested by the 82380. Interrupt Request 1 (IRQ1) is tied internally to the 82380 DMA Controller for this purpose. IRQ1 is generated by the 82380 when the new buffer information is loaded into the channel's Current Registers, leaving the Base Registers 'empty'. The interrupt service routine loads new buffer information into the Base Registers. The host processor is required to load the information for another buffer before the current Byte Count expires. The process repeats until the host programs the channel back to single buffer operation, or until the channel runs out of buffers.

The channel runs out of buffers when the Current Buffer expires and the Base Registers have not yet been loaded with new buffer information. When this occurs, the channel must be reprogrammed.

If an external EOP# is encountered while executing a Buffer Chaining Process, the current buffer is considered expired and the new buffer information is loaded into the Current Registers. If the Base Registers are 'empty', the chain is terminated.

The channel uses the Base Target Address Register as an indicator of whether or not the Base Registers are full. When the most significant byte of the Base Target Register is loaded, the channel considers all of the Base Registers loaded, and removes the interrupt request. This requires that the other Base Registers (Base Requester Address, Last Byte Count) must be loaded before the Base Target Address Register. The reason for implementing the reloading process this way is that, for most applications, the Byte Count and the Requester will not change from one buffer to the next, and therefore do not need to be reprogrammed. The details of programming the channel for the Buffer Chaining Process can be found in the section of DMA programming.

3.3.3 DATA TRANSFER MODES

Three Data Transfer modes are available in the 82380 DMA Controller. They are the Single Transfer, Block Transfer, and Demand Transfer Modes. These transfer modes can be used in conjunction with any one of three Buffer Transfer modes: Single Buffer, Auto-Initialized Buffer, and Buffer Chaining. Any Data Transfer Modes can be used under any of the Buffer Transfer Modes. These modes are independently available for all DMA channels.

Different devices being serviced by the DMA Controller require different handshaking sequences for data transfers to take place. Three handshaking modes are available on the 82380, giving the designer the opportunity to use the DMA Controller as efficiently as possible. The speed at which data can be presented or read by a device can affect the way a DMA controller uses the host's bus, thereby affecting not only data throughput during the DMA process, but also affecting the host's performance by limiting its access to the bus.

Single Transfer Mode

In the Single Transfer Mode, one data transfer to or from the Requester is performed by the DMA Controller at a time. The DREQn input is arbitrated and the HOLD/HLDA sequence is executed for each transfer. Transfers continue in this manner until the Byte Count expires, or until EOP# is sampled active. If the DREQn input is held active continuously, the entire DREQ-HOLD-HLDA-DACK sequence is repeated over and over until the programmed number of bytes has been transferred. Bus control is released to the host between each transfer. Figure 3-4 shows the logical flow of events which make up a buffer transfer using the Single Transfer Mode. Refer to section 3.4 for an explanation of the bus control arbitration procedure.

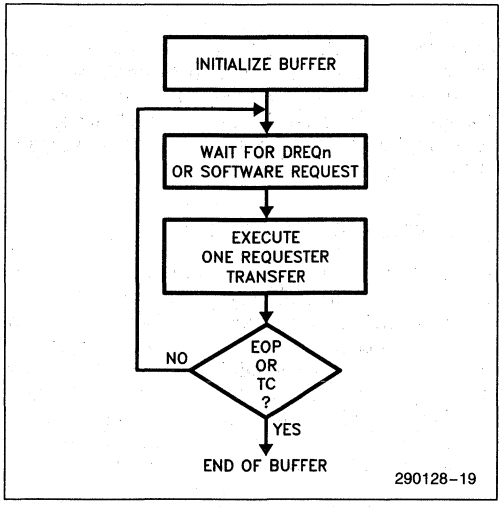


Figure 3-4. Buffer Transfer in Single Transfer Mode

The Single Transfer Mode is used for devices which require complete handshake cycles with each data access. Data is transferred to or from the Requester only when the Requester is ready to perform the transfer. Each transfer requires the entire DREQ-HOLD-HLDA-DACK handshake cycle. Figure 3-5 shows the timing of the Single Transfer Mode cycles.

Block Transfer Mode

In the Block Transfer Mode, the DMA process is initiated by a DMA request and continues until the Byte count expires, or until EOP# is activated by the Requester. The DREQn signal need only be held active until the first Requester access. Only a refresh cycle will interrupt the block transfer process.

Figure 3-6 illustrates the operation of the DMA during the Block Transfer Mode. Figure 3-7 shows the timing of the handshake signals during Block Mode Transfers.

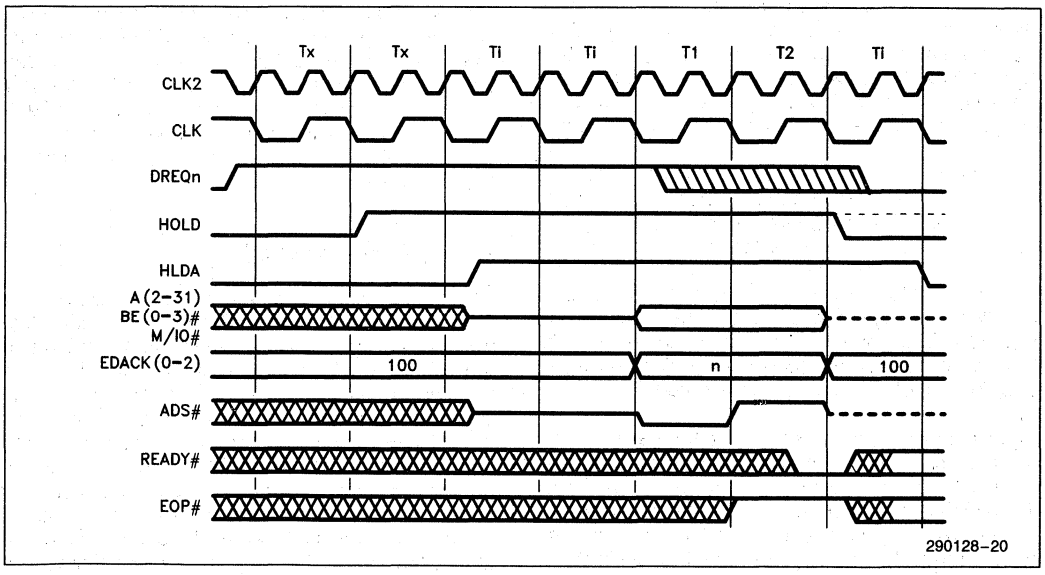


Figure 3-5. DMA Single Transfer Mode

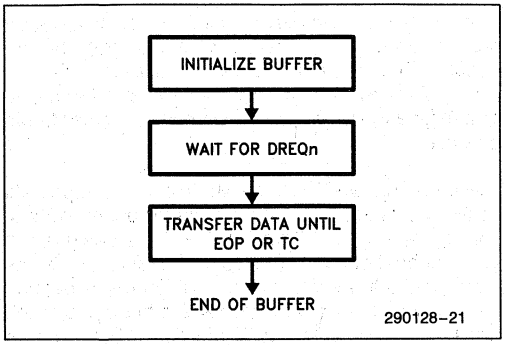


Figure 3-6. Buffer Transfer in Block Transfer Mode

Demand Transfer Mode

The Demand Transfer Mode provides the most flexible handshaking procedures during the DMA process. A Demand Transfer is initiated by a DMA request. The process continues until the Byte Count expires, or an external EOP# is encountered. If the device being serviced (Requester) desires, it can interrupt the DMA process by de-activating the DREQn line. Action is taken on the condition of DREQn during Requester accesses only. The access during which DREQn is sampled inactive is the last Requester access which will be performed during the current transfer. Figure 3-8 shows the flow of events during the transfer of a buffer in the Demand Mode.

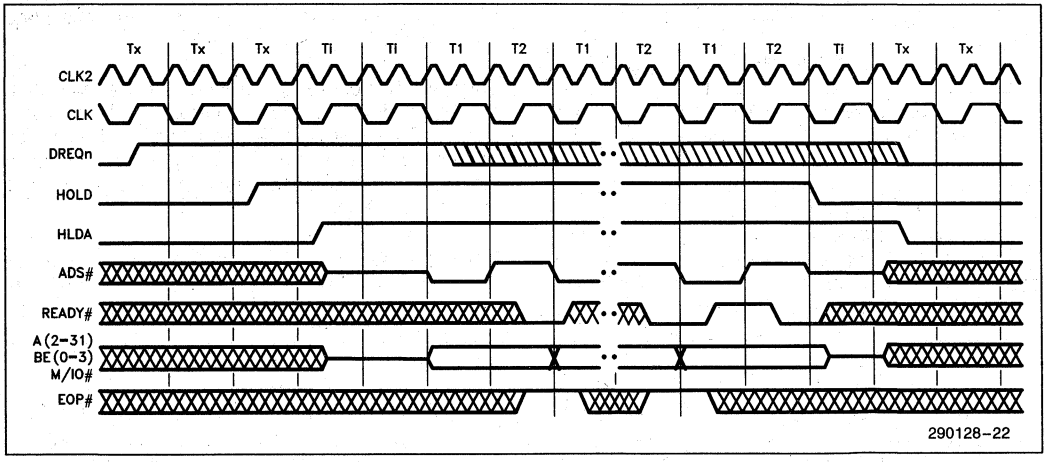


Figure 3-7. Block Mode Transfers

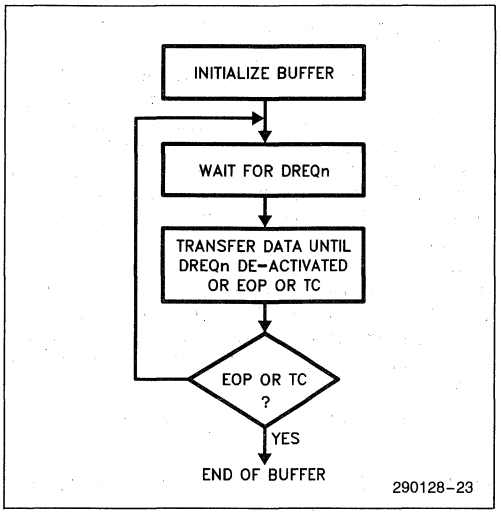


Figure 3-8. Buffer Transfer in Demand Transfer Mode

When the DREQn line goes inactive, the DMA controller will complete the current transfer, including any necessary accesses to the Target, and relinquish control of the bus to the host. The current process information is saved (byte count, Requester and Target addresses, and Temporary Register).

The Requester can restart the transfer process by reasserting DREQn. The 82380 will arbitrate the request with other pending requests and begin the process where it left off. Figure 3-9 shows the timing of handshake signals during Demand Transfer Mode operation.

Using the Demand Transfer Mode allows peripherals to access memory in small, irregular bursts without wasting bus control time. The 82380 is designed to give the best possible bus control latency in the Demand Transfer Mode. Bus control latency is defined here as the time from the last active bus cycle of the previous bus master to the first active bus cycle of the new bus master. The 82380 DMA Controller will perform its first bus access cycle two bus states after HLDA goes active. In the typical configuration, bus control is returned to the host one bus state after the DREQn goes inactive.

There are two cases where there may be more than one bus state of bus control latency at the end of a transfer. The first is at the end of an Auto-Initialize process, and the second is at the end of a process where the source is the Requester and Two-Cycle transfers are used.

When a Buffer Auto-Initialize Process is complete, the 82380 requires seven bus states to reload the

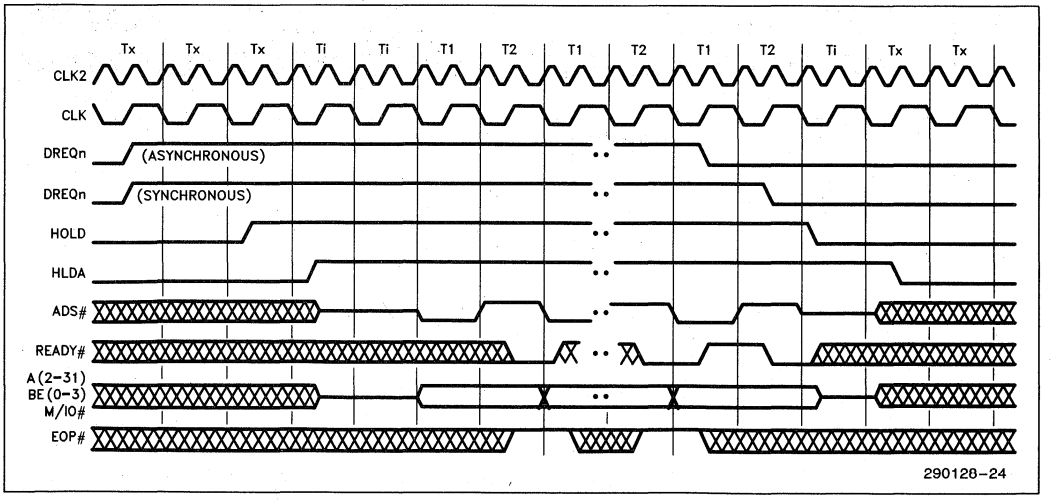


Figure 3-9. Demand Mode Transfers

Current Registers from the Base Registers of the Auto-Initialized channel. The reloading is done while the 82380 is still the bus master so that it is prepared to service the channel immediately after relinquishing the bus, if necessary.

In the case where the Requester is the source, and Two-Cycle transfers are being used, there are two extra idle states at the end of the transfer process. This occurs due to housekeeping in the DMA's internal pipeline. These two idle states are present only after the very last Requester access, before the DMA Controller de-activates the HOLD signal.

3.3.4 CHANNEL PRIORITY ARBITRATION

DMA channel priority can be programmed into one of two arbitration methods: Fixed or Rotating. The four lower DMA channels and the four upper DMA channels operate as if they were two separate DMA controllers operating in cascade. The lower group of four channels (0–3) is always prioritized between channels 7 and 4 of the upper group of channels (4–7). Figure 3-10 shows a pictorial representation of the priority grouping.

The priority can thus be set up as rotating for one group of channels and fixed for the other, or any other combination. While in Fixed Priority, the programmer can also specify which channel has the lowest priority.

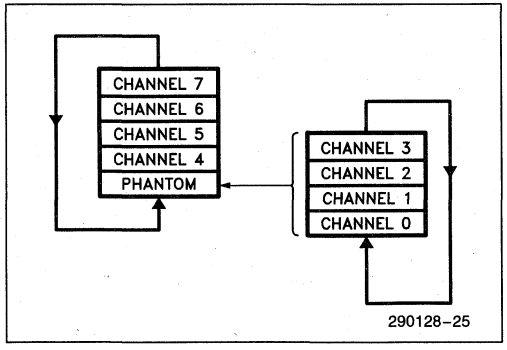


Figure 3-10. DMA Priority Grouping

The 82380 DMA Controller defaults to Fixed Priority. Channel 0 has the highest priority, then 1, 2, 3, 4, 5, 6, 7. Channel 7 has the lowest priority. Any time the DMA Controller arbitrates DMA requests, the requesting channel with the highest priority will be serviced next.

Fixed Priority can be entered into at any time by a software command. The priority levels in effect after the mode switch are determined by the current setting of the Programmable Priority.

Programmable Priority is available for fixing the priority of the DMA channels within a group to levels other than the default. Through a software command, the channel to have the lowest priority in a group can be specified. Each of the two groups of four channels can have the priority fixed in this way. The other channels in the group will follow the natural Fixed Priority sequence. This mode affects only the priority levels while operating with Fixed Priority.

For example, if channel 2 is programmed to have the lowest priority in its group, channel 3 has the highest priority. In descending order, the other channels would have the following priority: (3, 0, 1, 2), 4, 5, 6, 7 (channel 2 lowest, channel 3 highest). If the upper group were programmed to have channel 5 as the lowest priority channel, the priority would be (again, highest to lowest): 6, 7, (3, 0, 1, 2), 4, 5. Figure 3-11 shows this example pictorially. The lower group is always prioritized as a fifth channel of the upper group (between channels 4 and 7).

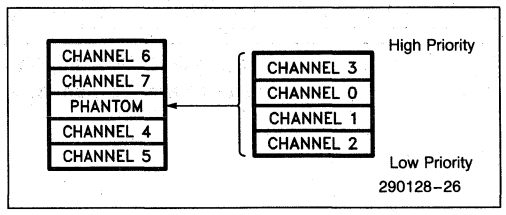


Figure 3-11. Example of Programmed Priority

The DMA Controller will only accept Programmable Priority commands while the addressed group is operating in Fixed Priority. Switching from Fixed to Rotating Priority preserves the current priority levels. Switching from Rotating to Fixed Priority returns the priority levels to those which were last programmed by use of Programmable Priority.

Rotating Priority allows the devices using DMA to share the system bus more evenly. An individual channel does not retain highest priority after being serviced, priority is passed to the next highest priority channel in the group. The channel which was most recently serviced inherits the lowest priority. This rotation occurs each time a channel is serviced. Figure 3-12 shows the sequence of events as priority is passed between channels. Note that the lower group rotates within the upper group, and that servicing a channel within the lower group causes rotation within the group as well as rotation of the upper group.

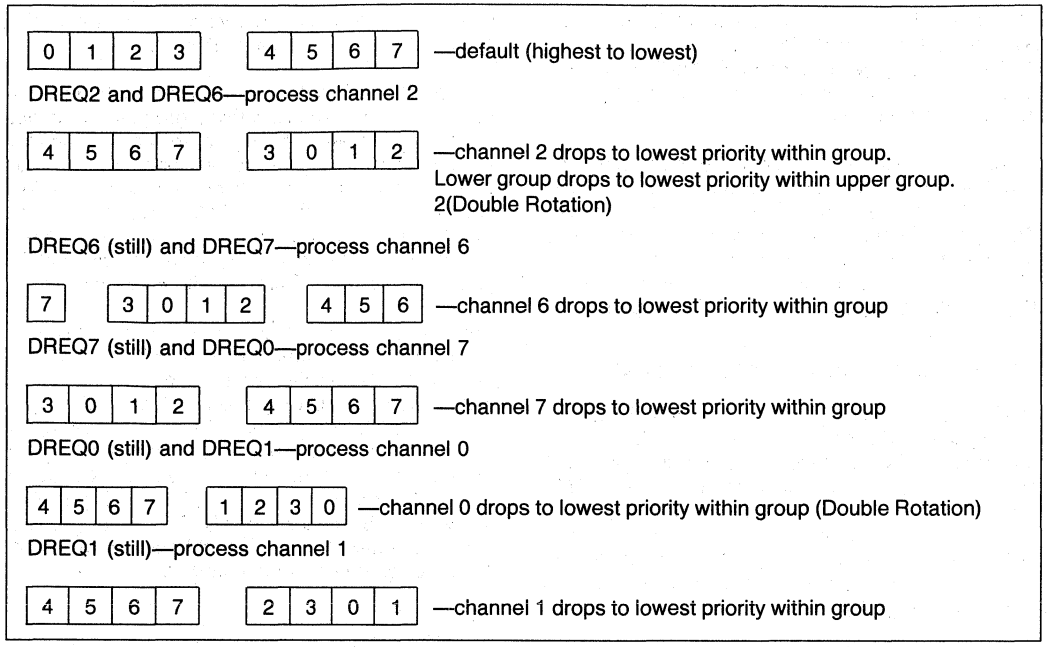


Figure 3-12. Rotating Channel Priority. Lower and Upper groups are programmed for the Rotating Priority Mode.

3.3.5 COMBINING PRIORITY MODES

Since the DMA Controller operates as two four-channel controllers in cascade, the overall priority scheme of all eight channels can take on a variety of forms. There are four possible combinations of priority modes between the two groups of channels: Fixed Priority only (default), Fixed Priority upper group/Rotating Priority lower group, Rotating Priority upper group/Fixed Priority lower group, and Rotating Priority only. Figure 3-13 illustrates the operation of the two combined priority methods.

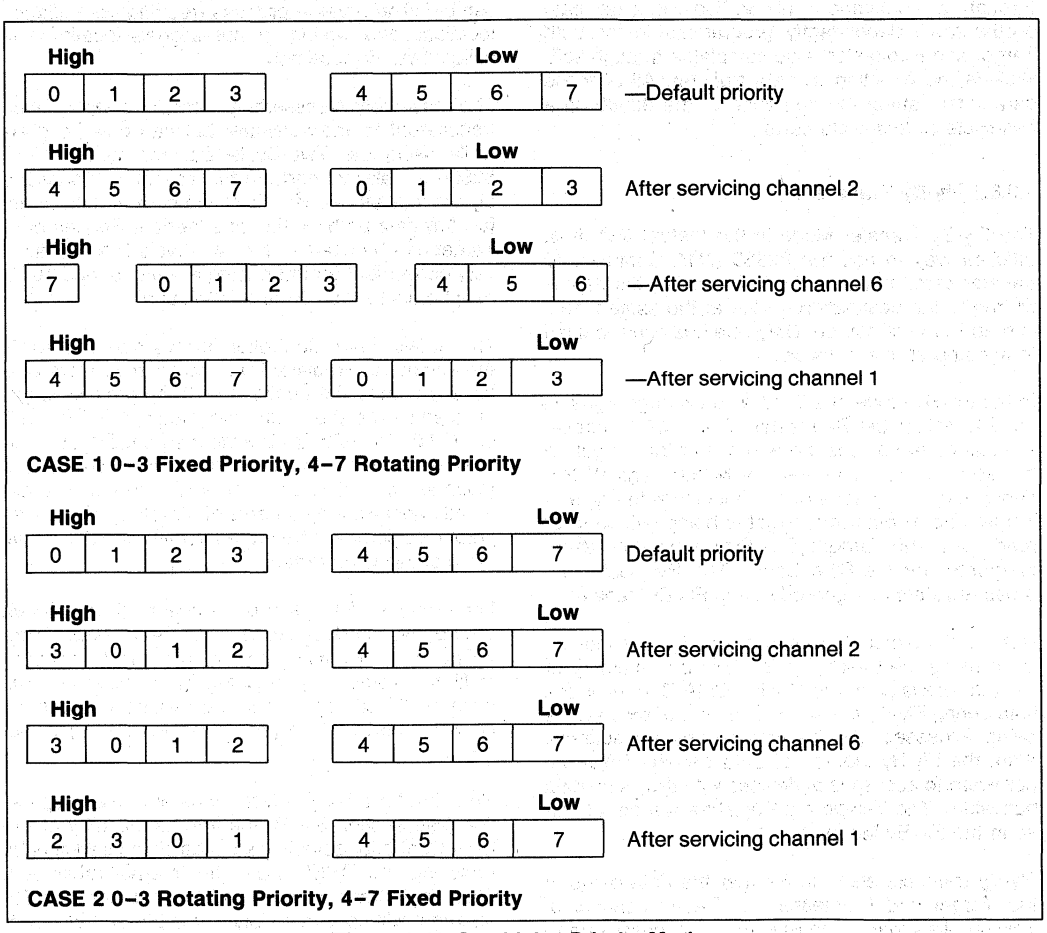


Figure 3-13. Combining Priority Modes

5

3.3.6 BUS OPERATION

Data may be transferred by the DMA Controller using two different bus cycle operations: Fly-By (one-cycle) and Two-Cycle. These bus handshake methods are selectable independently for each channel through a command register. Device data path widths are independently programmable for both Target and Requester. Also selectable through software is the direction of data transfer. All of these parameters affect the operation of the 82380 on a bus-cycle by bus-cycle basis.

3.3.6.1 Fly-By Transfers

The Fly-By Transfer Mode is the fastest and most efficient way to use the 82380 DMA Controller to transfer data. In this method of transfer, the data is written to the destination device at the same time it is read from the source. Only one bus cycle is used to accomplish the transfer.

In the Fly-By Mode, the DMA acknowledge signal is used to select the Requester. The DMA Controller simultaneously places the address of the Target on the address bus. The state of M/IO# and W/R# during the Fly-By transfer cycle indicate the type of Target and whether the target is being written to or read from. The Target's Bus Size is used as an incrementer for the Byte Count. The Requester address registers are ignored during Fly-By transfers.

Note that memory-to-memory transfers cannot be done using the Fly-By Mode. Only one memory or I/O address is generated by the DMA Controller at a time during Fly-By transfers. Only one of the devices being accessed can be selected by an address. Also, the Fly-By method of data transfer limits the hardware to accesses of devices with the same data bus width. The Temporary Registers are not affected in the Fly-By Mode.

Fly-By transfers also require that the data paths of the Target and Requester be directly connected. This requires that successive Fly-By accesses be to doubleword boundaries, or that the Requester be capable of switching its connections to the data bus.

3.3.6.2 Two-Cycle Transfers

Two-Cycle transfers can also be performed by the 82380 DMA Controller. These transfers require at least two bus cycles to execute. The data being transferred is read into the DMA Controller's Temporary Register during the first bus cycle(s). The second bus cycle is used to write the data from the Temporary Register to the destination.

If the addresses of the data being transferred are not word or doubleword aligned, the 82380 will recognize the situation and read and write the data in groups of bytes, placing them always at the proper destination. This process of collecting the desired bytes and putting them together is called 'byte assembly'. The reverse process (reading from aligned locations and writing to non-aligned locations) is called 'byte disassembly'.

The assembly/disassembly process takes place transparent to the software, but can only be done while using the Two-Cycle transfer method. The 82380 will always perform the assembly/disassembly process as necessary for the current data transfer. Any data path widths for either the Requester or Target can be used in the Two-Cycle Mode. This is very convenient for interfacing existing 8- and 16-bit peripherals to the 80386's 32-bit bus.

The 82380 DMA Controller always attempts to fill the Temporary Register from the source before writing any data to the destination. If the process is terminated before the Temporary Register is filled (TC or EOP#), the 82380 will write the partial data to the destination. If a process is temporarily suspended (such as when DREQn is de-activated during a demand transfer), the contents of a partially filled Temporary Register will be stored within the 82380 until the process is restarted.

For example, if the source is specified as an 8-bit device and the destination as a 32-bit device, there will be four reads as necessary from the 8-bit source to fill the Temporary Register. Then the 82380 will write the 32-bit contents to the destination. This cycle will repeat until the process is terminated or suspended.

Note that for a Single-Cycle transfer mode of operation (see section 3.3.3), the internal circuitry of the DMA Controller actually executes single transfers by removing the DREQ from the internal arbitration. Thus single transfers from an 8-bit requester to a 32-bit target will consist of four complete and independent 8-bit requester cycles, between which bus control is released and re-requested. Finally, the 32-bit data will be transferred to the target device from the temporary register before the fifth requester cycle.

With Two-Cycle transfers, the devices that the 82380 accesses can reside at any address within I/O or memory space. The device must be able to decode the byte-enables (BEn#). Also, if the device cannot accept data in byte quantities, the programmer must take care not to allow the DMA Controller to access the device on any address other than the device boundary.

3.3.6.3 Data Path Width and Data Transfer Rate Considerations

The number of bus cycles used to transfer a single 'word' of data is affected by whether the Two-Cycle or the Fly-By (Single-Cycle) transfer method is used.

The number of bus cycles used to transfer data directly affects the data transfer rate. Inefficient use of bus cycles will decrease the effective data transfer rate that can be obtained. Generally, the data transfer rate is halved by using Two-Cycle transfers instead of Fly-By transfers.

The choice of data path widths of both Target and Requester affects the data transfer rate also. During each bus cycle, the largest pieces of data possible should be transferred.

The data path width of the devices to be accessed must be programmed into the DMA controller. The 82380 defaults after reset to 8-bit-to-8-bit data transfers, but the Target and Requester can have different data path widths, independent of each other and independent of the other channels. Since this is a software programmable function, more discussion of the uses of this feature are found in the section on programming.

3.3.6.4 Read, Write, and Verify Cycles

Three different bus cycle types may be used in a data transfer. They are the Read, Write, and Verify cycles. These cycle types dictate the way in which the 82380 operates on the data to be transferred.

A Read Cycle transfers data from the Target to the Requester. A Write Cycle transfers data from the Requester to the target. In a Fly-By transfer, the address and bus status signals indicate the access (read or write) to the Target; the access to the Requester is assumed to be the opposite.

The Verify Cycle is used to perform a data read only. No write access is indicated or assumed in a Verify Cycle. The Verify Cycle is useful for validating block fill operations. An external comparator must be provided to do any comparisons on the data read.

3.4 Bus Arbitration and Handshaking

Figure 3-14 shows the flow of events in the DMA request arbitration process. The arbitration sequence starts when the Requester asserts a DREQn (or DMA service is requested by software). Figure 3-15 shows the timing of the sequence of events following a DMA request. This sequence is executed for each channel that is activated. The DREQn signal can be replaced by a software DMA channel request with no change in the sequence.

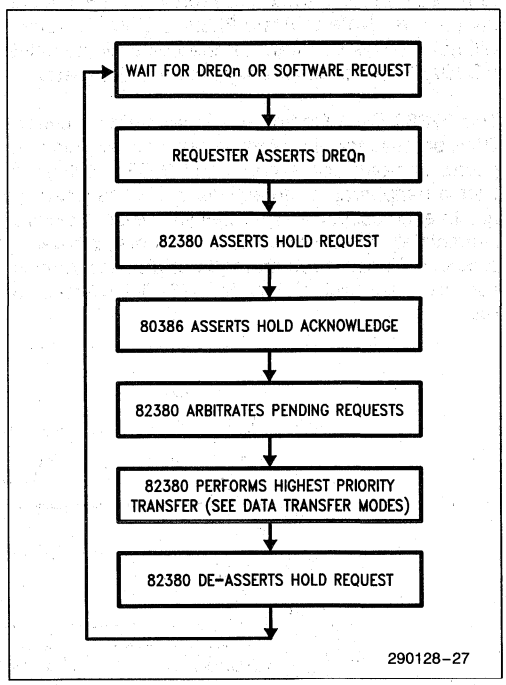


Figure 3-14. Bus Arbitration and DMA Sequence

After the Requester asserts the service request, the 82380 will request control of the bus via the HOLD signal. The 82380 will always assert the HOLD signal one bus state after the service request is asserted. The 80386 responds by asserting the HLDA signal, thus releasing control of the bus to the 82380 DMA Controller.

Priority of pending DMA service requests is arbitrated during the first state after HLDA is asserted by the 80386. The next state will be the beginning of the first transfer access of the highest priority process.

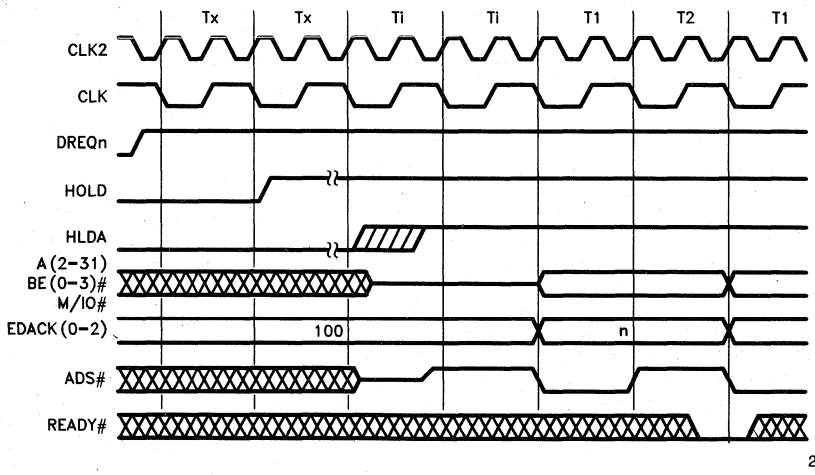
When the 82380 DMA Controller is finished with its current bus activity, it returns control of the bus to the host processor. This is done by driving the HOLD signal inactive. The 82380 does not drive any address or data bus signals after HOLD goes low. It enters the Slave Mode until another DMA process is requested. The processor acknowledges that it has regained control of the bus by forcing the HLDA signal inactive. Note that the 82380's DMA Controller will not re-request control of the bus until the entire HOLD/HLDA handshake sequence is complete.

The 82380 DMA Controller will terminate a current DMA process for one of three reasons: expired byte count, end-of-process command (EOP# activated) from a peripheral, or de-activated DMA request signal. In each case, the controller will de-assert HOLD immediately after completing the data transfer in progress. These three methods of process termination are illustrated in Figures 3-16, 3-19, and 3-18, respectively.

An expired byte count indicates that the current process is complete as programmed and the channel has no further transfers to process. The channel must be restarted according to the currently programmed Buffer Transfer Mode, or reprogrammed completely, including a new Buffer Transfer Mode.

If the peripheral activates the EOP# signal, it is indicating that it will not accept or deliver any more data for the current buffer. The 82380 DMA Controller considers this as a completion of the channel's current process and interprets the condition the same way as if the byte count expired.

The action taken by the 82380 DMA Controller in response to a de-activated DREQn signal depends on the Data Transfer Mode of the channel. In the Demand Mode, data transfers will take place as long as the DREQn is active and the byte count has not expired. In the Block Mode, the controller will complete the entire block transfer without relinquishing



NOTE:
Channel priority resolution takes place during the bus state before HLDA is asserted, allowing the DMA Controller to respond to HLDA without extra idle bus states.

Figure 3-15. Beginning of a DMA process

the bus, even if DREQn goes inactive before the transfer is complete. In the Single Mode, the controller will execute single data transfers, relinquishing the bus between each transfer, as long as DREQn is active.

Normal termination of a DMA process due to expiration of the byte count (Terminal Count-TC) is shown in Figure 3-16. The condition of DREQn is ignored until after the process is terminated. If the channel is programmed to auto-initialize, HOLD will be held active for an additional seven clock cycles while the auto-initialization takes place.

Table 3-3 shows the DMA channel activity due to EOP# or Byte Count expiring (Terminal Count).

Buffer Process:	Single or Chaining-Base Empty		Auto-Initialize		Chaining-Base Loaded	
Event						
Terminal Count	True	X	True	X	True	X
EOP# Input	X	0	X	0	X	0
Results						
Current Registers	—	—	Load	Load	Load	Load
Channel Mask	Set	Set	—	—	—	—
EOP# Output	0	X	0	X	1	X
Terminal Count Status	Set	Set	Set	Set	—	—
Software Request	CLR	CLR	CLR	CLR	—	—

Table 3-3. DMA Channel Activity Due to Terminal Count or External EOP#

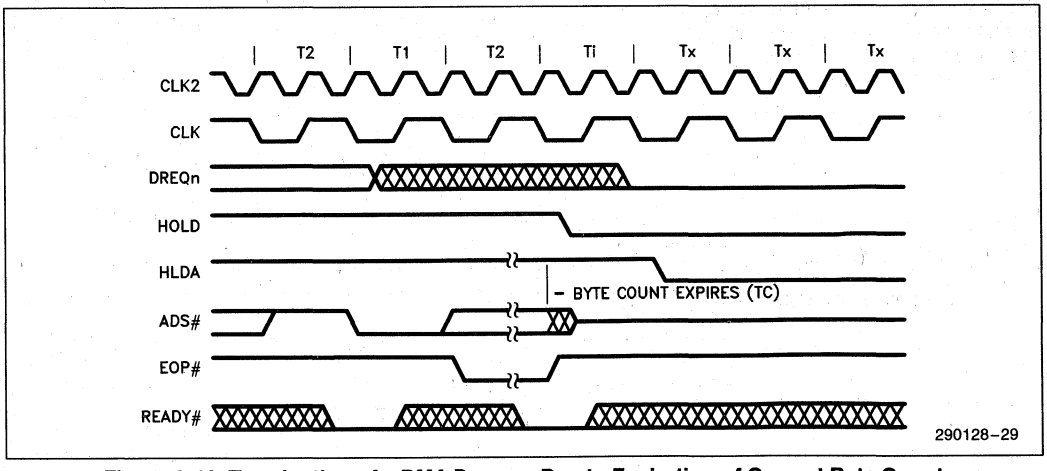


Figure 3-16. Termination of a DMA Process Due to Expiration of Current Byte Count

5

The 82380 always relinquishes control of the bus between channel services. This allows the hardware designer the flexibility to externally arbitrate bus hold requests, if desired. If another DMA request is pending when a higher priority channel service is completed, the 82380 will relinquish the bus until the hold acknowledge is inactive. One bus state after the HLDA signal goes inactive, the 82380 will assert HOLD again. This is illustrated in Figure 3-17.

3.4.1 SYNCHRONOUS AND ASYNCHRONOUS SAMPLING OF DREQn AND EOP#

As an indicator that a DMA service is to be started, DREQn is always sampled asynchronously. It is sampled at the beginning of a bus state and acted upon at the end of the state. Figure 3-15 illustrates the start of a DMA process due to a DREQn input.

The DREQn and EOP# inputs can be programmed to be sampled either synchronously or asynchronously to signal the end of a transfer.

The synchronous mode affords the Requester one bus state of extra time to react to an access. This means the Requester can terminate a process on the current access, without losing any data. The asynchronous mode requires that the input signal be presented prior to the beginning of the last state of the Requester access.

The timing relationships of the DREQn and EOP# signals to the termination of a DMA transfer are shown in Figures 3-18 and 3-19. Figure 3-18 shows the termination of a DMA transfer due to inactive DREQn. Figure 3-19 shows the termination of a DMA process due to an active EOP# input.

In the Synchronous Mode, DREQn and EOP# are sampled at the end of the last state of every Requester data transfer cycle. If EOP# is active or DREQn is inactive at this time, the 82380 recognizes this access to the Requester as the last transfer. At this point, the 82380 completes the transfer in progress, if necessary, and returns bus control to the host.

In the asynchronous mode, the inputs are sampled at the beginning of every state of a Requester access. The 82380 waits until the end of the state to act on the input.

DREQn and EOP# are sampled at the latest possible time when the 82380 can determine if another transfer is required. In the Synchronous Mode, DREQn and EOP# are sampled on the trailing edge of the last bus state before another data access cycle begins. The Asynchronous Mode requires that the signals be valid one clock cycle earlier.

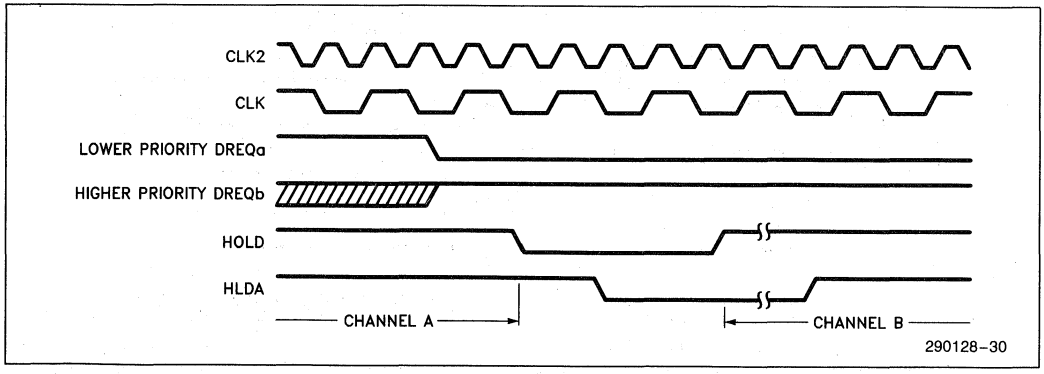


Figure 3-17. Switching between Active DMA Channels

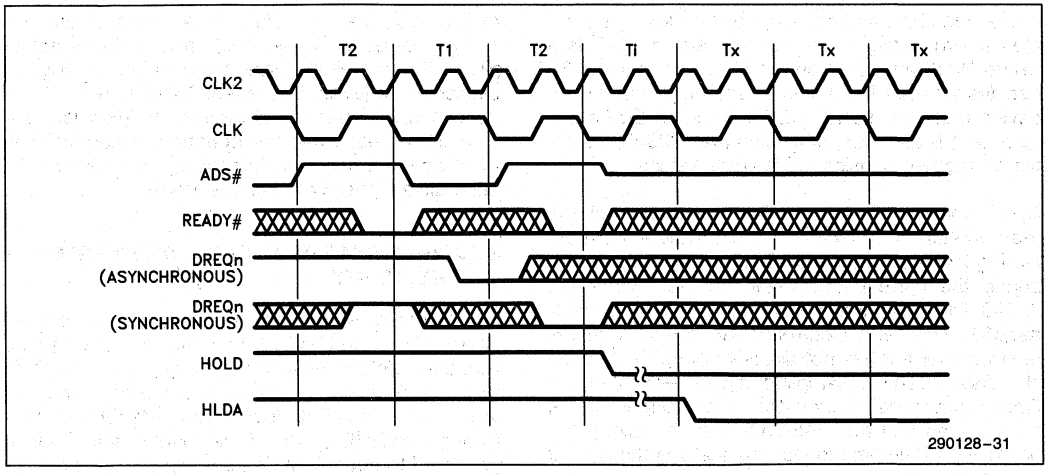


Figure 3-18. Termination of a DMA Process Due to De-Asserting DREQn

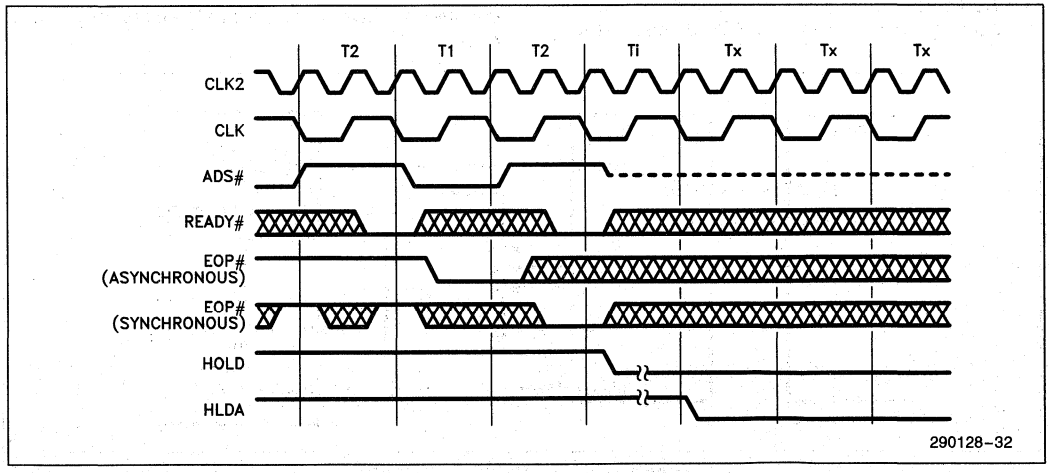


Figure 3-19. Termination of a DMA Process Due to an External EOP#

While in the Pipeline Mode, if the NA# signal is sampled active during a transfer, the end of the state where NA# was sampled active is when the 82380 decides whether to commit to another transfer. The device must de-assert DREQn or assert EOP# before NA# is asserted, otherwise the 82380 will commit to another, possibly undesired, transfer.

Synchronous DREQn and EOP# sampling allows the peripheral to prevent the next transfer from occurring by de-activating DREQn or asserting EOP# during the current Requester access, before the 82380 DMA Controller commits itself to another transfer. The DMA Controller will not perform the next transfer if it has not already begun the bus cycle. Asynchronous sampling allows less stringent timing requirements than the Synchronous Mode, but requires that the DREQn signal be valid at the beginning of the next to last bus state of the current Requester access.

Using the Asynchronous Mode with zero wait states can be very difficult. Since the addresses and control signals are driven by the 82380 near half-way through the first bus state of a transfer, and the Asynchronous Mode requires that DREQn be active before the end of the state, the peripheral being accessed is required to present DREQn only a few nanoseconds after the control information is available. This means that the peripheral's control logic must be extremely fast (practically non-causal). An alternative is the Synchronous Mode.

3.4.2 ARBITRATION OF CASCADED MASTER REQUESTS

The Cascade Mode allows another DMA-type device to share the bus by arbitrating its bus accesses with the 82380's. Seven of the eight DMA channels (0–3 and 5–7) can be connected to a cascaded device. The cascaded device requests bus control through the DREQn line of the channel which is programmed to operate in Cascade Mode. Bus hold acknowledge is signaled to the cascaded device through the EDACK lines. When the EDACK lines are active with the code for the requested cascade channel, the bus is available to the cascaded master device.

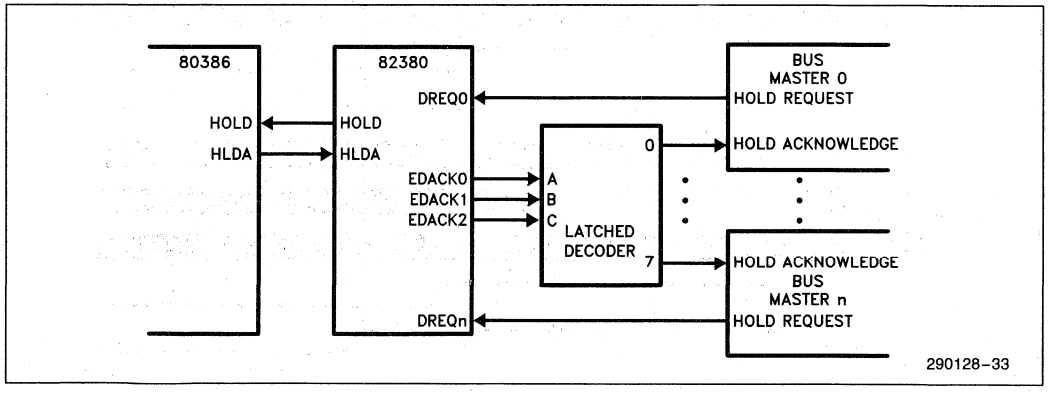


Figure 3-20. Cascaded Bus Master

A Cascade cycle begins the same way a regular DMA cycle begins. The requesting bus master asserts the DREQn line on the 82380. This bus control request arbitrated as any other DMA request would be. If any channel receives a DMA request, the 82380 requests control of the bus. When the host acknowledges that it has released bus control, the 82380 acknowledges to the requesting master that it may access the bus. The 82380 enters an idle state until the new master relinquishes control.

A cascade cycle will be terminated by one of two events: DREQn going inactive, or HLDA going inactive. The normal way to terminate the cascade cycle is for the cascaded master to drop the DREQn signal. Figure 3-21 shows the two cascade cycle termination sequences.

The Refresh Controller may interrupt the cascaded master to perform a refresh cycle. If this occurs, the 82380 DMA Controller will de-assert the EDACK signal (hold acknowledge to cascaded master) and wait for the cascaded master to remove its hold request. When the 82380 regains bus control, it will perform the refresh cycle in its normal fashion. After the refresh cycle has been completed, and if the cascaded device has re-asserted its request, the 82380 will return control to the cascaded master which was interrupted.

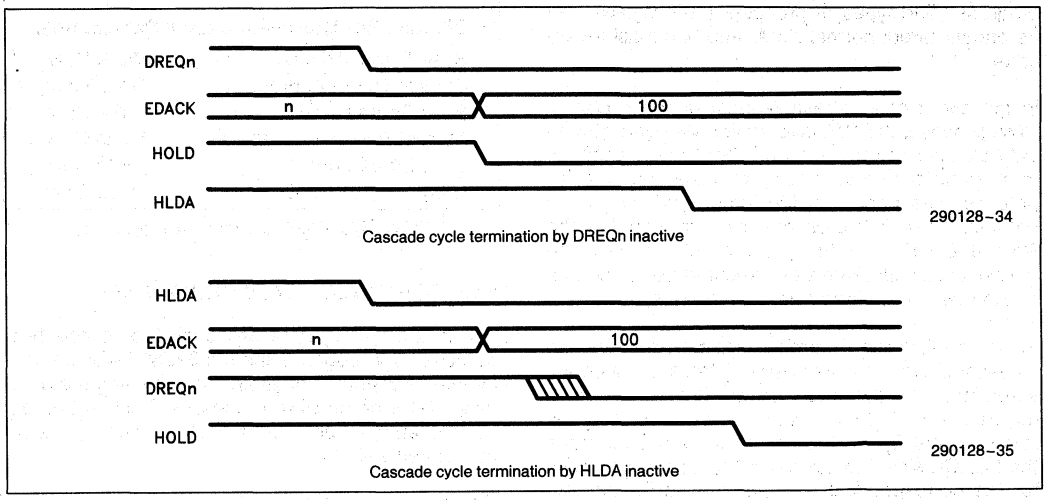


Figure 3-21. Cascade Cycle Termination

The 82380 assumes that it is the only device monitoring the HLDA signal. If the system designer wishes to place other devices on the bus as bus masters, the HLDA from the processor must be intercepted before presenting it to the 82380. Using the Cascade capability of the 82380 DMA Controller offers a much better solution.

3.4.3 ARBITRATION OF REFRESH REQUESTS

The arbitration of refresh requests by the DRAM Refresh Controller is slightly different from normal DMA channel request arbitration. The 82380 DRAM Refresh Controller always has the highest priority of any DMA process. It also can interrupt a process in progress. Two types of processes in progress may be encountered: normal DMA, and bus master cascade.

In the event of a refresh request during a normal DMA process, the DMA Controller will complete the data transfer in progress and then execute the refresh cycle before continuing with the current DMA process. The priority of the interrupted process is not lost. If the data transfer cycle interrupted by the Refresh Controller is the last of a DMA process, the refresh cycle will always be executed before control of the bus is transferred back to the host.

When the Refresh Controller request occurs during a cascade cycle, the Refresh Controller must be assured that the cascaded master device has relinquished control of the bus before it can execute the refresh cycle. To do this, the DMA Controller drops the EDACK signal to the cascaded master and waits for the corresponding DREQn input to go inactive. By dropping the DREQn signal, the cascaded master relinquishes the bus. The Refresh Controller then performs the refresh cycle. Control of the bus is returned to the cascaded master if DREQn returns to an active state before the end of the refresh cycle, otherwise control is passed to the processor and the cascaded master loses its priority.

3.5 DMA Controller Register Overview

The 82380 DMA Controller contains 44 registers which are accessable to the host processor. Twenty-four of these registers contain the device addresses and data counts for the individual DMA channels (three per channel). The remaining registers are control and status registers for initiating and monitoring the operation of the 82380 DMA Controller. Table 3-4 lists the DMA Controller's registers and their accessability.

Register Name	Access
Control/Status Register—One Each Per Group	
Command Register I	Write Only
Command Register II	Write Only
Mode Register I	Write Only
Mode Register II	Write Only
Software Request Register	Read/Write
Mask Set-Reset Register	Write Only
Mask Read-Write Register	Read/Write
Status Register	Read Only
Bus Size Register	Write Only
Chaining Register	Read/Write
Channel Registers—One Each Per Channel	
Base Target Address	Write Only
Current Target Address	Read Only
Base Requester Address	Write Only
Current Requester Address	Read Only
Base Byte Count	Write Only
Current Byte Count	Read Only

Table 3-4. DMA Controller Registers

3.5.1 CONTROL/STATUS REGISTERS

The following registers are available to the host processor for programming the 82380 DMA Controller into its various modes and for checking the operating status of the DMA processes. Each set of four DMA channels has one of each of these registers associated with it.

Command Register I

Enables or disables the DMA channels as a group. Sets the Priority Mode (Fixed or Rotating) of the group. This write-only register is cleared by a hardware reset, defaulting to all channels enabled and Fixed Priority Mode.

Command Register II

Sets the sampling mode of the DREQn and EOP# inputs. Also sets the lowest priority channel for the group in the Fixed Priority Mode. The functions programmed through Command Register II default after a hardware reset to: asynchronous DREQn and EOP#, and channels 3 and 7 lowest priority.

Mode Register I

Mode Register I is identical in function to the Mode register of the 8237A. It programs the following functions for an individually selected channel:

Type of Transfer—read, write, verify
Auto—Initialize—enable or disable
Target Address Count—increment or decrement
Data Transfer Mode—demand, single, block, cascade

Mode Register I functions default to the following after reset: verify transfer, Auto-Initialize disabled, Increment Target address, Demand Mode.

Mode Register II

Programs the following functions for an individually selected channel:

Target Address Hold—enable or disable
Requester Address Count—increment or decrement
Requester Address Hold—enable or disable
Target Device Type—I/O or Memory
Requester Device Type—I/O or Memory
Transfer Cycles—Two-Cycle or Fly-By

Mode Register II functions are defined as follows after a hardware reset: Disable Target Address Hold, Increment Requester Address, Target (and Requester) in memory, Fly-By Transfer Cycles. Note: Requester Device Type ignored in Fly-By Transfers.

Software Request Register

The DMA Controller can respond to service requests which are initiated by software. Each channel has an internal request status bit associated with it. The host processor can write to this register to set or reset the request bit of a selected channel.

The status of the group's software DMA service requests can be read from this register as well. Each request bit is cleared upon Terminal Count or external EOP#.

The software DMA requests are non-maskable and subject to priority arbitration with all other software and hardware requests. The entire register is cleared by a hardware reset.

Mask Registers

Each channel has associated with it a mask bit which can be set/reset to disable/enable that channel. Two methods are available for setting and clearing the mask bits. The Mask Set/Reset Register is a write-only register which allows the host to select an individual channel and either set or reset the mask bit for that channel only. The Mask Read/Write Register is available for reading the mask bit status and for writing mask bits in groups of four.

The mask bits of a group may be cleared in one step by executing the Clear Mask Command. See the DMA Programming section for details. A hardware reset sets all of the channel mask bits, disabling all channels.

Status Register

The Status register is a read-only register which contains the Terminal Count (TC) and Service Request status for a group. Four bits indicate the TC status and four bits indicate the hardware request status for the four channels in the group. The TC bits are set when the Byte Count expires, or when an external EOP# is asserted. These bits are cleared by reading from the Status Register. The Service Request bit for a channel indicates when there is a hardware DMA request (DREQn) asserted for that channel. When the request has been removed, the bit is cleared.

Bus Size Register

This write-only register is used to define the bus size of the Target and Requester of a selected channel. The bus sizes programmed will be used to dictate the sizes of the data paths accessed when the DMA channel is active. The values programmed into this register affect the operation of the Temporary Register. Any byte-assembly required to make the transfers using the specified data path widths will be done in the Temporary Register. The Bus Size register of the Target is used as an increment/decrement value for the Byte Counter and Target Address when in the Fly-By Mode. Upon reset, all channels default to 8-bit Targets and 8-bit Requesters.

Chaining Register

As a command or write register, the Chaining register is used to enable or disable the Chaining Mode for a selected channel. Chaining can either be disabled or enabled for an individual channel, independently of the Chaining Mode status of other channels. After a hardware reset, all channels default to Chaining disabled.

When read by the host, the Chaining Register provides the status of the Chaining Interrupt of each of the channels. These interrupt status bits are cleared when the new buffer information has been loaded.

3.5.2 CHANNEL REGISTERS

Each channel has three individually programmable registers necessary for the DMA process; they are the Base Byte Count, Base Target Address, and Base Requester Address registers. The 24-bit Base

Byte Count register contains the number of bytes to be transferred by the channel. The 32-bit Base Target Address Register contains the beginning address (memory or I/O) of the Target device. The 32-bit Base Requester Address register contains the base address (memory or I/O) of the device which is to request DMA service.

Three more registers for each DMA channel exist within the DMA Controller which are directly related to the registers mentioned above. These registers contain the current status of the DMA process. They are the Current Byte Count register, the Current Target Address, and the Current Requester Address. It is these registers which are manipulated (incremented, decremented, or held constant) by the 82380 DMA Controller during the DMA process. The Current registers are loaded from the Base registers.

The Base registers are loaded when the host processor writes to the respective channel register addresses. Depending on the mode in which the channel is operating, the Current registers are typically loaded in the same operation. Reading from the channel register addresses yields the contents of the corresponding Current register.

To maintain compatibility with software which accesses an 8237A, a Byte Pointer Flip-Flop is used to control access to the upper and lower bytes of some words of the Channel Registers. These words are accessed as byte pairs at single port addresses. The Byte Pointer Flip-Flop acts as a one-bit pointer which is toggled each time a qualifying Channel Register byte is accessed. It always points to the next logical byte to be accessed of a pair of bytes.

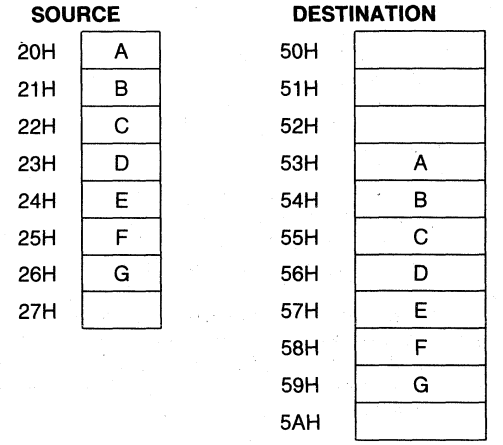
The Channel registers are arranged as pairs of words, each pair with its own port address. Addressing the port with the Byte Pointer Flip-Flop reset accesses the least significant byte of the pair. The most significant byte is accessed when the Byte Pointer is set.

For compatibility with existing 8237A designs, there is one exception to the above statements about the Byte Pointer Flip-Flop. The third byte (bits 16–23) of the Target Address is accessed through its own port address. The Byte Pointer Flip-Flop is not affected by any accesses to this byte.

The upper eight bits of the Byte Count Register are cleared when the least significant byte of the register is loaded. This provides compatibility with software which accesses an 8237A. The 8237A has 16-bit Byte Count Registers.

3.5.3 TEMPORARY REGISTERS

Each channel has a 32-bit Temporary Register used for temporary data storage during two-cycle DMA transfers. It is this register in which any necessary byte assembly and disassembly of non-aligned data is performed. Figure 3-22 shows how a block of data will be moved between memory locations with different boundaries. Note that the order of the data does not change.



Target = source = 00000020H
Requester = destination = 00000053H
Byte Count = 000006H

Figure 3-22. Transfer of Data between Memory Locations with Different Boundaries. This will be the result, independent of data path width.

If the destination is the Requester and an early process termination has been indicated by the EOP# signal or DREQn inactive in the Demand Mode, the Temporary Register is not affected. If data remains in the Temporary Register due to differences in data path widths of the Target and Requester, it will not be transferred or otherwise lost, but will be stored for later transfer.

If the destination is the Target and the EOP# signal is sensed active during the Requester access of a transfer, the DMA Controller will complete the transfer by sending to the Target whatever information is in the Temporary Register at the time of process termination. This implies that the Target could be accessed with partial data. For this reason it is advisable to have an I/O device designated as a Requester, unless it is capable of handling partial data transfers.

3.6 DMA Controller Programming

Programming a DMA Channel to perform a needed DMA function is in general a four step process. First the global attributes of the DMA Controller are programmed via the two Command Registers. These global attributes include: priority levels, channel group enables, priority mode, and DREQn/EOP# input sampling.

The second step involves setting the operating modes of the particular channel. The Mode Registers are used to define the type of transfer and the handshaking modes. The Bus Size Register and Chaining Register may also need to be programmed in this step.

The third step is setting up the channel is to load the Base Registers in accordance with the needs of the operating modes chosen in step two. The Current Registers are automatically loaded from the Base Registers, if required by the Buffer Transfer Mode in effect. The information loaded and the order in which it is loaded depends on the operating mode. A channel used for cascading, for example, needs no buffer information and this step can be skipped entirely.

The last step is to enable the newly programmed channel using one of the Mask Registers. The channel is then available to perform the desired data transfer. The status of the channel can be observed at any time through the Status Register, Mask Register, Chaining Register, and Software Request register.

Once the channel is programmed and enabled, the DMA process may be initiated in one of two ways, either by a hardware DMA request (DREQn) or a software request (Software Request Register).

Once programmed to a particular Process/Mode configuration, the channel will operate in that configuration until programmed otherwise. For this reason, restarting a channel after the current buffer expires does not require complete reprogramming of the channel. Only those parameters which have changed need to be reprogrammed. The Byte Count Register is always changed and must be reprogrammed. A Target or Requester Address Register which is incremented or decremented should be reprogrammed also.

3.6.1 BUFFER PROCESSES

The Buffer Process is determined by the Auto-Initialize bit of Mode Register I and the Chaining Register. If Auto-Initialize is enabled, Chaining should not be used.

3.6.1.1 Single Buffer Process

The Single Buffer Process is programmed by disabling Chaining via the Chaining Register and programming Mode Register I for non-Auto-Initialize.

3.6.1.2 Buffer Auto-Initialize Process

Setting the Auto-Initialize bit in Mode Register I is all that is necessary to place the channel in this mode. Buffer Auto-Initialize must not be enabled simultaneous to enabling the Buffer Chaining Mode as this will have unpredictable results.

Once the Base Registers are loaded, the channel is ready to be enabled. The channel will reload its Current Registers from the Base Registers each time the Current Buffer expires, either by an expired Byte Count or an external EOP#.

3.6.1.3 Buffer Chaining Process

The Buffer Chaining Process is entered into from the Single Buffer Process. The Mode Registers should be programmed first, with all of the Transfer Modes defined as if the channel were to operate in the Single Buffer Process. The channel's Base and Current Registers are then loaded. When the channel has been set up in this way, and the chaining interrupt service routine is in place, the Chaining Process can be entered by programming the Chaining Register. Figure 3.23 illustrates the Buffer Chaining Process.

An interrupt (IRQ1) will be generated immediately after the Chaining Process is entered, as the channel

then perceives the Base Registers as empty and in need of reloading. It is important to have the interrupt service routine in place at the time the Chaining Process is entered into. The interrupt request is removed when the most significant byte of the Base Target Address is loaded.

The interrupt will occur again when the first buffer expires and the Current Registers are loaded from the Base Registers. The cycle continues until the Chaining Process is disabled, or the host fails to respond to IRQ1 before the Current Buffer expires.

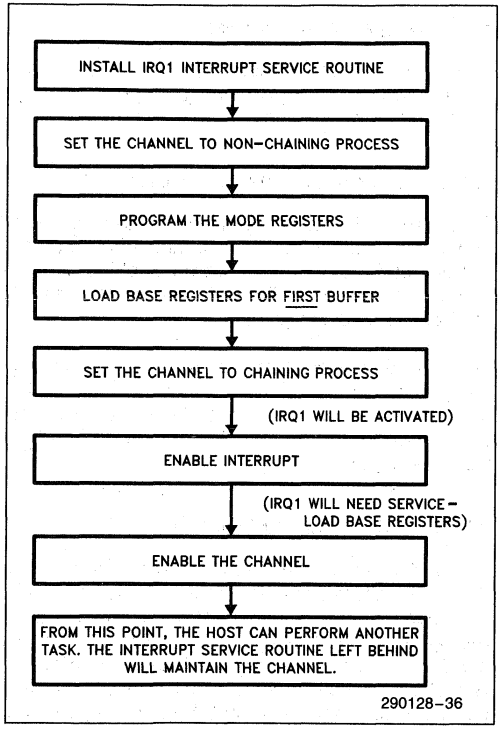


Figure 3-23. Flow of Events in the Buffer Chaining Process

Exiting the Chaining Process can be done by resetting the Chaining Mode Register. If an interrupt is pending for the channel when the Chaining Register is reset, the interrupt request will be removed. The Chaining Process can be temporarily disabled by setting the channel's Mask bit in the Mask Register.

The interrupt service routine for IRQ1 has the responsibility of reloading the Base Register as necessary. It should check the status of the channel to determine the cause of channel expiration, etc. It should also have access to operating system information regarding the channel, if any exists. The IRQ1 service routine should be capable of determining whether the chain should be continued or terminated and act on that information.

3.6.2 DATA TRANSFER MODES

The Data Transfer Modes are selected via Mode Register I. The Demand, Single, and Block Modes are selected by bits D6 and D7. The individual transfer type (Fly-By vs Two-Cycle, Read-Write-Verify, and I/O vs Memory) is programmed through both of the Mode registers.

3.6.3 CASCADED BUS MASTERS

The Cascade Mode is set by writing ones to D7 and D6 of Mode Register I. When a channel is programmed to operate in the Cascade Mode, all of the other modes associated with Mode Registers I and II are ignored. The priority and DREQn/EOP# definitions of the Command Registers will have the same effect on the channel's operation as any other mode.

3.6.4 SOFTWARE COMMANDS

There are five port addresses which, when written to, command certain operations to be performed by the 82380 DMA Controller. The data written to these locations is not of consequence, writing to the location is all that is necessary to command the 82380 to perform the indicated function. Following are descriptions of the command function.

Clear Byte Pointer Flip-Flop—location 000CH

Resets the Byte Pointer Flip-Flop. This command should be performed at the beginning of any access to the channel registers in order to be assured of beginning at a predictable place in the register programming sequence.

Master Clear—location 000DH

All DMA functions are set to their default states. This command is the equivalent of a hardware reset to the DMA Controller. Functions other than those in the DMA Controller section of the 82380 are not affected by this command.

Clear Mask Register —Channels 0–3—location 000EH
Channels 4–7—location 00CEH

This command simultaneously clears the Mask Bits of all channels in the addressed group, enabling all of the channels in the group.

Clear TC Interrupt Request—location 001EH

This command resets the Terminal Count Interrupt Request Flip-Flop. It is provided to allow the program which made a software DMA request to acknowledge that it has responded to the expiration of the requested channel(s).

3.7 Register Definitions

The following diagrams outline the bit definitions and functions of the 82380 DMA Controller's Status and Control Registers. The function and programming of the registers is covered in the previous section on DMA Controller Programming. An entry of 'X' as a bit value indicates "don't care."

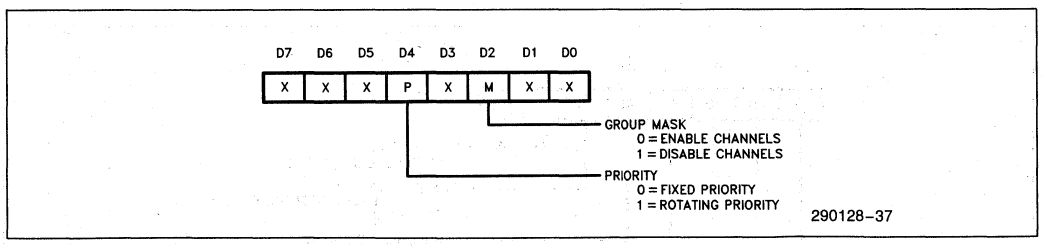
Channel Registers Channel	Register Name	(Read Current, Write Base) Address (Hex)	Byte Pointer	Bits Accessed
Channel 0	Target Address	00	0	0–7
			1	8–15
		87	x	16–23
		10	0	24–31
	Byte Count	01	0	0–7
			1	8–15
		11	0	16–23
	Requester Address	90	0	0-7
			1	8–15
		91	0	16–23
			1	24–31
Channel 1	Target Address	02	0	0–7
			1	8–15
		83	x	16–23
		12	0	24–31
	Byte Count	03	0	0–7
			1	8–15
		13	0	16–23
	Requester Address	92	0	0-7
			1	8–15
		93	0	16–23
			1	24–31

Channel Registers Channel	Register Name	(Read Current, Write Base) Address (Hex)	Byte Pointer	Bits Accessed
Channel 2	Target Address	04	0	0–7
			1	8–15
		81	x	16–23
		14	0	24–31
	Byte Count	05	0	0–7
			1	8–15
		15	0	16–23
	Requester Address	94	0	0-7
			1	8–15
		95	0	16–23
			1	24–31
Channel 3	Target Address	06	0	0–7
			1	8–15
		82	x	16–23
		16	0	24–31
	Byte Count	07	0	0–7
			1	8–15
		17	0	16–23
	Requester Address	96	0	0-7
			1	8–15
		97	0	16–23
			1	24–31
Channel 4	Target Address	C0	0	0–7
			1	8–15
		8F	x	16–23
		D0	0	24–31
	Byte Count	C1	0	0–7
			1	8–15
		D1	0	16–23
	Requester Address	98	0	0-7
			1	8–15
		99	0	16–23
			1	24–31
Channel 5	Target Address	C2	0	0–7
			1	8–15
		8B	x	16–23
		D2	0	24–31
	Byte Count	C3	0	0–7
			1	8–15
		D3	0	16–23
	Requester Address	9A	0	0-7
			1	8–15
		9B	0	16–23
			1	24–31

Channel Registers Channel	Register Name	(Read Current, Write Base) Address (Hex)	Byte Pointer	Bits Accessed
Channel 6	Target Address	C4	0	0–7
			1	8–15
		89	x	16–23
		D4	0	24–31
	Byte Count	C5	0	0–7
			1	8–15
		D5	0	16–23
	Requester Address	9C	0	0-7
			1	8–15
		9D	0	16–23
			1	24–31
Channel 7	Target Address	C6	0	0–7
			1	8–15
		8A	x	16–23
		D6	0	24–31
	Byte Count	C7	0	0–7
			1	8–15
		D7	0	16–23
	Requester Address	9E	0	0-7
			1	8–15
		9F	0	16–23
			1	24–31

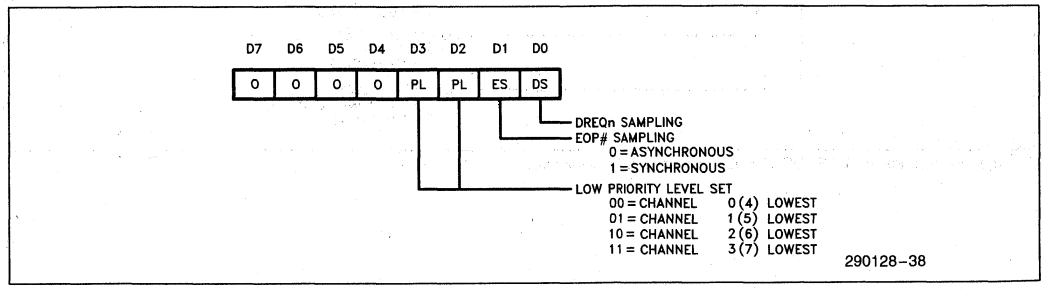
Command Register I (Write Only)

Port Address—Channels 0–3—0008H
Channels 4–7—00C8H



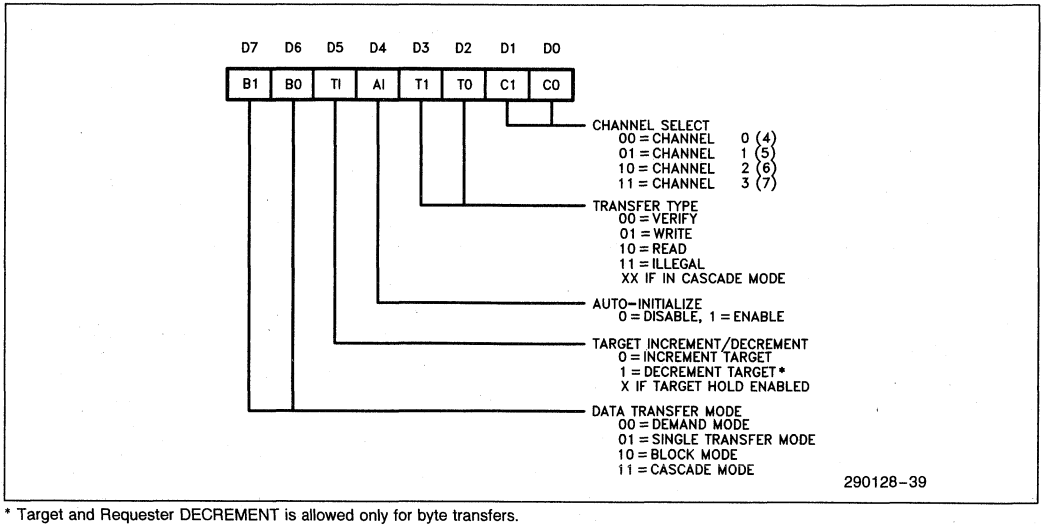
Command Register II (Write Only)

Port Addresses—Channels 0–3—001AH
Channels 4–7—00DAH



Mode Register I (Write Only)

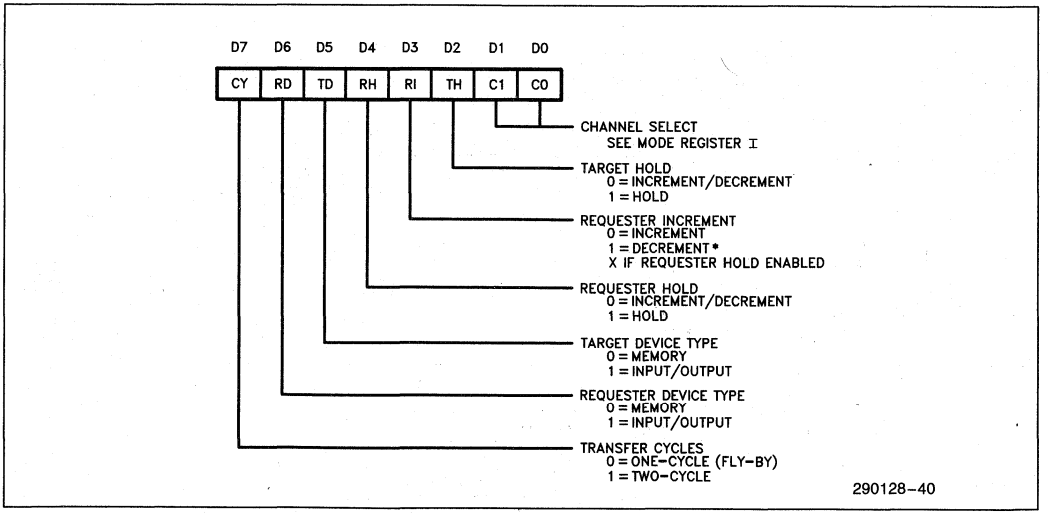
Port Addresses—Channels 0–3—000BH
Channels 4–7—00CBH



* Target and Requester DECREMENT is allowed only for byte transfers.

Mode Register II (Write Only)

Port Addresses—Channels 0–3—001BH
Channels 4–7—00DBH

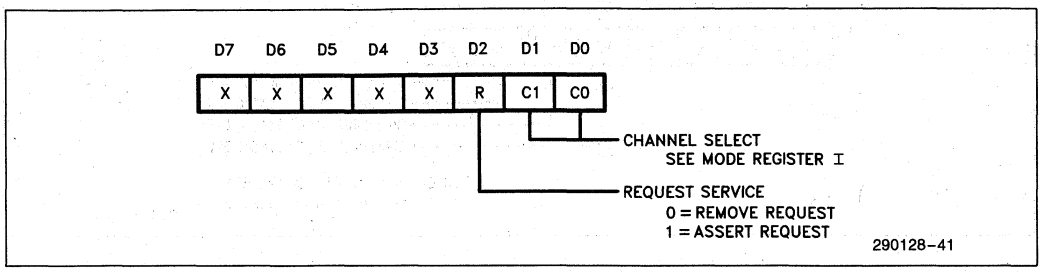


* Target and Requester DECREMENT is allowed only for byte transfers.

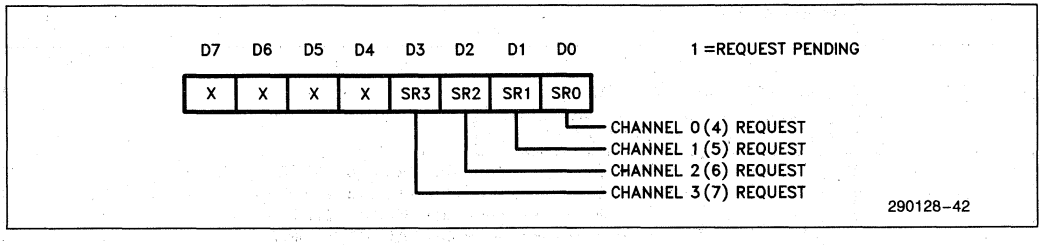
Software Request Register (Read/Write)

Port Addresses—Channels 0-3—0009H
Channels 4-7—00C9H

Write Format: Software DMA Service Request

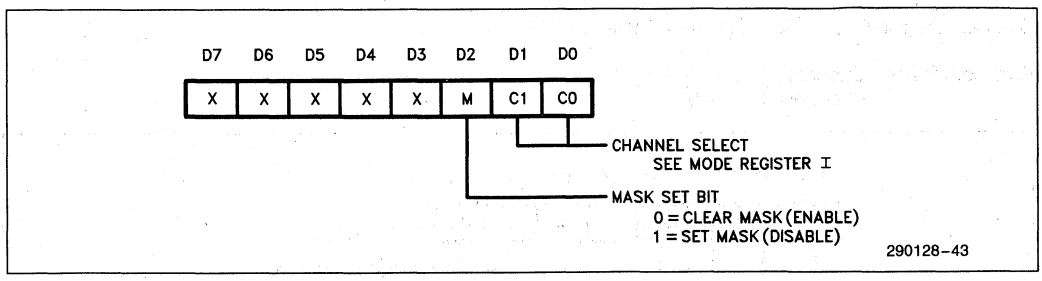


Read Format: Software Requests Pending



Mask Set/Reset Register Individual Channel Mask (Write Only)

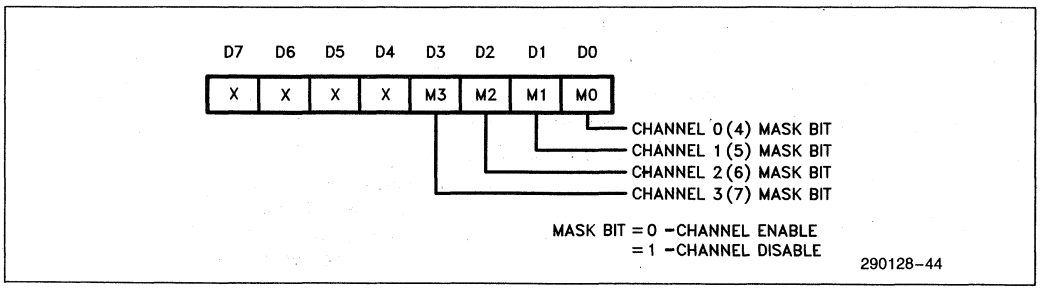
Port Addresses—Channels 0-3—000AH
Channels 4-7—00CAH



5

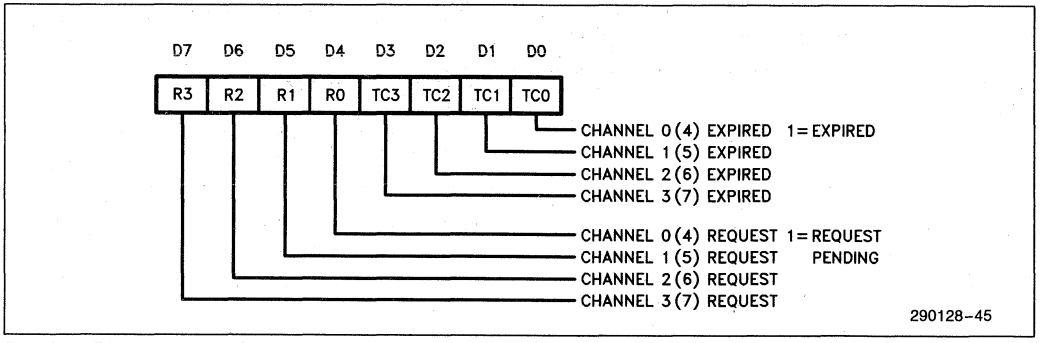
Mask Read/Write Register Group Channel Mask (Read/Write)

Port Addresses—Channels 0–3—000FH
Channels 4–7—00CFH



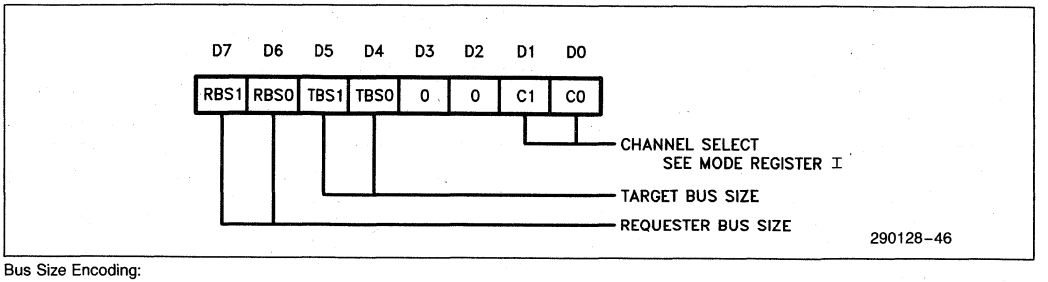
Status Register Channel Process Status (Read Only)

Port Addresses—Channels 0–3—0008H
Channels 4–7—00C8H



Bus Size Register Set Data Path Width (Write Only)

Port Addresses—Channels 0–3—0018H
Channels 4–7—00D8H

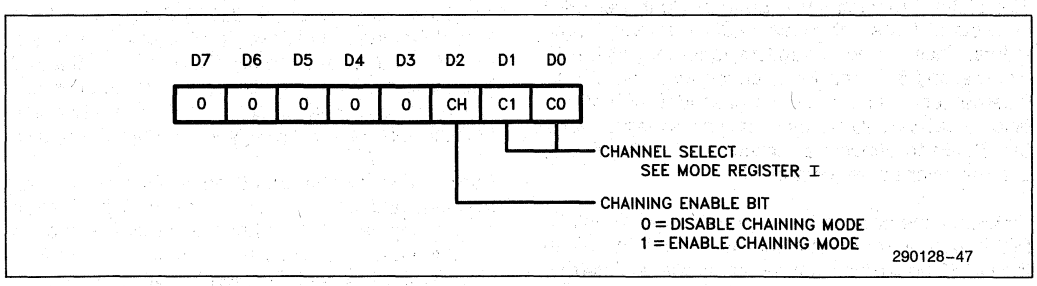


Bus Size Encoding:
00 = Reserved by Intel 10 = 16-bit Bus
01 = 32-bit Bus 11 = 8-bit Bus

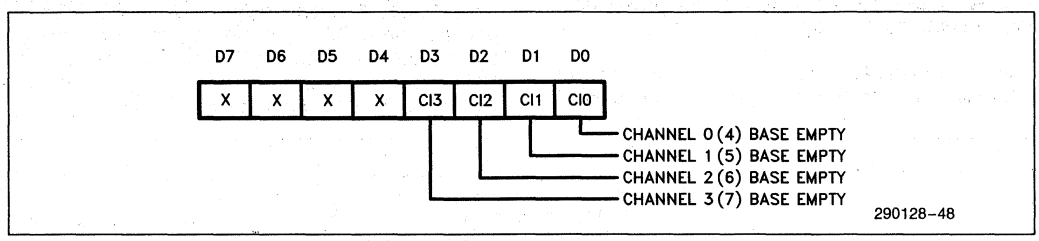
Chaining Register (Read/Write)

Port Addresses—Channels 0-3—0019H
Channels 4-7—00D9H

Write Format: Set Chaining Mode



Read Format: Channel Interrupt Status



3.8 8237A Compatibility

The register arrangement of the 82380 DMA Controller is a superset of the 8237A DMA Controller. Functionally the 82380 DMA Controller is very different from the 8237A. Most of the functions of the 8237A are performed also by the 82380. The following discussion points out the differences between the 8237A and the 82380.

The 8237A is limited to transfers between I/O and memory only (except in one special case, where two channels can be used to perform memory-to-memory transfers). The 82380 DMA Controller can transfer between any combination of memory and I/O. Several other features of the 8237A are enhanced or expanded in the 82380 and other features are added.

The 8237A is an 8-bit only DMA device. For programming compatibility, all of the 8-bit registers are preserved in the 82380. The 82380 is programmed via 8-bit registers. The address registers in the 82380 are 32-bit registers in order to support the 80386's 32-bit bus. The Byte Count Registers are 24-bit registers, allowing support of larger data blocks than possible with the 8237A.

All of the 8237A's operating modes are supported by the 82380 (except the cumbersome two-channel memory-to-memory transfer). The 82380 performs memory-to-memory transfers using only one channel. The 82380 has the added features of buffer pipelining (Buffer Chaining Process), programmable priority levels, and Byte Assembly.

The 82380 also adds the feature of address registers for both destination and source. These addresses may be incremented, decremented, or held constant, as required by the application of the individual channel. This allows any combination of destination and source device.

Each DMA channel has associated with it a Target and a Requester. In the 8237A, the Target is the device which can be accessed by the address register, the Requester is the device which is accessed by the DMA Acknowledge signals and must be an I/O device.

5

4.0 Programmable Interrupt Controller (PIC)

4.1 Functional Description

The 82380 Programmable Interrupt Controller (PIC) consists of three enhanced 82C59A Interrupt Controllers. These three controllers together provide 15 external and 5 internal interrupt request inputs. Each external request input can be cascaded with an additional 82C59A slave collector. This scheme allows the 82380 to support a maximum of 120 (15 x 8) external interrupt request inputs.

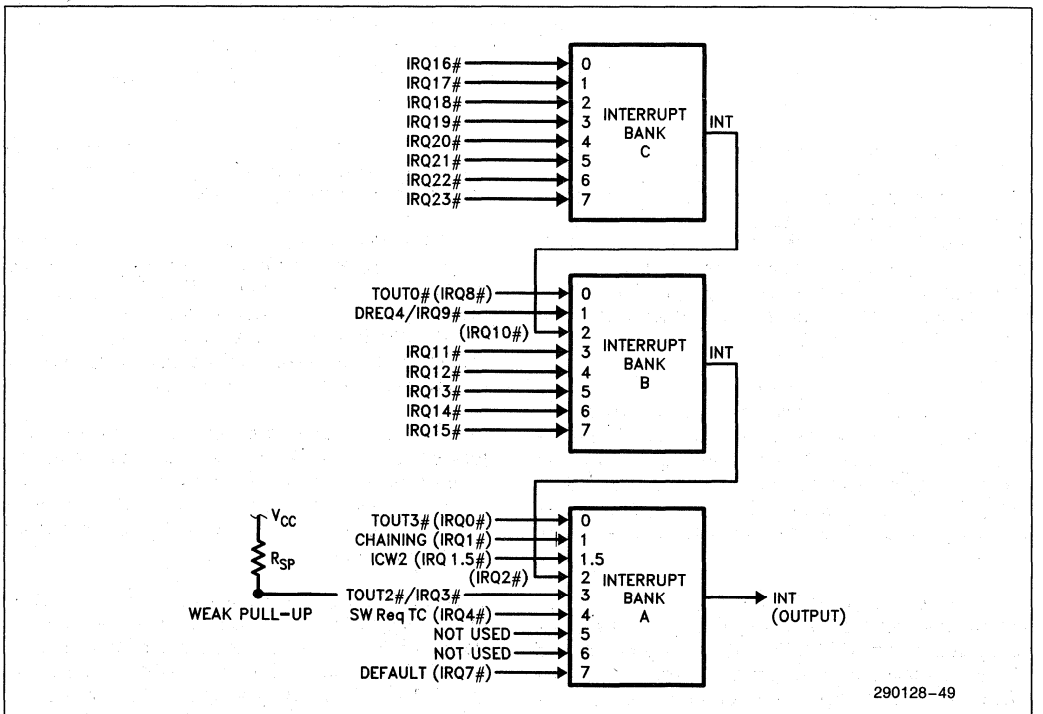
Following one or more interrupt requests, the 82380 PIC issues an interrupt signal to the 80386. When the 80386 host processor responds with an interrupt acknowledge signal, the PIC will arbitrate between the pending interrupt requests and place the interrupt vector associated with the highest priority pending request on the data bus.

The major enhancement in the 82380 PIC over the 82C59A is that each of the interrupt request inputs can be individually programmed with its own interrupt vector, allowing more flexibility in interrupt vector mapping.

4.1.1 INTERNAL BLOCK DIAGRAM

The block diagram of the 82380 Programmable Interrupt Controller is shown in Figure 4-1. Internally, the PIC consists of three 82C59A banks: A, B and C. The three banks are cascaded to one another: C is cascaded to B, B is cascaded to A. The INT output of Bank A is used externally to interrupt the 80386.

Bank A has nine interrupt request inputs (two are unused), and Banks B and C have eight interrupt request inputs. Of the fifteen external interrupt request inputs, two are shared by other functions. Specifically, the Interrupt Request 3 input (IRQ3#) can be used as the Timer 2 output (TOUT2#). This pin can be used in three different ways: IRQ3# input only, TOUT2# output only, or using TOUT2# to generate an IRQ3# interrupt request. Also, the Interrupt Request 9 input (IRQ 9#) can be used as DMA Request 4 input (DREQ4). Typically, only IRQ9# or DREQ4 can be used at a time.



NOTE:
Masking IRQ1.5# also masks IRQ2#

Figure 4-1. Interrupt Controller Block Diagram

4.1.2 INTERRUPT CONTROLLER BANKS

All three banks are identical, with the exception of the IRQ1.5 on Bank A. Therefore, only one bank will be discussed. In the 82380 PIC, all external requests can be cascaded into and each interrupt controller bank behaves like a master. As compared to the 82C59A, the enhancements in the banks are:

— All interrupt vectors are individually programmable. (In the 82C59A, the vectors must be programmed in eight consecutive interrupt vector locations.)

— The cascade address is provided on the Data Bus (D0–D7). (In the 82C59A, three dedicated control signals (CAS0, CAS1, CAS2) are used for master/slave cascading.)

The block diagram of a bank is shown in Figure 4-2. As can be seen from this figure, the bank consists of six major blocks: the Interrupt Request Register (IRR), the In-Service Register (ISR), the Interrupt Mask Register (IMR), the Priority Resolver (PR), the Vector Register (VR), and the Control Logic. The functional description of each block follows.

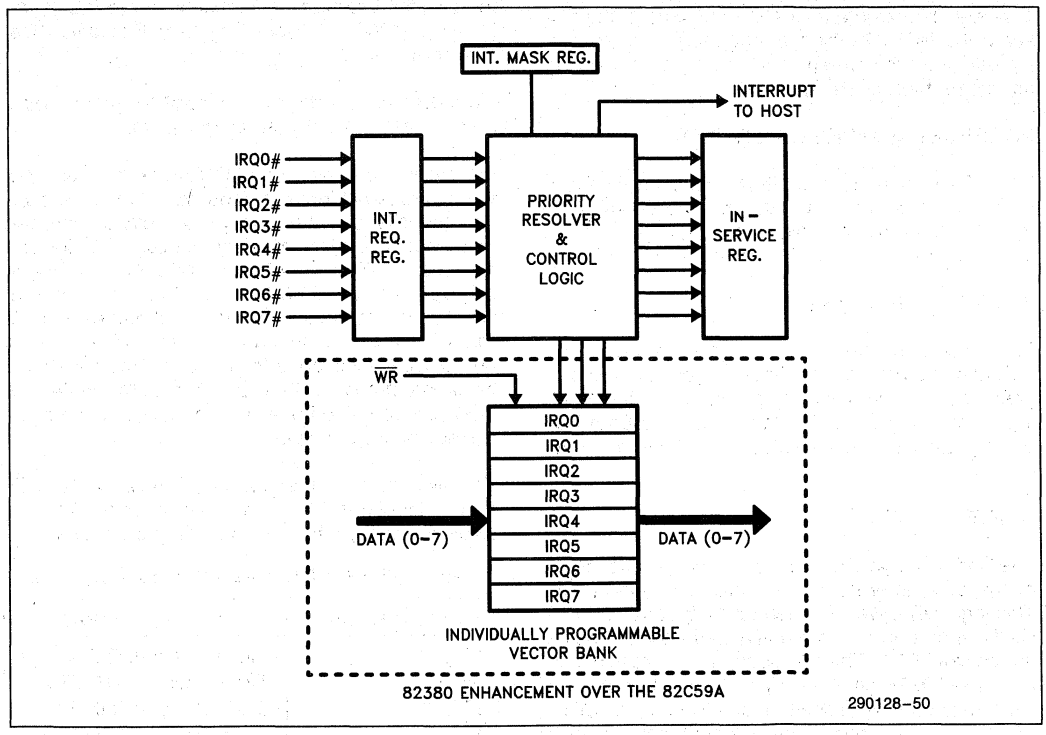


Figure 4-2. Interrupt Bank Block Diagram

5

INTERRUPT REQUEST (IRR) AND IN-SERVICE REGISTER (ISR)

The interrupts at the Interrupt Request (IRQ) input lines are handled by two registers in cascade, the Interrupt Request Register (IRR) and the In-Service Register (ISR). The IRR is used to store all interrupt levels which are requesting service; and the ISR is used to store all interrupt levels which are being serviced.

PRIORITY RESOLVER (PR)

This logic block determines the priorities of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during an Interrupt Acknowledge cycle.

INTERRUPT MASK REGISTER (IMR)

The IMR stores the bits which mask the interrupt lines to be masked (disabled). The IMR operates on the IRR. Masking of a higher priority input will not affect the interrupt request lines of lower priority.

VECTOR REGISTERS (VR)

This block contains a set of Vector Registers, one for each interrupt request line, to store the pre-programmed interrupt vector number. The corresponding vector number will be driven onto the Data Bus of the 82380 during the Interrupt Acknowledge cycle.

CONTROL LOGIC

The Control Logic coordinates the overall operations of the other internal blocks within the same bank. This logic will drive the Interrupt Output signal (INT) HIGH when one or more unmasked interrupt inputs are active (LOW). The INT output signal goes directly to the 80386 (in Bank A) or to another bank to which this bank is cascaded (see Figure 4-1). Also, this logic will recognize an Interrupt Acknowledge cycle (via M/IO#, D/C# and W/R# signals). During this bus cycle, the Control Logic will enable the corresponding Vector Register to drive the interrupt vector onto the Data Bus.

In Bank A, the Control Logic is also responsible for handling the special ICW2 interrupt request input (IRQ1.5#).

4.2 Interface Signals

4.2.1 INTERRUPT INPUTS

There are 15 external Interrupt Request inputs and 5 internal Interrupt Requests. The external request inputs are: IRQ3#, IRQ9#, IRQ11# to IRQ23#. They are shown in bold arrows in Figure 4-1. All IRQ inputs are active LOW and they can be programmed (via a control bit in the Initialization Command Word 1 (ICW1)) to be either edge-triggered or level-triggered. In order to be recognized as a valid interrupt request, the interrupt input must be active (LOW) until the first INTA# cycle (see Bus Functional Description).

Note that all 15 external Interrupt Request inputs have weak internal pull-up resistors.

As mentioned earlier, an 82C59A can be cascaded to each external interrupt input to expand the interrupt capacity to a maximum of 120 levels. Also, two of the interrupt inputs are dual functions: IRQ3# can be used as Timer 2 output (TOUT2#) and IRQ9# can be used as DREQ4 input. IRQ3# is a bidirectional dual function pin. This interrupt request input is wired-OR with the output of Timer 2 (TOUT2#). If only IRQ3# function is to be used, Timer 2 should be programmed so that OUT2 is LOW. Note that TOUT2# can also be used to generate an interrupt request to IRQ3# input.

The five internal interrupt requests serve special system functions. They are shown in Table 4-1. The following paragraphs describe these interrupts.

Table 4-1. 82380 Internal Interrupt Requests

Interrupt Request	Interrupt Source
IRQ0#	Timer 3 Output (TOUT3#)
IRQ8#	Timer 0 Output (TOUT0#)
IRQ1#	DMA Chaining Request
IRQ4#	DMA Terminal Count
IRQ1.5#	ICW2 Written

TIMER 0 AND TIMER 3 INTERRUPT REQUESTS [IRQ0#]

IRQ8# and IRQ0# interrupt requests are initiated by the output of Timers 0 and 3, respectively. Each of these requests is generated by an edge-detector flip-flop. The flip-flops are activated by the following conditions:

Set— Rising edge of timer output (TOUT);

Clear— Interrupt acknowledge for this request; OR Request is masked (disabled); OR Hardware Reset.

CHAINING AND TERMINAL COUNT INTERRUPTS [IRQ1#]

These interrupt requests are generated by the 82380 DMA Controller. The chaining request (IRQ1#) indicates that the DMA Base Register is not loaded. The Terminal Count request (IRQ4#) indicates that a software DMA request was cleared.

ICW2 INTERRUPT REQUEST [IRQ1.5#]

Whenever an Initialization Control Word 2 (ICW2) is written to a Bank, a special ICW2 interrupt request is generated. The interrupt will be cleared when the newly programmed ICW2 Register is read. This interrupt request is in Bank A at level 1.5. This interrupt request is internally ORed with the Cascaded Request from Bank B and is always assigned a higher priority than the Cascaded Request.

This special interrupt is provided to support compatibility with the original 82C59A. A detailed description of this interrupt is discussed in the Programming section.

DEFAULT INTERRUPT [IRQ7#]

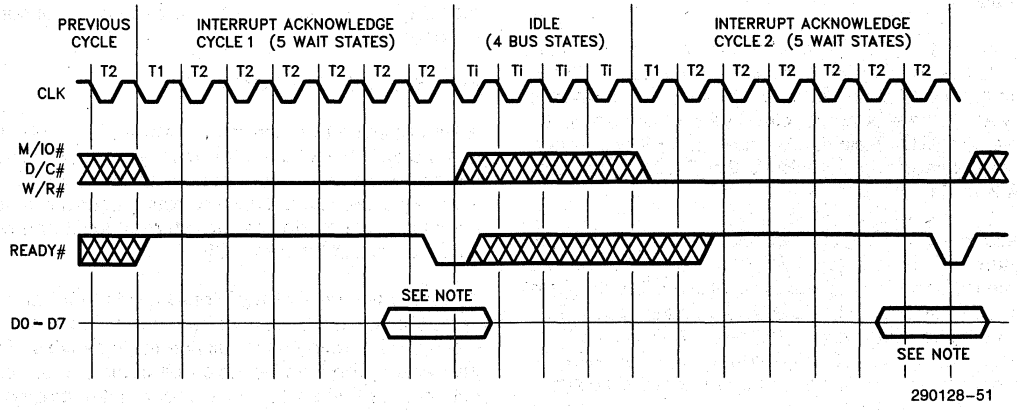
During an Interrupt Acknowledge cycle, if there is no active pending request, the PIC will automatically generate a default vector. This vector corresponds to the IRQ7# vector in Bank A.

4.2.2 INTERRUPT OUTPUT (INT)

The INT output pin is taken directly from bank A. This signal should be tied to the Maskable Interrupt Request (INTR) of the 80386. When this signal is active (HIGH), it indicates that one or more internal/external interrupt requests are pending. The 80386 is expected to respond with an interrupt acknowledge cycle.

4.3 Bus Functional Description

The INT output of bank A will be activated as a result of any unmasked interrupt request. This may be a non-cascaded or cascaded request. After the PIC has driven the INT signal HIGH, 80386 will respond by performing two interrupt acknowledge cycles. The timing diagram in Figure 4-3 shows a typical interrupt acknowledge process between the 82380 and the 80386 CPU.



NOTE:
What is actually driven on the Data Bus depends on if the current interrupt request is a Slave Request.

	INTA Cycle 1	INTA Cycle 2
NON-SLAVE REQUEST	00H	Vector
SLAVE REQUEST	Slave Address	High Impedance*

*Slave will place a vector at this time.

Figure 4-3. Interrupt Acknowledge Cycle

After activating the INT signal, the 82380 monitors the status lines (M/IO#, D/C#, W/R#) and waits for the 80386 to initiate the first interrupt acknowledge cycle. In the 80386 environment, two successive interrupt acknowledge cycles (INTA) marked by M/IO# = LOW, D/C# = LOW, and W/R# = LOW are performed. During the first INTA cycle, the PIC will determine the highest priority request. Assuming this interrupt input has no external Slave Controller cascaded to it, the 82380 will drive the Data Bus with 00H in the first INTA cycle. During the second INTA cycle, the 82380 PIC will drive the Data Bus with the corresponding preprogrammed interrupt vector.

If the PIC determines (from the ICW3) that this interrupt input has an external Slave Controller cascaded to it, it will drive the Data Bus with the specific Slave Cascade Address (instead of 00H) during the first INTA cycle. This Slave Cascade Address is the preprogrammed content in the corresponding Vector Register. This means that no Slave Address should be chosen to be 00H. Note that the Slave Address and Interrupt Vector are different interpretations of the same thing. They are both the contents of the programmable Vector Register. During the second INTA cycle, the Data Bus will be floated so that the external Slave Controller can drive its interrupt vector on the bus. Since the Slave Interrupt Controller resides on the system bus, bus transceiver enable and direction control logic must take this into consideration.

In order to have a successful interrupt service, the interrupt request input must be held active (LOW) until the beginning of the first interrupt acknowledge cycle. If there is no pending interrupt request when the first INTA cycle is generated, the PIC will generate a default vector, which is the IRQ7 vector (bank A level 7).

According to the Bus Cycle definition of the 80386, there will be four Bus Idle States between the two interrupt acknowledge cycles. These idle bus cycles will be initiated by the 80386. Also, during each interrupt acknowledge cycle, the internal Wait State Generator of the 82380 will automatically generate the required number of wait states for internal delays.

4.4 Mode of Operation

A variety of modes and commands are available for controlling the 82380 PIC. All of them are programmable; that is, they may be changed dynamically under software control. In fact, each bank can be programmed individually to operate in different modes. With these modes and commands, many possible configurations are conceivable, giving the user enough versatility for almost any interrupt controlled application.

This section is not intended to show how the 82380 PIC can be programmed. Rather, it describes the operation in different modes.

4.4.1 END-OF-INTERRUPT

Upon completion of an interrupt service routine, the interrupted bank needs to be notified so its ISR can be updated. This allows the PIC to keep track of which interrupt levels are in the process of being serviced and their relative priorities. Three different End-Of-Interrupt (EOI) formats are available. They are: Non-Specific EOI Command, Specific EOI Command, and Automatic EOI Mode. Selection of which EOI to use is dependent upon the interrupt operations the user wishes to perform.

If the 82380 is NOT programmed in the Automatic EOI Mode, an EOI command must be issued by the 80386 to the specific 82380 PIC Controller Bank. Also, if this controller bank is cascaded to another internal bank, an EOI command must also be sent to the bank to which this bank is cascaded. For example, if an interrupt request of Bank C in the 82380 PIC is serviced, an EOI should be written into Bank C, Bank B and Bank A. If the request comes from an external interrupt controller cascaded to Bank C, then an EOI should be written into the external controller as well.

NON-SPECIFIC EOI COMMAND

A Non-Specific EOI command sent from the 80386 lets the 82380 PIC bank know when a service routine has been completed, without specification of its exact interrupt level. The respective interrupt bank automatically determines the interrupt level and resets the correct bit in the ISR.

To take advantage of the Non-Specific EOI, the interrupt bank must be in a mode of operation in which it can predetermine its in-service routine levels. For this reason, the Non-Specific EOI command should only be used when the most recent level acknowledged and serviced is always the highest priority level (i.e., in the Fully Nested Mode structure to be described below). When the interrupt bank receives a Non-Specific EOI command, it simply resets the highest priority ISR bit to indicate that the highest priority routine in service is finished.

Special consideration should be taken when deciding to use the Non-Specific EOI command. Here are two operating conditions in which it is best NOT

used since the Fully Nested Mode structure will be destroyed:

— Using the Set Priority command within an interrupt service routine.

— Using a Special Mask Mode.

These conditions are covered in more detail in their own sections, but are listed here for reference.

SPECIFIC EOI COMMAND

Unlike a Non-Specific EOI command which automatically resets the highest priority ISR bit, a Specific EOI command specifies an exact ISR bit to be reset. Any one of the IRQ levels of an interrupt bank can be specified in the command.

The Specific EOI command is needed to reset the ISR bit of a completed service routine whenever the interrupt bank is not able to automatically determine it. The Specific EOI command can be used in all conditions of operation, including those that prohibit Non-Specific EOI command usage mentioned above.

AUTOMATIC EOI MODE

When programmed in the Automatic EOI Mode, the 80386 no longer needs to issue a command to notify the interrupt bank it has completed an interrupt routine. The interrupt bank accomplishes this by performing a Non-Specific EOI automatically at the end of the second INTA cycle.

Special consideration should be taken when deciding to use the Automatic EOI Mode because it may disturb the Fully Nested Mode structure. In the Automatic EOI Mode, the ISR bit of a routine in service is reset right after it is acknowledged, thus leaving no designation in the ISR that a service routine is being executed. If any interrupt request within the same bank occurs during this time and interrupts are enabled, it will get serviced regardless of its priority. Therefore, when using this mode, the 80386 should keep its interrupt request input disabled during execution of a service routine. By doing this, higher priority interrupt levels will be serviced only after the completion of a routine in service. This guideline restores the Fully Nested Mode structure. However, in this scheme, a routine in service cannot be interrupted since the host's interrupt request input is disabled.

4.4.2 INTERRUPT PRIORITIES

The 82380 PIC provides various methods for arranging the interrupt priorities of the interrupt request inputs to suit different applications. The following subsections explain these methods in detail.

4.4.2.1 Fully Nested Mode

The Fully Nested Mode of operation is a general purpose priority mode. This mode supports a multi-level interrupt structure in which all of the Interrupt Request (IRQ) inputs within one bank are arranged from highest to lowest.

Unless otherwise programmed, the Fully Nested Mode is entered by default upon initialization. At this time, IRQ0# is assigned the highest priority (priority = 0) and IRQ7# the lowest (priority = 7). This default priority can be changed, as will be explained later in the Rotating Priority Mode.

When an interrupt is acknowledged, the highest priority request is determined from the Interrupt Request Register (IRR) and its vector is placed on the bus. In addition, the corresponding bit in the In-Service Register (ISR) is set to designate the routine in service. This ISR bit will remain set until the 80386 issues an End Of Interrupt (EOI) command immediately before returning from the service routine; or alternately, if the Automatic End Of Interrupt (AEOI) bit is set, the ISR bit will be reset at the end of the second INTA cycle.

While the ISR bit is set, all further interrupts of the same or lower priority are inhibited. Higher level interrupts can still generate an interrupt, which will be acknowledged only if the 80386 internal interrupt enable flip-flop has been re-enabled (through software inside the current service routine).

4.4.2.2 Automatic Rotation—Equal Priority Devices

Automatic rotation of priorities serves in applications where the interrupting devices are of equal priority within an interrupt bank. In this kind of environment, once a device is serviced, all other equal priority peripherals should be given a chance to be serviced before the original device is serviced again. This is accomplished by automatically assigning a device the lowest priority after being serviced. Thus, in the worst case, the device would have to wait until all other peripherals connected to the same bank are serviced before it is serviced again.

There are two methods of accomplishing automatic rotation. One is used in conjunction with the Non-Specific EOI command and the other is used with the Automatic EOI mode. These two methods are discussed below.

ROTATE ON NON-SPECIFIC EOI COMMAND

When the Rotate On Non-Specific EOI command is issued, the highest ISR bit is reset as in a normal Non-Specific EOI command. However, after it is reset, the corresponding Interrupt Request (IRQ) level is assigned the lowest priority. Other IRQ priorities rotate to conform to the Fully Nested Mode based on the newly assigned low priority.

Figure 4-4 shows how the Rotate On Non-Specific EOI command affects the interrupt priorities. Assume the IRQ priorities were assigned with IRQ0 the highest and IRQ7 the lowest. IRQ6 and IRQ4 are already in service but neither is completed. Being the higher priority routine, IRQ4 is necessarily the routine being executed. During the IRQ4 routine, a rotate on Non-Specific EOI command is executed. When this happens, Bit 4 in the ISR is reset. IRQ4 then becomes the lowest priority and IRQ5 becomes the highest.

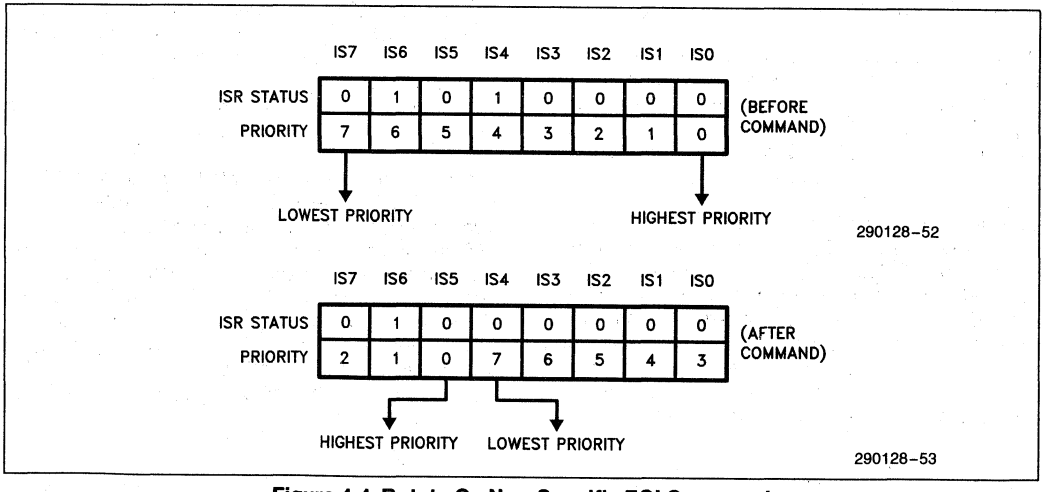


Figure 4-4. Rotate On Non-Specific EOI Command

ROTATE ON AUTOMATIC EOI MODE

The Rotate On Automatic EOI Mode works much like the Rotate On Non-Specific EOI Command. The main difference is that priority rotation is done automatically after the second INTA cycle of an interrupt request. To enter or exit this mode, a Rotate-On-Automatic-EOI Set Command and Rotate-On-Automatic-EOI Clear Command is provided. After this mode is entered, no other commands are needed as in the normal Automatic EOI Mode. However, it must be noted again that when using any form of the Automatic EOI Mode, special consideration should be taken. The guideline presented in the Automatic EOI Mode also applies here.

4.4.2.3 Specific Rotation—Specific Priority

Specific rotation gives the user versatile capabilities in interrupt controlled operations. It serves in those applications in which a specific device's interrupt priority must be altered. As opposed to Automatic Rotation which will automatically set priorities after each interrupt request is serviced, specific rotation is completely user controlled. That is, the user selects which interrupt level is to receive the lowest or the highest priority. This can be done during the main program or within interrupt routines. Two specific rotation commands are available to the user: Set Priority Command and Rotate On Specific EOI Command.

SET PRIORITY COMMAND

The Set Priority Command allows the programmer to assign an IRQ level the lowest priority. All other interrupt levels will conform to the Fully Nested Mode based on the newly assigned low priority.

ROTATE ON SPECIFIC EOI COMMAND

The Rotate On Specific EOI Command is literally a combination of the Set Priority Command and the Specific EOI Command. Like the Set Priority Command, a specified IRQ level is assigned lowest priority. Like the Specific EOI Command, a specified level will be reset in the ISR. Thus, this command accomplishes both tasks in one single command.

4.4.2.4 Interrupt Priority Mode Summary

In order to simplify understanding the many modes of interrupt priority, Table 4-2 is provided to bring out their summary of operations.

Table 4-2. Interrupt Priority Mode Summary

Interrupt Priority Mode	Operation Summary	Effect On Priority After EOI	
		Non-Specific/Automatic	Specific
Fully-Nested Mode	IRQ0#-Highest Priority IRQ7#-Lowest Priority	No change in priority. Highest ISR bit is reset.	Not Applicable.
Automatic Rotation (Equal Priority Devices)	Interrupt level just serviced is the lowest priority. Other priorities rotate to conform to Fully-Nested Mode.	Highest ISR bit is reset and the corresponding level becomes the lowest priority.	Not Applicable.
Specific Rotation (Specific Priority Devices)	User specifies the lowest priority level. Other priorities rotate to conform to Fully-Nested Mode.	Not Applicable.	As described under 'Operation Summary'.

4.4.3 INTERRUPT MASKING

VIA INTERRUPT MASK REGISTER

Each bank in the 82380 PIC has an Interrupt Mask Register (IMR) which enhances interrupt control capabilities. This IMR allows individual IRQ masking. When an IRQ is masked, its interrupt request is disabled until it is unmasked. Each bit in the 8-bit IMR disables one interrupt channel if it is set (HIGH). Bit 0 masks IRQ0, Bit 1 masks IRQ1 and so forth. Masking an IRQ channel will only disable the corresponding channel and does not affect the others operations.

The IMR acts only on the output of the IRR. That is, if an interrupt occurs while its IMR bit is set, this request is not 'forgotten'. Even with an IRQ input masked, it is still possible to set the IRR. Therefore, when the IMR bit is reset, an interrupt request to the 80386 will then be generated, providing that the IRQ request remains active. If the IRQ request is removed before the IMR is reset, the Default Interrupt Vector (Bank A, level 7) will be generated during the interrupt acknowledge cycle.

SPECIAL MASK MODE

In the Fully Nested Mode, all IRQ levels of lower priority than the routine in service are inhibited. However, in some applications, it may be desirable to let a lower priority interrupt request to interrupt the routine in service. One method to achieve this is by using the Special Mask Mode. Working in conjunction with the IMR, the Special Mask Mode enables interrupts from all levels except the level in service. This is usually done inside an interrupt service routine by masking the level that is in service and then issuing the Special Mask Mode Command. Once the Special Mask Mode is enabled, it remains in effect until it is disabled.

4.4.4 EDGE OR LEVEL INTERRUPT TRIGGERING

Each bank in the 82380 PIC can be programmed independently for either edge or level sensing for the interrupt request signals. Recall that all IRQ inputs are active LOW. Therefore, in the edge triggered mode, an active edge is defined as an input transition from an inactive (HIGH) to active (LOW) state. The interrupt input may remain active without generating another interrupt. During level triggered mode, an interrupt request will be recognized by an active (LOW) input, and there is no need for edge detection. However, the interrupt request must be removed before the EOI Command is issued, or the 80386 must be disabled to prevent a second false interrupt from occurring.

In either modes, the interrupt request input must be active (LOW) during the first INTA cycle in order to be recognized. Otherwise, the Default Interrupt Vector will be generated at level 7 of Bank A.

4.4.5 INTERRUPT CASCADING

As mentioned previously, the 82380 allows for external Slave interrupt controllers to be cascaded to any of its external interrupt request pins. The 82380 PIC indicates that a external Slave Controller is to be serviced by putting the contents of the Vector Register associated with the particular request on the 80386 Data Bus during the first INTA cycle (instead of 00H during a non-slave service). The external logic should latch the vector on the Data Bus using the INTA status signals and use it to select the external Slave Controller to be serviced (see Figure 4-5). The selected Slave will then respond to the second INTA cycle and place its vector on the Data Bus. This method requires that if external Slave Controllers

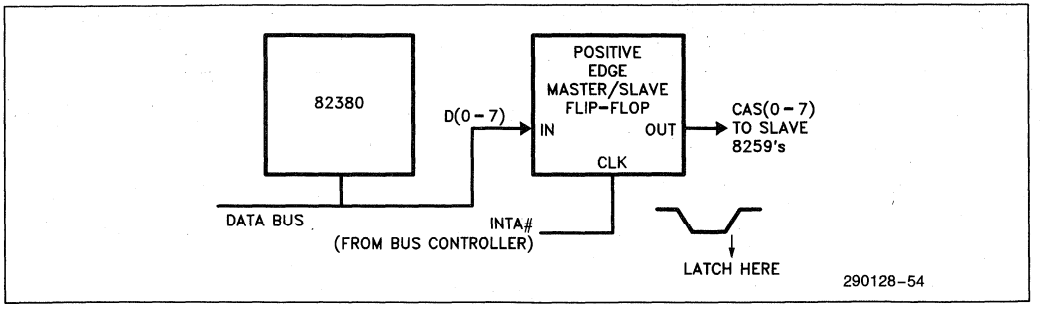


Figure 4-5. Slave Cascade Address Capturing

are used in the system, no vector should be programmed to 00H.

Since the external Slave Cascade Address is provided on the Data Bus during INTA cycle 1, an external latch is required to capture this address for the Slave Controller. A simple scheme is depicted in Figure 4-5.

4.4.5.1 Special Fully Nested Mode

This mode will be used where cascading is employed and the priority is to be conserved within each Slave Controller. The Special Fully Nested Mode is similar to the 'regular' Fully Nested Mode with the following exceptions:

— When an interrupt request from a Slave Controller is in service, this Slave Controller is not locked out from the Master's priority logic. Further interrupt requests from the higher priority logic within the Slave Controller will be recognized by the 82380 PIC and will initiate interrupts to the 80386. In comparing to the 'regular' Fully Nested Mode, the Slave Controller is masked out when its request is in service and no higher requests from the same Slave Controller can be serviced.

— Before exiting the interrupt service routine, the software has to check whether the interrupt serviced was the only request from the Slave Controller. This is done by sending a Non-Specific EOI Command to the Slave Controller and then reading its In Service Register. If there are no requests in the Slave Controller, a Non-Specific EOI can be sent to the corresponding 82380 PIC bank also. Otherwise, no EOI should be sent.

4.4.6 READING INTERRUPT STATUS

The 82380 PIC provides several ways to read different status of each interrupt bank for more flexible interrupt control operations. These include polling the highest priority pending interrupt request and reading the contents of different interrupt status registers.

4.4.6.1 Poll Command

The 82380 PIC supports status polling operations with the Poll Command. In a Poll Command, the pending interrupt request with the highest priority can be determined. To use this command, the INT output is not used, or the 80386 interrupt is disabled. Service to devices is achieved by software using the Poll Command.

This mode is useful if there is a routine command common to several levels so that the INTA sequence is not needed. Another application is to use the Poll Command to expand the number of priority levels.

Notice that the ICW2 mechanism is not supported for the Poll Command. However, if the Poll Command is used, the programmable Vector Registers are of no concern since no INTA cycle will be generated.

4.4.6.2 Reading Interrupt Registers

The contents of each interrupt register (IRR, ISR, and IMR) can be read to update the user's program on the present status of the 82380 PIC. This can be a versatile tool in the decision making process of a service routine, giving the user more control over interrupt operations.

The reading of the IRR and ISR contents can be performed via the Operation Control Word 3 by using a Read Status Register Command and the content of IMR can be read via a simple read operation of the register itself.

5

4.5 Register Set Overview

Each bank of the 82380 PIC consists of a set of 8-bit registers to control its operations. The address map of all the registers is shown in Table 4-3. Since all three register sets are identical in functions, only one set will be described.

Functionally, each register set can be divided into five groups. They are: the four Initialization Command Words (ICW's), the three Operation Control Words (OCW's), the Poll/Interrupt Request/In-Service Register, the Interrupt Mask Register, and the Vector Registers. A description of each group follows.

Table 4-3. Interrupt Controller Register Address Map

Port Address	Access	Register Description
20H	Write	Bank B ICW1, OCW2, or OCW3
	Read	Bank B Poll, Request or In-Service Status Register
21H	Write	Bank B ICW2, ICW3, ICW4, OCW1
	Read	Bank B Mask Register
22H	Read	Bank B ICW2
28H	Read/Write	IRQ8 Vector Register
29H	Read/Write	IRQ9 Vector Register
2AH	Read/Write	Reserved
2BH	Read/Write	IRQ11 Vector Register
2CH	Read/Write	IRQ12 Vector Register
2DH	Read/Write	IRQ13 Vector Register
2EH	Read/Write	IRQ14 Vector Register
2FH	Read/Write	IRQ15 Vector Register
A0H	Write	Bank C ICW1, OCW2, or OCW3
	Read	Bank C Poll, Request or In-Service Status Register
A1H	Write	Bank C ICW2, ICW3, ICW4, OCW1
	Read	Bank C Mask Register
A2H	Read	Bank C ICW2
A8H	Read/Write	IRQ16 Vector Register
A9H	Read/Write	IRQ17 Vector Register
AAH	Read/Write	IRQ18 Vector Register
ABH	Read/Write	IRQ19 Vector Register
ACH	Read/Write	IRQ20 Vector Register
ADH	Read/Write	IRQ21 Vector Register
AEH	Read/Write	IRQ22 Vector Register
AFH	Read/Write	IRQ23 Vector Register
30H	Write	Bank A ICW1, OCW2, or OCW3
	Read	Bank A Poll, Request or In-Service Status Register
31H	Write	Bank A ICW2, ICW3, ICW4, OCW1
	Read	Bank A Mask Register
32H	Read	Bank ICW2
38H	Read/Write	IRQ0 Vector Register
39H	Read/Write	IRQ1 Vector Register
3AH	Read/Write	IRQ1.5 Vector Register
3BH	Read/Write	IRQ3 Vector Register
3CH	Read/Write	IRQ4 Vector Register
3DH	Read/Write	Reserved
3EH	Read/Write	Reserved
3FH	Read/Write	IRQ7 Vector Register

4.5.1 INITIALIZATION COMMAND WORDS (ICW)

Before normal operation can begin, the 82380 PIC must be brought to a known state. There are four 8-bit Initialization Command Words in each interrupt bank to setup the necessary conditions and modes for proper operation. Except for the second common word (ICW2) which is a read/write register, the other three are write-only registers. Without going into detail of the bit definitions of the command words, the following subsections give a brief description of what functions each command word controls.

ICW1

The ICW1 has three major functions. They are:

— To select between the two IRQ input triggering modes (edge-or level-triggered);

— To designate whether or not the interrupt bank is to be used alone or in the cascade mode. If the cascade mode is desired, the interrupt bank will accept ICW3 for further cascade mode programming. Otherwise, no ICW3 will be accepted;

— To determine whether or not ICW4 will be issued; that is, if any of the ICW4 operations are to be used.

ICW2

ICW2 is provided for compatibility with the 82C59A only. Its contents do not affect the operation of the interrupt bank in any way. Whenever the ICW2 of any of the three banks is written into, an interrupt is generated from Bank A at level 1.5. The interrupt request will be cleared after the ICW2 register has been read by the 80386. The user is expected to program the corresponding vector register or to use it as an indicator that an attempt was made to alter the contents. Note that each ICW2 register has different addresses for read and write operations.

ICW3

The interrupt bank will only accept an ICW3 if programmed in the external cascade mode (as indicated in ICW1). ICW3 is used for specific programming within the cascade mode. The bits in ICW3 indicate which interrupt request inputs have a Slave cascaded to them. This will subsequently affect the interrupt vector generation during the interrupt acknowledge cycles as described previously.

ICW4

The ICW4 is accepted only if it was selected in ICW1. This command word register serves two functions:

— To select either the Automatic EOI mode or software EOI mode;

— To select if the Special Nested mode is to be used in conjunction with the cascade mode.

4.5.2 OPERATION CONTROL WORDS (OCW)

Once initialized by the ICW's, the interrupt banks will be operating in the Fully Nested Mode by default and they are ready to accept interrupt requests. However, the operations of each interrupt bank can be further controlled or modified by the use of OCW's. Three OCW's are available for programming various modes and commands. Note that all OCW's are 8-bit write-only registers.

The modes and operations controlled by the OCW's are:

— Fully Nested Mode;
— Rotating Priority Mode;
— Special Mask Mode;
— Poll Mode;
— EOI Commands;
— Read Status Commands.

OCW1

OCW1 is used solely for masking operations. It provides a direct link to the Interrupt Mask Register (IMR). The 80386 can write to this OCW register to enable or disable the interrupt inputs. Reading the pre-programmed mask can be done via the Interrupt Mask Register which will be discussed shortly.

OCW2

OCW2 is used to select End-Of-Interrupt, Automatic Priority Rotation, and Specific Priority Rotation operations. Associated commands and modes of these operations are selected using the different combinations of bits in OCW2.

Specifically, the OCW2 is used to:

— Designate an interrupt level (0–7) to be used to reset a specific ISR bit or to set a specific priority. This function can be enabled or disabled;

— Select which software EOI command (if any) is to be executed (i.e., Non-Specific or Specific EOI);

— Enable one of the priority rotation operations (i.e., Rotate On Non-Specific EOI, Rotate On Automatic EOI, or Rotate on Specific EOI).

OCW3

There are three main categories of operation that OCW3 controls. That are summarized as follows:

— To select and execute the Read Status Register Commands, either reading the Interrupt Request Register (IRR) or the In-Service Register (ISR);
— To issue the Poll Command. The Poll Command will override a Read Register Command if both functions are enabled simultaneously;
— To set or reset the Special Mask Mode.

4.5.3 POLL/INTERRUPT REQUEST/IN-SERVICE STATUS REGISTER

As the name implies, this 8-bit read-only register has multiple functions. Depending on the command issued in the OCW3, the content of this register reflects the result of the command executed. For a Poll Command, the register read contains the binary code of the highest priority level requesting service (if any). For a Read IRR Command, the register content will show the current pending interrupt request(s). Finally, for a Read ISR Command, this register will specify all interrupt levels which are being serviced.

4.5.4 INTERRUPT MASK REGISTER (IMR)

This is a read-only 8-bit register which, when read, will specify all interrupt levels within the same bank that are masked.

4.5.5 VECTOR REGISTER (VR)

Each interrupt request input has an 8-bit read/write programmable vector register associated with it. The registers should be programmed to contain the interrupt vector for the corresponding request. The contents of the Vector Register will be placed on the Data Bus during the INTA cycles as described previously.

4.6 Programming

Programming the 82380 PIC is accomplished by using two types of command words: ICW's and OCW's. All modes and commands explained in the previous sections are programmable using the ICW's and OCW's. The ICW's are issued from the 80386 in a sequential format and are used to setup the banks in the 82380 PIC in an initial state of operation. The OCW's are issued as needed to vary and control the 82380 PIC's operations.

Both ICW's and OCW's are sent by the 80386 to the interrupt banks via the Data Bus. Each bank distinguishes between the different ICW's and OCW's by the I/O address map, the sequence they are issued (ICW's only), and by some dedicated bits among the ICW's and OCW's.

All three interrupt banks are programmed in a similar way. Therefore, only a single bank will be described.

4.6.1 INITIALIZATION (ICW)

Before normal operation can begin, each bank must be initialized by programming a sequence of two to four bytes written into the ICW's.

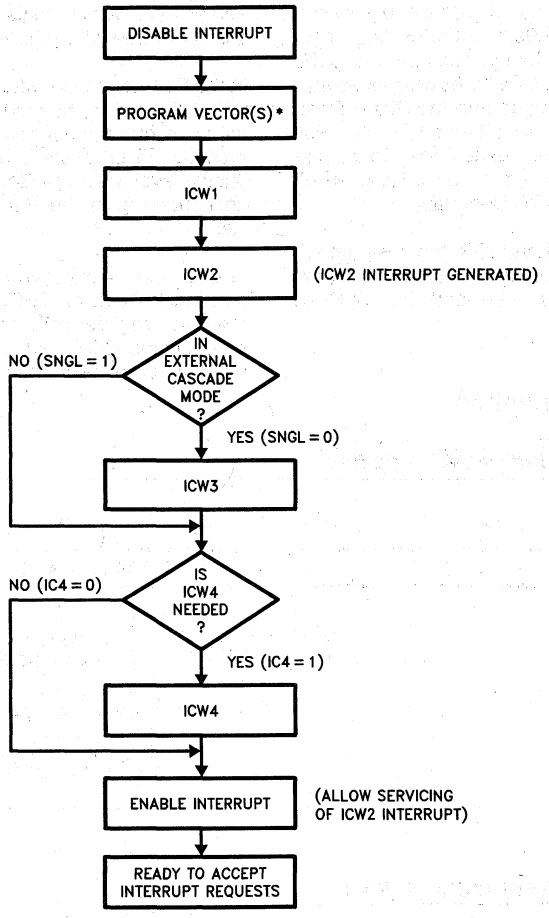
Figure 4-6 shows the initialization flow for an interrupt bank. Both ICW1 and ICW2 must be issued for any form of operation. However, ICW3 and ICW4 are used only if designated in ICW1. Once initialized, if any programming changes within the ICW's are to be made, the entire ICW sequence must be reprogrammed, not just an individual ICW.

Note that although the ICW2's in the 82380 PIC do not affect the Bank's operation, they still must be programmed in order to preserve the compatibility with the 82C59A. The contents programmed are not relevant to the overall operations of the interrupt banks. Also, whenever one of the three ICW2's is programmed, an interrupt level 1.5 in Bank A will be generated. This interrupt request will be cleared upon reading of the ICW2 registers. Since the three ICW2's share the same interrupt level and the system may not know the origin of the interrupt, all three ICW2's must be read.

However, it is not necessary to provide an interrupt service routine for the ICW2 interrupt. One way to avoid this is as follows. At the beginning of the initialization of the interrupt banks, the 80386 interrupt should be disabled. After each ICW2 register write operation is performed during the initialization, the corresponding ICW2 register is read. This read operation will clear the interrupt request of the 82380. At the end of the initialization, the 80386 interrupt is reenabled. With this method, the 80386 will not detect the ICW2 interrupt request, thus eliminating the need of an interrupt service routine.

Certain internal setup conditions occur automatically within the interrupt bank after the first ICW (ICW1) has been issued. They are:

— The edge sensitive circuit is reset, which means that following initialization, an interrupt request input must make a HIGH-to-LOW transition to generate an interrupt;
— The Interrupt Mask Register (IMR) is cleared; that is, all interrupt inputs are enabled;
— IRQ7 input of each bank is assigned priority 7 (lowest);
— Special Mask Mode is cleared and Status Read is set to IRR;
— If no ICW4 is needed, then no Automatic-EOI is selected.



*ICW2 vector address must be programmed now.
Other vector addresses may be programmed via ICW2 interrupt service routine.

Figure 4-6. Initialization Sequence

4.6.2 VECTOR REGISTERS (VR)

Each interrupt request input has a separate Vector Register. These Vector Registers are used to store the pre-programmed vector number corresponding to their interrupt sources. In order to guarantee proper interrupt handling, all Vector Registers must be programmed with the predefined vector numbers. Since an interrupt request will be generated whenever an ICW2 is written during the initialization sequence, it is important that the Vector Register of IRQ1.5 in Bank A should be initialized and the interrupt service routine of this vector is set up before the ICW's are written.

4.6.3 OPERATION CONTROL WORDS (OCW)

After the ICW's are programmed, the operations of each interrupt controller bank can be changed by writing into the OCW's as explained before. There is no special programming sequence required for the OCW's. Any OCW may be written at any time in order to change the mode of or to perform certain operations on the interrupt banks.

4.6.3.1 Read Status and Poll Commands (OCW3)

Since the reading of IRR and ISR status as well as the result of a Poll Command are available on the

same read-only Status Register, a special Read Status/Poll Command must be issued before the Poll/Interrupt Request/In-Service Status Register is read. This command can be specified by writing the required control word into OCW3. As mentioned earlier, if both the Poll Command and the Status Read Command are enabled simultaneously, the Poll Command will override the Status Read. That is, after the command execution, the Status Register will contain the result of the Poll Command.

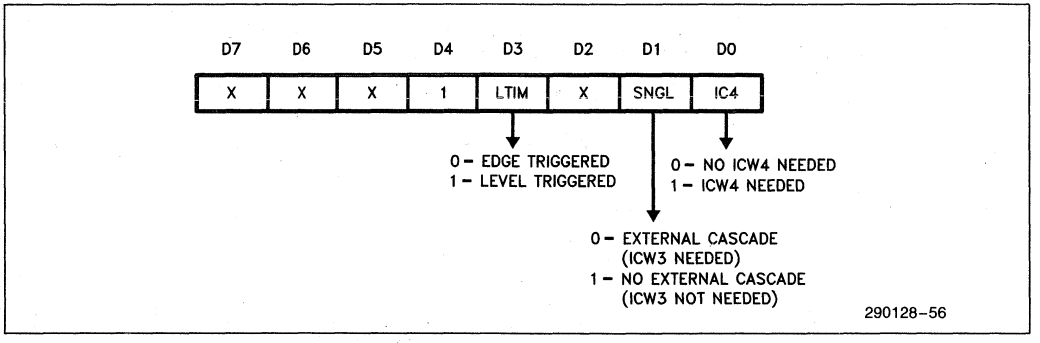
Note that for reading IRR and ISR, there is no need to issue a Read Status Command to the OCW3 every time the IRR or ISR is to be read. Once a Read Status Command is received by the interrupt bank, it 'remembers' which register is selected. However, this is not true when the Poll Command is used.

In the Poll Command, after the OCW3 is written, the 82380 PIC treats the next read to the Status Register as an interrupt acknowledge. This will set the appropriate IS bit if there is a request and read the priority level. Interrupt Request input status remains unchanged from the Poll Command to the Status Read.

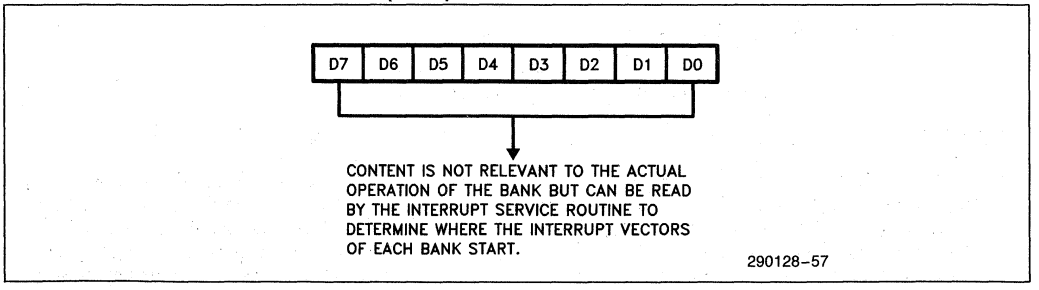
In addition to the above read commands, the Interrupt Mask Register (IMR) can also be read. When read, this register reflects the contents of the preprogrammed OCW1 which contains information on which interrupt request(s) is(are) currently disabled.

4.7 Register Bit Definition

INITIALIZATION COMMAND WORD 1 (ICW1)

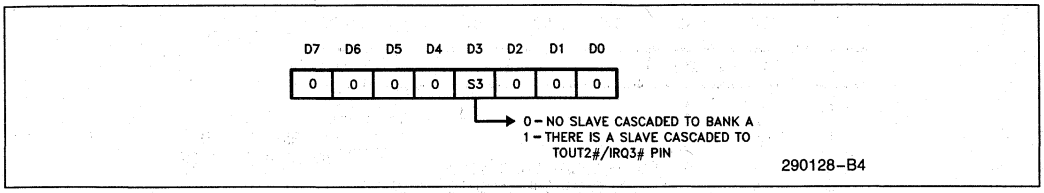


INITIALIZATION COMMAND WORD 2 (ICW2)

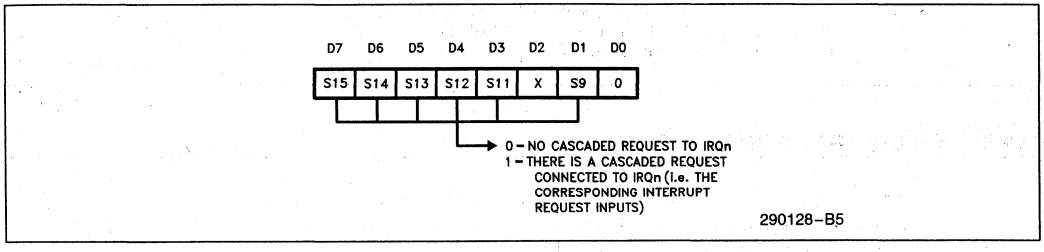


INITIALIZATION COMMAND WORD 3 (ICW3)

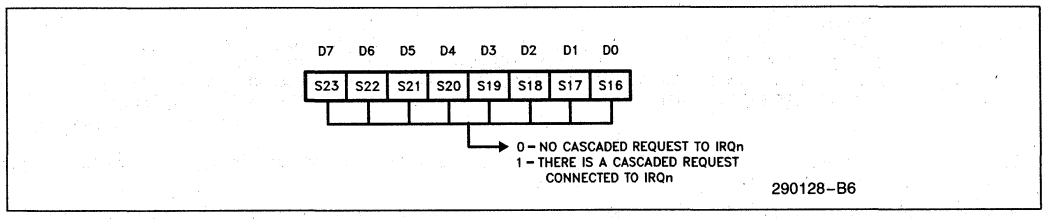
ICW3 for Bank A:



ICW3 for Bank B:

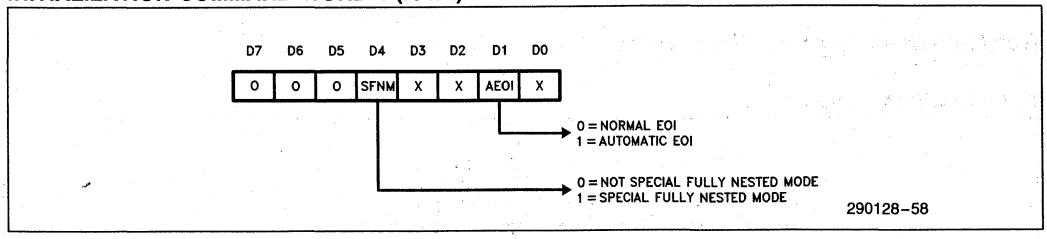


ICW3 for Bank C:

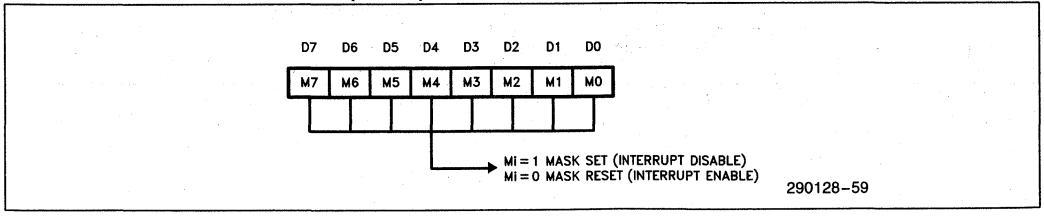


5

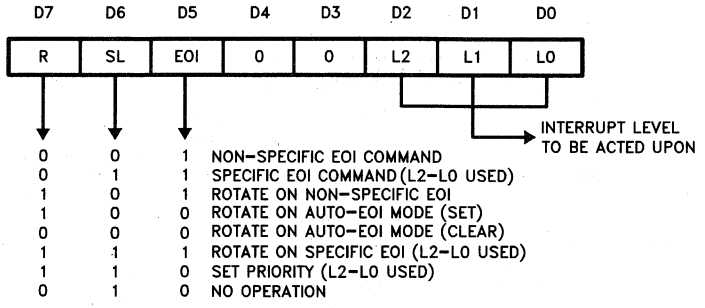
INITIALIZATION COMMAND WORD 4 (ICW4)



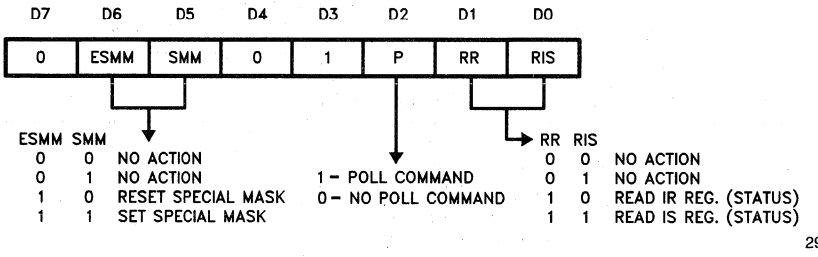
OPERATION CONTROL WORD 1 (OCW1)



OPERATION CONTROL WORD 2 (OCW2)



OPERATION CONTROL WORD 3 (OCW3)

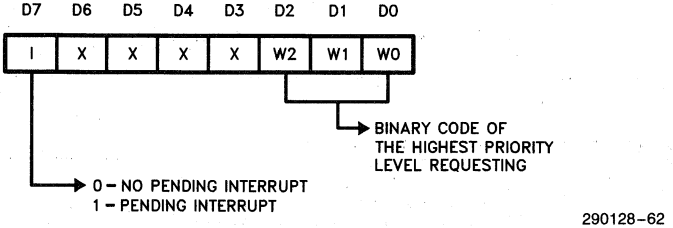


ESMM—Enable Special Mask Mode. When this bit is set to 1, it enables the SMM bit to set or reset the Special Mask Mode. When this bit is set to 0, SMM bit becomes don't care.

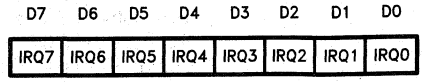
SMM—Special Mask Mode. If ESMM = 1 and SMM = 1, the interrupt controller bank will enter Special Mask Mode. If ESMM = 1 and SMM = 0, the bank will revert to normal mask mode. When ESMM = 0, SMM has no effect.

Poll/Interrupt Request/In-Service Status Register

POLL COMMAND STATUS



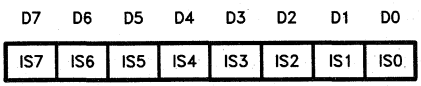
INTERRUPT REQUEST STATUS



290128-63

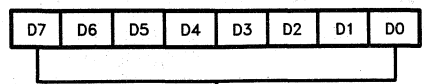
NOTE:
Although all Interrupt Request inputs are active LOW, the internal logical will invert the state of the pins so that when there is a pending interrupt request at the input, the corresponding IRQ bit will be set to HIGH in the Interrupt Request Status register.

IN-SERVICE STATUS



290128-64

VECTOR REGISTER (VR)



290128-65

4.8 Register Operational Summary

For ease of reference, Table 4-4 gives a summary of the different operating modes and commands with their corresponding registers.

Table 4-4 Register Operational Summary

Operational Description	Command Words	Bits
Fully Nested Mode	OCW-Default	—
Non-specific EOI Command	OCW2	EOI
Specific EOI Command	OCW2	SL, EOI, LO-L2
Automatic EOI Mode	ICW1, ICW4	IC4, AEOI
Rotate On Non-Specific EOI Command	OCW2	EOI
Rotate On Automatic EOI Mode	OCW2	R, SL, EOI
Set Priority Command	OCW2	L0-L2
Rotate On Specific EOI Command	OCW2	R, SL, EOI
Interrupt Mask Register	OCW1	M0-M7
Special Mask Mode	OCW3	ESMM, SMM
Level Triggered Mode	ICW1	LTIM
Edge Triggered Mode	ICW1	LTIM
Read Register Command, IRR	OCW3	RR, RIS
Read Register Command, ISR	OCW3	RR, RIS
Red IMR	IMR	M0-M7
Poll Command	OCW3	P
Special Fully Nested Mode	ICW2, ICW4	IC4, SFNM

5

5.0 PROGRAMMABLE INTERVAL TIMER

5.1 Functional Description

The 82380 contains four independently Programmable Interval Timers: Timer 0–3. All four timers are functionally compatible to the Intel 82C54. The first three timers (Timer 0–2) have specific functions. The fourth timer, Timer 3, is a general purpose timer. Table 5-1 depicts the functions of each timer. A brief description of each timer's function follows.

Table 5-1. Programmable Interval Timer Functions

Timer	Output	Function
0	IRQ8	Event Based IRQ8 Generator
1	TOUT1/REF#	Gen. Purpose/DRAM Refresh Req.
2	TOUT2#/IRQ3#	Gen. Purpose/Speaker Out/IRQ3#
3	TOUT3#	Gen. Purpose/IRQ0 Generator

TIMER 0— Event Based IRQ8 Generator

Timer 0 is intended to be used as an Event Counter. The output of this timer will generate an Interrupt Request 8 (IRQ8) upon a rising edge of the timer output (TOUT0). Typically, this timer is used to implement a time-of-day clock or system tick. The Timer 0 output is not available as an external signal.

TIMER 1— General Purpose/DRAM Refresh Request

The output of Timer 1, TOUT1, can be used as a general purpose timer or as a DRAM Refresh Request signal. The rising edge of this output creates a DRAM refresh request to the 82380 DRAM Refresh Controller. Upon reset, the Refresh Request function is disabled, and the output pin is the Timer 1 output.

TIMER 2—General Purpose/Speaker Out/IRQ3#

The Timer 2 output, TOUT2#, could be used to support tone generation to an external speaker. This pin is a bidirectional signal. When used as an input, a logic LOW asserted at this pin will generate an Interrupt Register 3 (IRQ3#) (see Programmable Interrupt Controller).

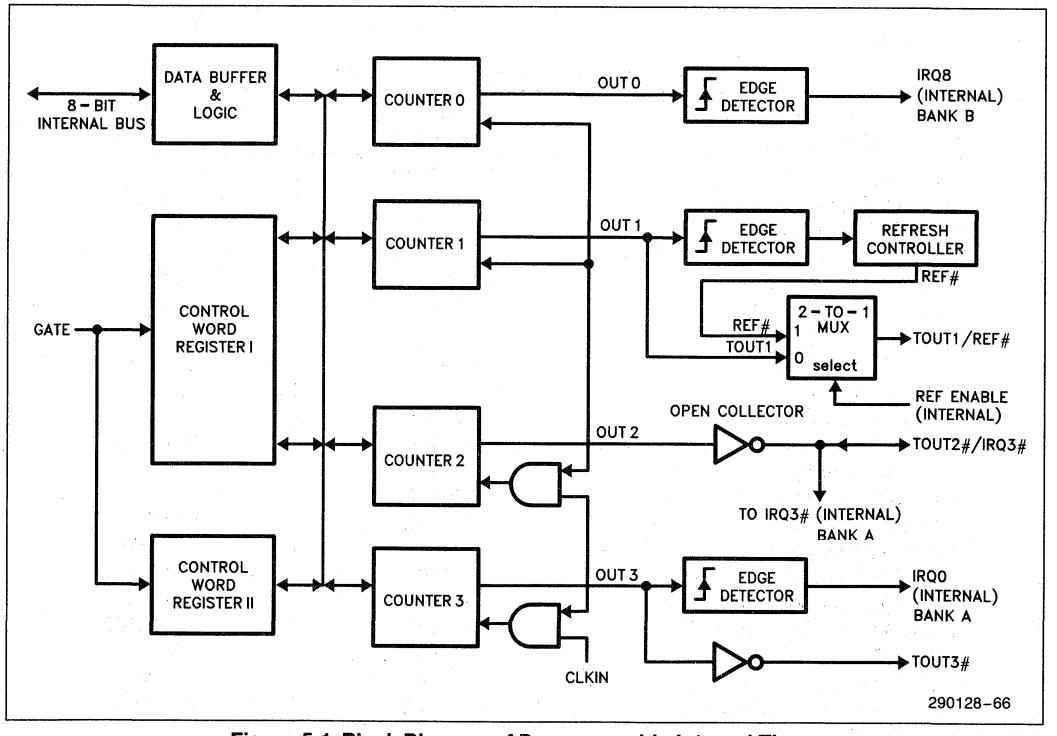


Figure 5-1. Block Diagram of Programmable Interval Timer

TIMER 3—General Purpose/Interrupt Request 0 Generator

The output of Timer 3 is fed to an edge detector and generates an Interrupt Request 0 (IRQ0) in the 82380. The inverted output of this timer (TOUT3#) is also available as an external signal for general purpose use.

5.1.1 INTERNAL ARCHITECTURE

The functional block diagram of the Programmable Interval Timer section is shown in Figure 5-1. Following is a description of each block.

DATA BUFFER & READ/WRITE LOGIC

This part of the Programmable Interval Timer is used to interface the four timers to the 82380 internal bus. The Data Buffer is for transferring commands and data between the 8-bit internal bus and the timers.

The Read/Write Logic accepts inputs from the internal bus and generates signals to control other functional blocks within the timer section.

CONTROL WORD REGISTERS I & II

The Control Word Registers are write-only registers. They are used to control the operating modes of the timers. Control Word Register I controls Timers 0, 1 and 2, and Control Word Register II controls Timer 3. Detailed description of the Control Word Registers will be included in the Register Set Overview section.

COUNTER 0, COUNTER 1, COUNTER 2, COUNTER 3

Counters 0, 1, 2, and 3 are the major parts of Timers 0, 1, 2, and 3, respectively. These four functional blocks are identical in operation, so only a single counter will be described. The internal block diagram of one counter is shown in Figure 5-2.

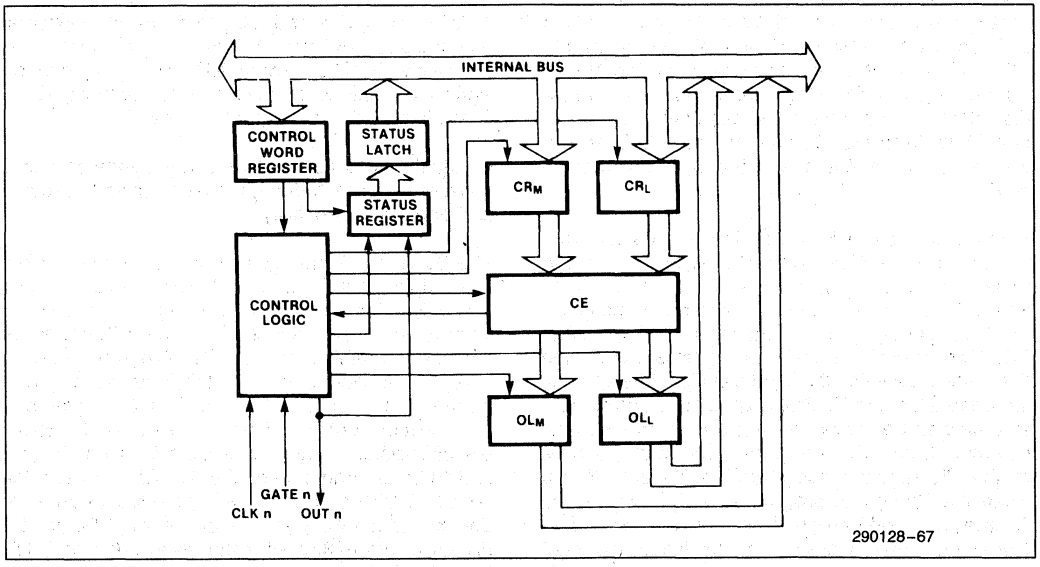


Figure 5-2. Internal Block Diagram of A Counter

The four counters share a common clock input (CLKIN), but otherwise are fully independent. Each counter is programmable to operate in a different Mode.

Although the Control Word Register is shown in the Figure 5-2, it is not part of the counter itself. Its programmed contents are used to control the operations of the counters.

The Status Register, when latched, contains the current contents of the Control Word Register and status of the output and Null Count Flag (see Read Back Command).

The Counting Element (CE) is the actual counter. It is a 16-bit presettable synchronous down counter.

The Output Latches (OL) contain two 8-bit latches (OLM and OLL). Normally, these latches 'follow' the content of the CE. OLM contains the most significant byte of the counter and OLL contains the least significant byte. If the Counter Latch Command is sent to the counter, OL will latch the present count until read by the 80386 and then return to follow the CE. One latch at a time is enabled by the timer's Control Logic to drive the internal bus. This is how the 16-bit Counter communicates over the 8-bit internal bus. Note that CE cannot be read. Whenever the count is read, it is one of the OL's that is being read.

When a new count is written into the counter, the value will be stored in the Count Registers (CR), and transferred to CE. The transferring of the contents from CR's to CE is defined as 'loading' of the counter. The Count Register contains two 8-bit registers: CRM (which contains the most significant byte) and CRL (which contains the least significant byte). Similar to the OL's, the Control Logic allows one register at a time to be loaded from the 8-bit internal bus. However, both bytes are transferred from the CR's to the CE simultaneously. Both CR's are cleared when the Counter is programmed. This way, if the Counter has been programmed for one byte count (either the most significant or the least significant byte only), the other byte will be zero. Note that CE cannot be written into directly. Whenever a count is written, it is the CR that is being written.

As shown in the diagram, the Control Logic consists of three signals: CLKIN, GATE, and OUT. CLKIN and GATE will be discussed in detail in the section that follows. OUT is the internal output of the counter. The external outputs of some timers (TOUT) are the inverted version of OUT (see TOUT1, TOUT2#, TOUT3#). The state of OUT depends on the mode of operation of the timer.

5.2 Interface Signals

5.2.1 CLKIN

CLKIN is an input signal used by all four timers for internal timing reference. This signal can be independent of the 82380 system clock, CLK2. In the following discussion, each 'CLK Pulse' is defined as the time period between a rising edge and a falling edge, in that order, of CLKIN.

During the rising edge of CLKIN, the state of GATE is sampled. All new counts are loaded and counters are decremented on the falling edge of CLKIN.

Please note that there are restrictions on the CLKIN signal during WRITE cycles to the 82380 timer unit. Refer to the appendix of this data manual for details on this issue.

5.2.2 TOUT1, TOUT2#, TOUT3#

TOUT1, TOUT2# and TOUT3# are the external output signals of Timer 1, Timer 2 and Timer 3, respectively. TOUT2# and TOUT3# are the inverted signals of their respective counter outputs, OUT. There is no external output for Timer 0.

If Timer 2 is to be used as a tone generator of a speaker, external buffering must be used to provide sufficient drive capability.

The Outputs of Timer 2 and 3 are dual function pins. The output pin of Timer 2 (TOUT2#/IRQ3#), which is a bidirectional open-collector signal, can also be used as interrupt request input. When the interrupt function is enabled (through the Programmable Interrupt Controller), a LOW on this input will generate an Interrupt Request 3# to the 82380 Programmable Interrupt Controller. This pin has a weak internal pull-up resistor. To use the IRQ3# function, Timer 2 should be programmed so that OUT2 is LOW. Additionally, OUT3 of Timer 3 is connected to an edge detector which will generate an Interrupt Request 0 (IRQ0) to the 82380 after the rising edge of OUT3 (see Figure 5-1).

5.2.3 GATE

GATE is not an externally controllable signal. Rather, it can be software controlled with the Internal Control Port. The state of GATE is always sampled on the rising edge of CLKIN. Depending on the mode of operation, GATE is used to enable/disable counting or trigger the start of an operation.

For Timer 0 and 1, GATE is always enabled (HIGH). For Timer 2 and 3, GATE is connected to Bit 0 and

6, respectively, of an Internal Control Port (at address 61H) of the 82380. After a hardware reset, the state of GATE of Timer 2 and 3 is disabled (LOW).

5.3 Modes of Operation

Each timer can be independently programmed to operate in one of six different modes. Timers are programmed by writing a Control Word into the control Word Register followed by an Initial Count (see Programming).

The following are defined for use in describing the different modes of operation.

CLK Pulse—A rising edge, then a falling edge, in that order of CLKIN.
Trigger—A rising edge of a timer's GATE input.
Timer/Counter Loading—The transfer of a count from Count Register (CR) to Count Element (CE).

Note that figures 5-3 through 5-8 show the logical outputs of the timer units, OUT_x. This signal polarity does not reflect that of the $TOUT_x$ signals. See the first paragraph of Section 5.2.2.

5.3.1 MODE 0—INTERRUPT ON TERMINAL COUNT

Mode 0 is typically used for event counting. After the Control Word is written, OUT is initially LOW, and will remain LOW until the counter reaches zero. OUT then goes HIGH and remains HIGH until a new count or a new Mode 0 Control Word is written into the counter.

In this mode, GATE = HIGH enables counting; GATE = LOW disables counting. However, GATE has no effect on OUT.

After the Control Word and initial count are written to a timer, the initial count will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not go HIGH until N + 1 CLK pulses after the initial count is written.

If a new count is written to the timer, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1. Writing the first byte disables counting, OUT is set LOW immediately (i.e., no CLK pulse required).
2. Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the counting sequence to be synchronized by software. Again, OUT does not go HIGH until N + 1 CLK pulses after the new count of N is written.

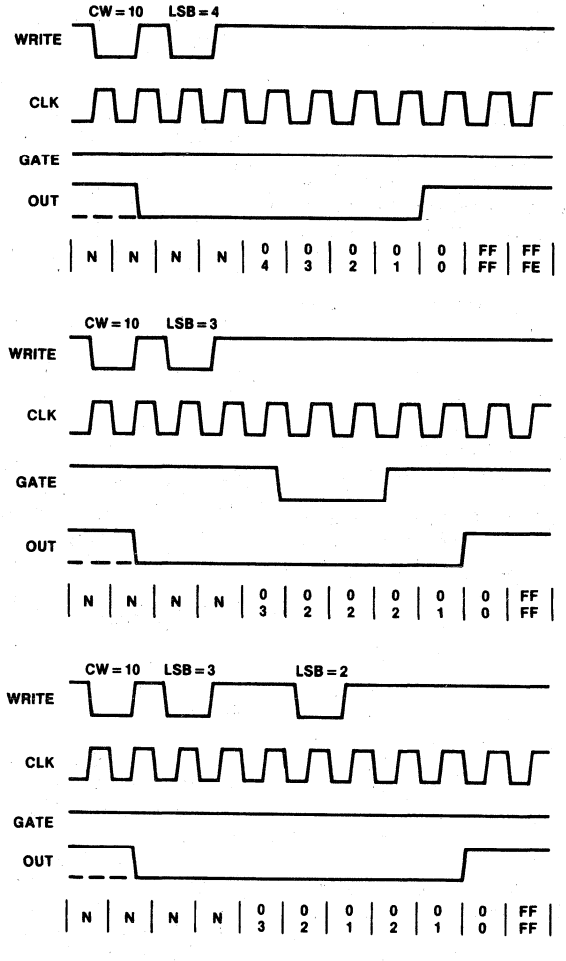
If an initial count is written while GATE is LOW, the counter will be loaded on the next CLK pulse. When GATE goes HIGH, OUT will go HIGH N CLK pulses later; no CLK pulse is needed to load the counter as this has already been done.

5.3.2 MODE 1—GATE RETRIGGERABLE ONE-SHOT

In this mode, OUT will be initially HIGH. OUT will go LOW on the CLK pulse following a trigger to start the one-shot operation. The OUT signal will then remain LOW until the timer reaches zero. At this point, OUT will stay HIGH until the next trigger comes in. Since the state of GATE signals of Timer 0 and 1 are internally set to HIGH.

After writing the Control Word and initial count, the timer is considered 'armed'. A trigger results in loading the timer and setting OUT LOW on the next CLK pulse. Therefore, an initial count of N will result in a one-shot pulse width of N CLK cycles. Note that this one-shot operation is retriggerable; i.e., OUT will remain LOW for N CLK pulses after every trigger. The one-shot operation can be repeated without rewriting the same count into the timer.

If a new count is written to the timer during a one-shot operation, the current one-shot pulse width will not be affected until the timer is retriggered. This is because loading of the new count to CE will occur only when the one-shot is triggered.



290128-68

NOTES:
The following conventions apply to all mode timing diagrams.
1. Counters are programmed for binary (not BCD) counting and for reading/writing least significant byte (LSB) only.
2. The counter is always selected ($\overline{CS}$ always low).
3. CW stands for "Control Word"; CW = 10 means a control word of 10, Hex is written to the counter.
4. LSB stands for "least significant byte" of count.
5. Numbers below diagrams are count values.
   The lower number is the least significant byte.
   The upper number is the most significant byte. Since the counter is programmed to read/write LSB only, the most significant byte cannot be read.
   N stands for an undefined count.
   Vertical lines show transitions between count values.

Figure 5-3. Mode 0

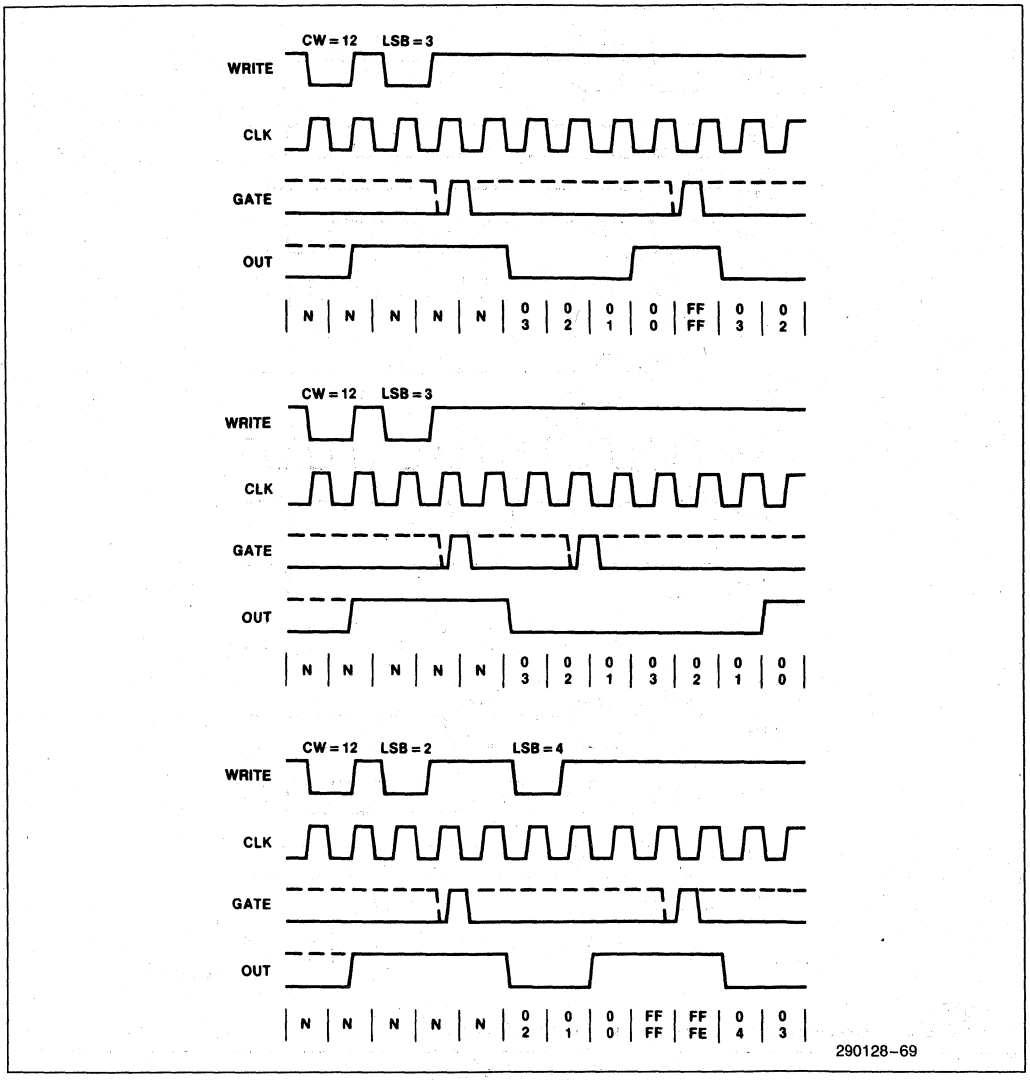
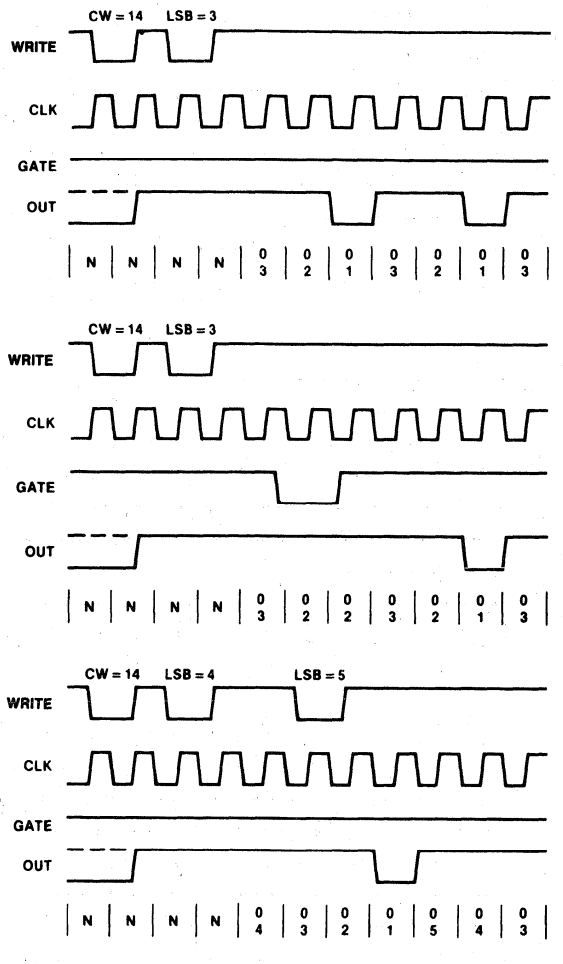


Figure 5-4. Mode 1

5.3.3 MODE 2—RATE GENERATOR

This mode is a divide-by-N counter. It is typically used to generate a Real Time Clock interrupt. OUT will initially be HIGH. When the initial count has decremented to 1, OUT goes LOW for one CLK pulse, then OUT goes HIGH again. Then the timer reloads the initial count and the process is repeated. In other words, this mode is periodic since the same sequence is repeated itself indefinitely. For an initial count of N, the sequence repeats every N CLK cycles.

Similar to Mode 0, GATE = HIGH enables counting, where GATE = LOW disables counting. If GATE goes LOW during an output pulse (LOW), OUT is set HIGH immediately. A trigger (rising edge on GATE) will reload the timer with the initial count on the next CLK pulse. Then, OUT will go LOW (for one CLK pulse) N CLK pulses after the new trigger. Thus, GATE can be used to synchronize the timer.



NOTE:
A GATE transition should not occur one clock prior to terminal count.

Figure 5-5. Mode 2

After writing a Control Word and initial count, the timer will be loaded on the next CLK pulse. OUT goes LOW (for the CLK pulse) N CLK pulses after the initial count is written. This is another way the timer may be synchronized by software.

Writing a new count while counting does not affect the current counting sequence because the new count will not be loaded until the end of the current counting cycle. If a trigger is received after writing a new count but before the end of the current period, the timer will be loaded with the new count on the next CLK pulse after the trigger, and counting will continue with the new count.

5.3.4 MODE 3—SQUARE WAVE GENERATOR

Mode 3 is typically used for Baud Rate generation. Functionally, this mode is similar to Mode 2 except for the duty cycle of OUT. In this mode, OUT will be initially HIGH. When half of the initial count has expired, OUT goes low for the remainder of the count.

The counting sequence will be repeated, thus this mode is also periodic. Note that an initial count of N results in a square wave with a period of N CLK pulses.

The GATE input can be used to synchronize the timer. GATE = HIGH enables counting; GATE = LOW disables counting. If GATE goes LOW while OUT is LOW, OUT is set HIGH immediately (i.e., no CLK pulse is required). A trigger reloads the timer with the initial count on the next CLK pulse.

After writing a Control Word and initial count, the timer will be loaded on the next CLK pulse. This allows the timer to be synchronized by software.

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current half-cycle of the square wave, the timer will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current half-cycle.

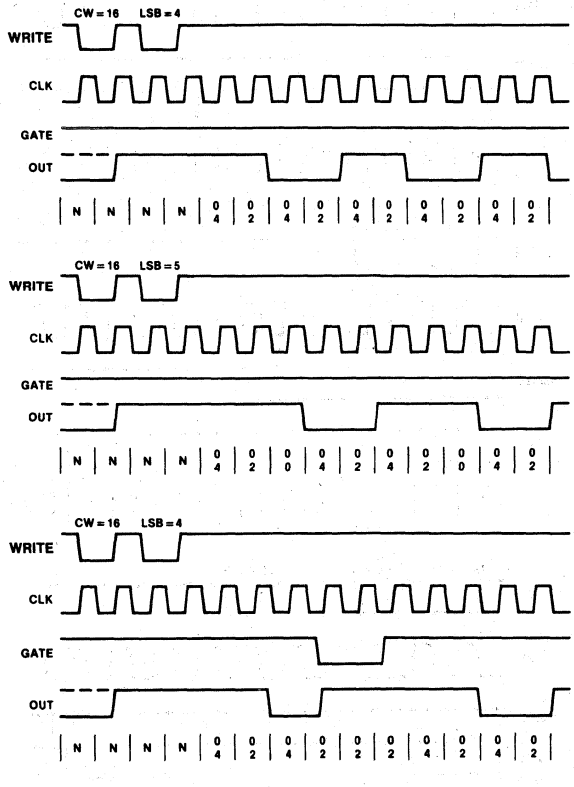
There is a slight difference in operation depending on whether the initial count is EVEN or ODD. The following description is to show exactly how this mode is implemented.

EVEN COUNTS:

OUT is initially HIGH. The initial count is loaded on one CLK pulse and is decremented by two on succeeding CLK pulses. When the count expires (decremented to 2), OUT changes to LOW and the timer is reloaded with the initial count. The above process is repeated indefinitely.

ODD COUNTS:

OUT is initially HIGH. The initial count minus one (which is an even number) is loaded on one CLK



290128-71

NOTE:
A-GATE transition should not occur one clock prior to terminal count.

Figure 5-6. Mode 3

pulse and is decremented by two on succeeding CLK pulses. One CLK pulse after the count expires (decremented to 2), OUT goes LOW and the timer is loaded with the initial count minus one again. Succeeding CLK pulses decrement the count by two. When the count expires, OUT goes HIGH immediately and the timer is reloaded with the initial count minus one. The above process is repeated indefinitely. So for ODD counts, OUT will be HIGH for (N + 1)/2 counts and LOW for (N − 1)/2 counts.

5.3.5 MODE 4—INITIAL COUNT TRIGGERED STROBE

This mode allows a strobe pulse to be generated by writing an initial count to the timer. Initially, OUT will be HIGH. When a new initial count is written into the timer, the counting sequence will begin. When the initial count expires (decremented to 1), OUT will go LOW for one CLK pulse and then go HIGH again.

Again, GATE = HIGH enables counting while GATE = LOW disables counting. GATE has no effect on OUT.

After writing the Control Word and initial count, the timer will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe LOW until N + 1 CLK pulses after initial count is written.

If a new count is written during counting, it will be loaded in the next CLK pulse and counting will continue from the new count.

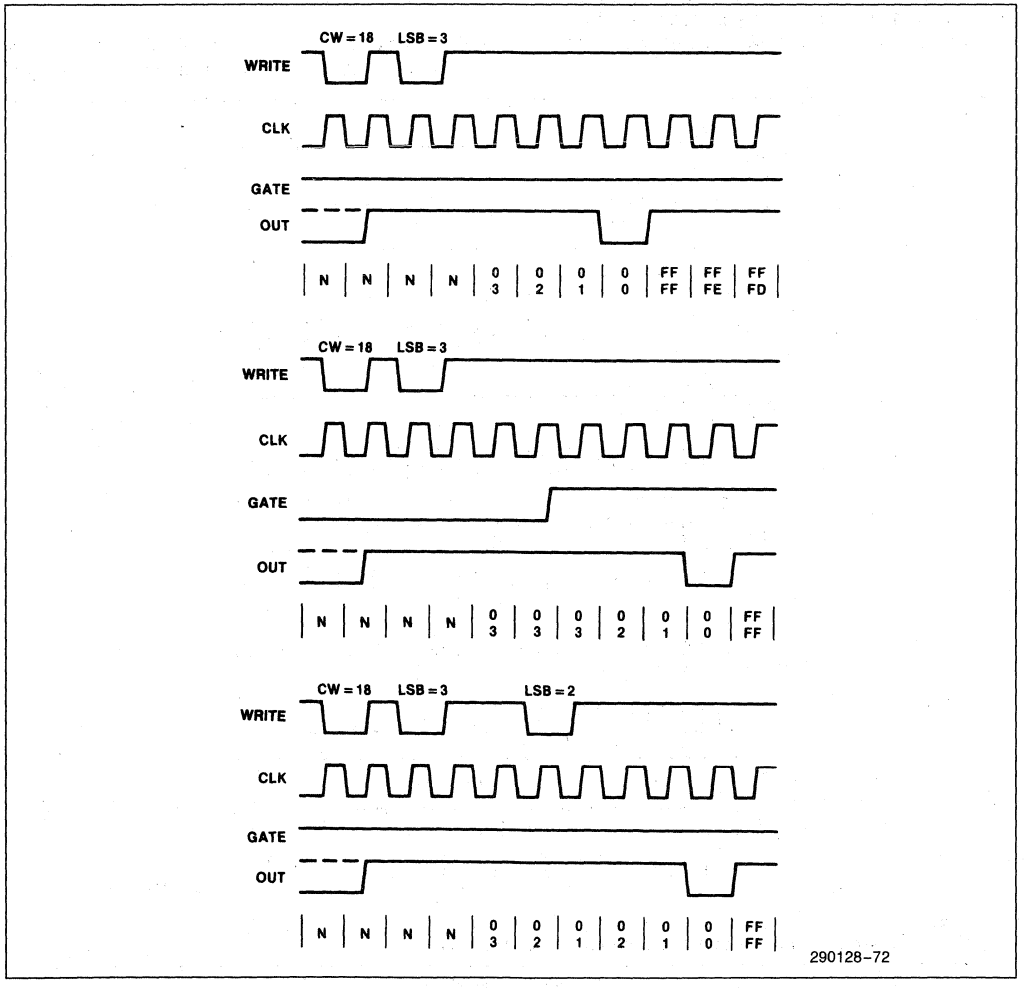


Figure 5-7. Mode 4

If a two-byte count is written, the following will occur:

1. Writing the first byte has no effect on counting.
2. Writing the second byte allows the new count to be loaded on the next CLK pulse.

OUT will strobe LOW N + 1 CLK pulses after the new count of N is written. Therefore, when the strobe pulse will occur after a trigger depends on the value of the initial count loaded.

5.3.6 MODE 5—GATE RETRIGGERABLE STROBE

Mode 5 is very similar to Mode 4 except the count sequence is triggered by the GATE signal instead of by writing an initial count. Initially, OUT will be HIGH. Counting is triggered by a rising edge of GATE. When the initial count has expired (decremented to 1), OUT will go LOW for one CLK pulse and then go HIGH again.

After loading the Control Word and initial count, the Count Element will not be loaded until the CLK pulse after a trigger. This CLK pulse does not decrement the count. Therefore, for an initial count of N, OUT does not strobe LOW until N + 1 CLK pulses after a trigger.

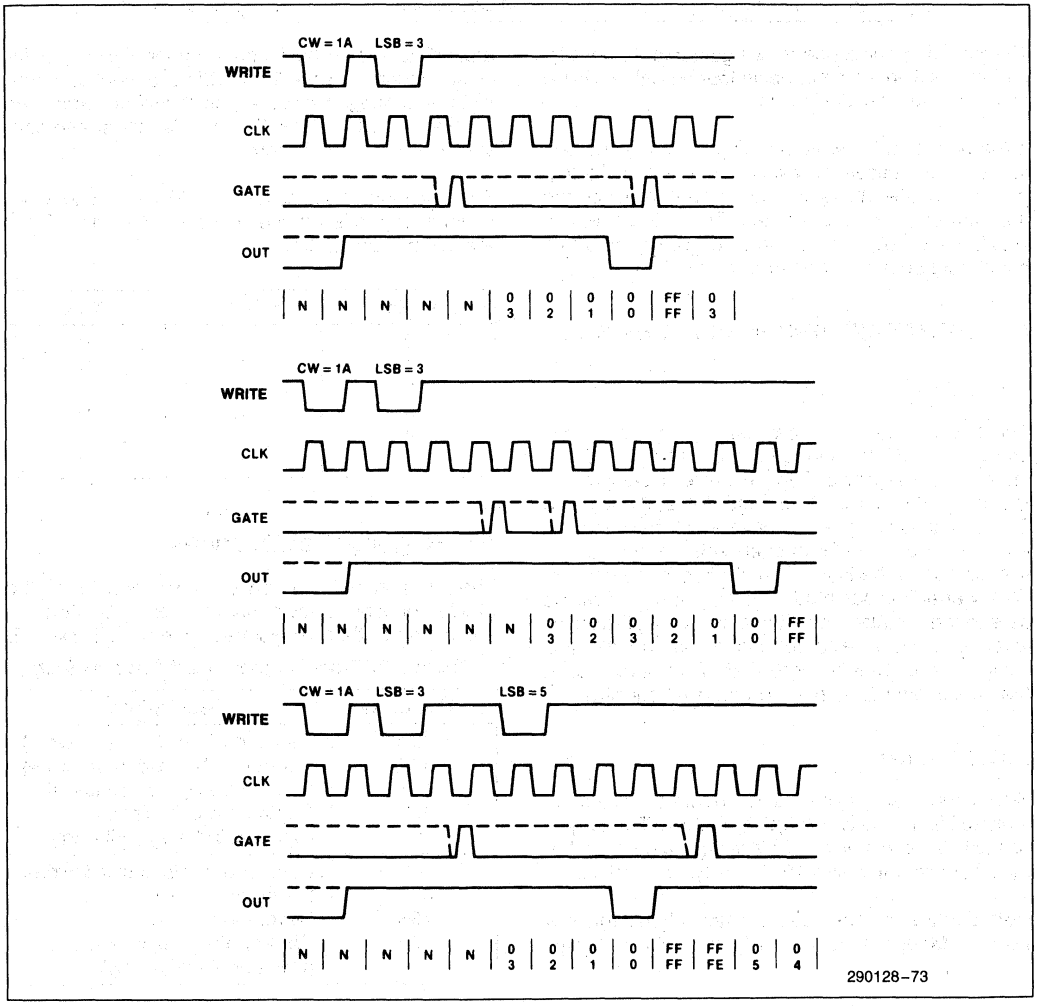


Figure 5-8. Mode 5

SUMMARY OF GATE OPERATIONS

Mode	GATE LOW or Going LOW	GATE Rising	GATE HIGH
0	Disable Count	No Effect	Enable Count
1	No Effect	1. Initiate Count 2. Reset Output After Next Clock	No Effect
2	1. Disable Count 2. Sets Output HIGH Immediately	Initiate Count	Enable Count
3	1. Disable Count 2. Sets Output HIGH Immediately	Initiate Count	Enable Count
4	Disable Count	No Effect	Enable Count
5	No Effect	Initiate Count	No Effect

The counting sequence is retriggerable. Every trigger will result in the timer being loaded with the initial count on the next CLK pulse.

If the new count is written during counting, the current counting sequence will not be affected. If a trigger occurs after the new count is written but before the current count expires, the timer will be loaded with the new count on the next CLK pulse and a new count sequence will start from there.

5.3.7 OPERATION COMMON TO ALL MODES

5.3.7.1 GATE

The GATE input is always sampled on the rising edge of CLKIN. In Modes 0, 2, 3 and 4, the GATE input is level sensitive. The logic level is sampled on the rising edge of CLKIN. In Modes 1, 2, 3 and 5, the GATE input is rising edge sensitive. In these modes, a rising edge of GATE (trigger) sets an edge sensitive flip-flop in the timer. The flip-flop is reset immediately after it is sampled. This way, a trigger will be detected no matter when it occurs; i.e., a HIGH logic level does not have to be maintained until the next rising edge of CLKIN. Note that in Modes 2 and 3, the GATE input is both edge and level sensitive.

5.3.7.2 Counter

New counts are loaded and counters are decremented on the falling edge of CLKIN. The largest possible initial count is 0. This is equivalent to 2**16 for binary counting and 10**4 for BCD counting.

Note that the counter does not stop when it reaches zero. In Modes 0, 1, 4, and 5, the counter 'wraps around' to the highest count: either FFFF Hex for binary counting or 9999 for BCD counting, and continues counting. Modes 2 and 3 are periodic. The counter reloads itself with the initial count and continues counting from there.

The minimum and maximum initial count in each counter depends on the mode of operation. They are summarized below.

Mode	Min	Max
0	1	0
1	1	0
2	2	0
3	2	0
4	1	0
5	1	0

5.4 Register Set Overview

The Programmable Interval Timer module of the 82380 contains a set of six registers. The port address map of these registers is shown in Table 5-2.

Table 5-2. Timer Register Port Address Map

Port Address	Description
40H	Counter 0 Register (read/write)
41H	Counter 1 Register (read/write)
42H	Counter 2 Register (read/write)
43H	Control Word Register I (Counter 0, 1 & 2) (write-only)
44H	Counter 3 Register (read/write)
45H	Reserved
46H	Reserved
47H	Control Word Register II (Counter 3) (write-only)

5.4.1 COUNTER 0, 1, 2, 3 REGISTERS

These four 8-bit registers are functionally identical. They are used to write the initial count value into the respective timer. Also, they can be used to read the latched count value of a timer. Since they are 8-bit registers, reading and writing of the 16-bit initial count must follow the count format specified in the Control Word Registers; i.e., least significant byte only, most significant byte only, or least significant byte then most significant byte (see Programming).

5.4.2 CONTROL WORD REGISTER I & II

There are two Control Word Registers associated with the Timer section. One of the two registers (Control Word Register I) is used to control the operations of Counters 0, 1, and 2 and the other (Control Word Register II) is for Counter 3. The major functions of both Control Word Registers are listed below:

— Select the timer to be programmed.

— Define which mode the selected timer is to operate in.

— Define the count sequence; i.e., if the selected timer is to count as a Binary Counter or a Binary Coded Decimal (BCD) Counter.

— Select the byte access sequence during timer read/write operations; i.e., least significant byte only, most significant byte only, or least significant byte first, then most significant byte.

Also, the Control Word Registers can be programmed to perform a Counter Latch Command or a Read Back Command which will be described later.

5.5 Programming

5.5.1 INITIALIZATION

Upon power-up or reset, the state of all timers is undefined. The mode, count value, and output of all timers are random. From this point on, how each timer operates is determined solely by how it is programmed. Each timer must be programmed before it can be used. Since the outputs of some timers can generate interrupt signals to the 82380, all timers should be initialized to a known state.

Timers are programmed by writing a Control Word into their respective Control Word Registers. Then, an Initial Count can be written into the corresponding Count Register. In general, the programming procedure is very flexible. Only two conventions need to be remembered:

1. For each timer, the Control Word must be written before the initial count is written.
2. The 16-bit initial count must follow the count format specified in the Control Word (least significant byte only, most significant byte only, or least significant byte first, followed by most significant byte).

Since the two Control Word Registers and the four Counter Registers have separate addresses, and each timer can be individually selected by the appropriate Control Word Register, no special instruction sequence is required. Any programming sequence that follows the conventions above is acceptable.

A new initial count may be written to a timer at any time without affecting the timer's programmed mode in any way. Count sequence will be affected as described in the Modes of Operation section. Note that the new count must follow the programmed count format.

If a timer is previously programmed to read/write two-byte counts, the following precaution applies. A program must not transfer control between writing the first and second byte to another routine which also writes into the same timer. Otherwise, the read/write will result in incorrect count.

Whenever a Control Word is written to a timer, all control logic for that timer(s) is immediately reset (i.e., no CLK pulse is required). Also, the corresponding output pin, TOUT(#), goes to a known initial state.

5.5.2 READ OPERATION

Three methods are available to read the current count as well as the status of each timer. They are: Read Counter Registers, Counter Latch Command and Read Back Command. Following is a description of these methods.

READ COUNTER REGISTERS

The current count of a timer can be read by performing a read operation on the corresponding Counter Register. The only restriction of this read operation is that the CLKIN of the timers must be inhibited by

using external logic. Otherwise, the count may be in the process of changing when it is read, giving an undefined result. Note that since all four timers are sharing the same CLKIN signal, inhibiting CLKIN to read a timer will unavoidably disable the other timers also. This may prove to be impractical. Therefore, it is suggested that either the Counter Latch Command or the Read Back Command be used to read the current count of a timer.

Another alternative is to temporarily disable a timer before reading its Counter Register by using the GATE input. Depending on the mode of operation, GATE = LOW will disable the counting operation. However, this option is available on Timer 2 and 3 only, since the GATE signals of the other two timers are internally enabled all the time.

COUNTER LATCH COMMAND

A Counter Latch Command will be executed whenever a special Control Word is written into a Control Word Register. Two bits written into the Control Word Register distinguish this command from a 'regular' Control Word (see Register Bit Definition). Also, two other bits in the Control Word will select which counter is to be latched.

Upon execution of this command, the selected counter's Output Latch (OL) latches the count at the time the Counter Latch Command is received. This count is held in the latch until it is read by the 80386, or until the timer is reprogrammed. The count is then unlatched automatically and the OL returns to 'following' the Counting Element (CE). This allows reading the contents of the counters 'on the fly' without affecting counting in progress. Multiple Counter Latch Commands may be used to latch more than one counter. Each latched count is held until it is read. Counter Latch Commands do not affect the programmed mode of the timer in any way.

If a counter is latched, and at some time later, it is latched again before the prior latched count is read, the second Counter Latch Command is ignored. The count read will then be the count at the time the first command was issued.

In any event, the latched count must be read according to the programmed format. Specifically, if the timer is programmed for two-byte counts, two bytes must be read. However, the two bytes do not have to be read right after the other. Read/write or programming operations of other timers may be performed between them.

Another feature of this Counter Latch Command is that read and write operations of the same timer may be interleaved. For example, if the timer is programmed for two-byte counts, the following sequence is valid.

1. Read least significant byte.
2. Write new least significant byte.
3. Read most significant byte.
4. Write new most significant byte.

If a timer is programmed to read/write two-byte counts, the following precaution applies. A program must not transfer control between reading the first and second byte to another routine which also reads from that same timer. Otherwise, an incorrect count will be read.

READ BACK COMMAND

The Read Back Command is another special Command Word operation which allows the user to read the current count value and/or the status of the selected timer(s). Like the Counter Latch Command, two bits in the Command Word identify this as a Read Back Command (see Register Bit Definition).

The Read Back Command may be used to latch multiple counter Output Latches (OL's) by selecting more than one timer within a Command Word. This single command is functionally equivalent to several Counter Latch Commands, one for each counter to be latched. Each counter's latched count will be held until it is read by the 80386 or until the timer is reprogrammed. The counter is automatically unlatched when read, but other counters remain latched until they are read. If multiple Read Back commands are issued to the same timer without reading the count, all but the first are ignored; i.e., the count read will correspond to the very first Read Back Command issued.

As mentioned previously, the Read Back Command may also be used to latch status information of the selected timer(s). When this function is enabled, the status of a timer can be read from the Counter Register after the Read Back Command is issued. The status information of a timer includes the following:

1. Mode of timer:

   This allows the user to check the mode of operation of the timer last programmed.

2. State of TOUT pin of the timer:

   This allows the user to monitor the counter's output pin via software, possibly eliminating some hardware from a system.

3. Null Count/Count available:

   The Null Count Bit in the status byte indicates if the last count written to the Count Register (CR) has been loaded into the Counting Element (CE). The exact time this happens depends on the mode of the timer and is described in the Programming section. Until the count is loaded into the Counting Element (CE), it cannot be read from the timer. If the count is latched or read before this occurs, the count value will not reflect the new count just written.

If multiple status latch operations of the timer(s) are performed without reading the status, all but the first command are ignored; i.e., the status read in will correspond to the first Read Back Command issued.

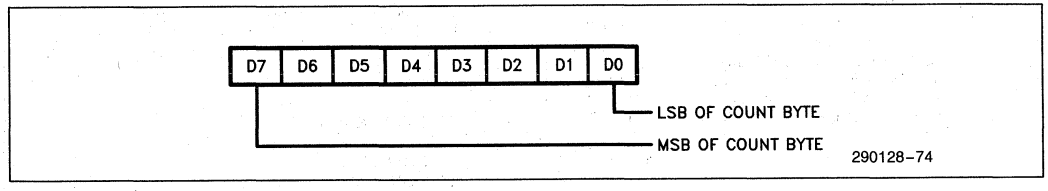
Both the current count and status of the selected timer(s) may be latched simultaneously by enabling both functions in a single Read Back Command. This is functionally the same as issuing two separate Read Back Commands at once. Once again, if multiple read commands are issued to latch both the count and status of a timer, all but the first command will be ignored.

If both count and status of a timer are latched, the first read operation of that timer will return the latched status, regardless of which was latched first. The next one or two (if two count bytes are to be read) read operations return the latched count. Note that subsequent read operations on the Counter Register will return the unlatched count (like the first read method discussed).

5.6 Register Bit Definitions

COUNTER 0, 1, 2, 3 REGISTER (READ/WRITE)

Port Address	Description
40H	Counter 0 Register (read/write)
41H	Counter 1 Register (read/write)
42H	Counter 2 Register (read/write)
44H	Counter 3 Register (read/write)
45H	Reserved
46H	Reserved

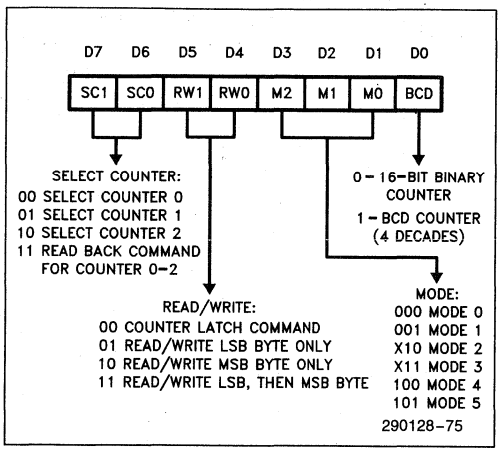


Note that these 8-bit registers are for writing and reading of one byte of the 16-bit count value, either the most significant or the least significant byte.

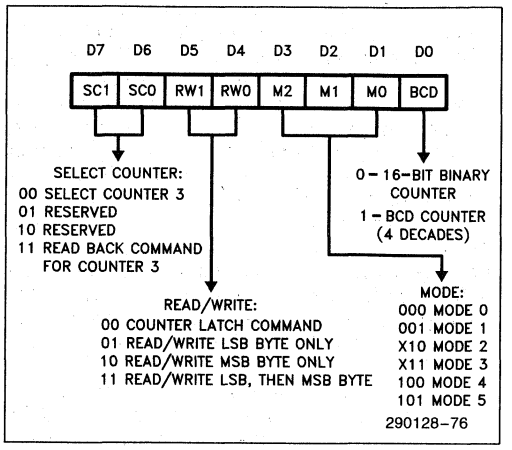
CONTROL WORD REGISTER I & II (WRITE-ONLY)

Port Address	Description
43H	Control Word Register I (Counter 0, 1, 2) (write-only)
47H	Control Word Register II (Counter 3) (write-only)

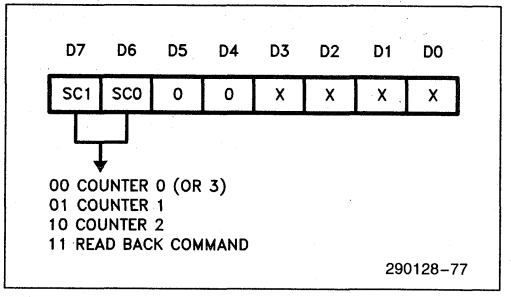
CONTROL WORD REGISTER I



CONTROL WORD REGISTER II



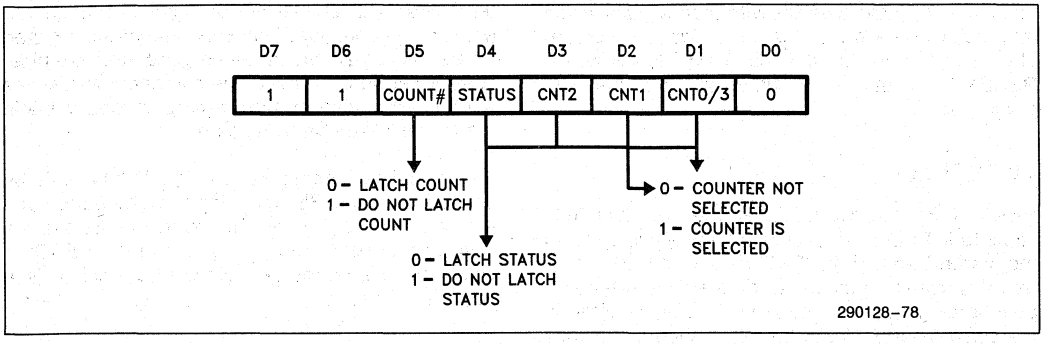
COUNTER LATCH COMMAND FORMAT (Write to Control Word Register)



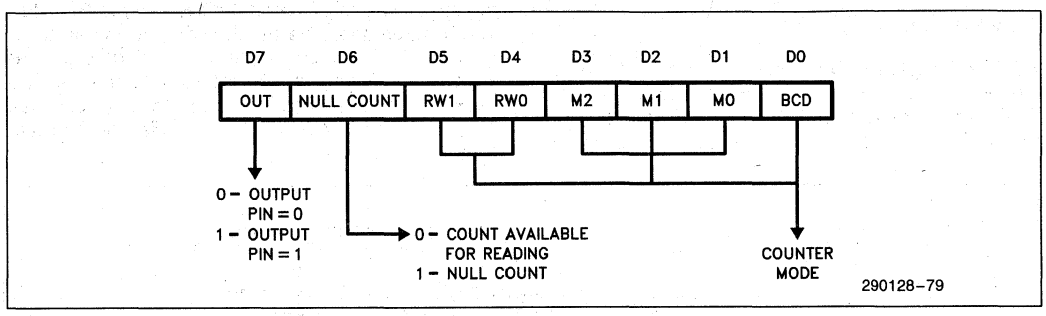
Mode	Timer				Gate Trigger		
	0	1	2	3	Edge	Level	
0						X	Interrupt on Terminal Count
1	NA	NA	①	①	X		Gate Retriggerable One Shot
2					X	X	Rate Generator
3					X	X	Square Wave Generator
4						X	Initial Count Triggered Strobe
5	NA	NA	①	①	X		Gate Retriggerable Strobe

① = Must use Port 61 to generate ⌐ edge.
NA = Not Applicable

READ BACK COMMAND FORMAT
(Write to Control Word Register)



STATUS FORMAT
(Returned from Read Back Command)



5

6.0 WAIT STATE GENERATOR

6.1 Functional Description

The 82380 contains a programmable Wait State Generator which can generate a pre-programmed number of wait states during both CPU and DMA initiated bus cycles. This Wait State Generator is capable of generating 1 to 16 wait states in non-pipelined mode, and 0 to 15 wait states in pipelined mode. Depending on the bus cycle type and the two Wait State Control inputs (WSC 0–1), a pre-programmed number of wait states in the selected Wait State Register will be generated.

The Wait State Generator can also be disabled to allow the use of devices capable of generating their own READY# signals. Figure 6-1 is a block diagram of the Wait State Generator.

6.2 Interface Signals

The following describes the interface signals which affect the operation of the Wait State Generator. The READY#, WSC0 and WSC1 signals are inputs. READYO# is the ready output signal to the host processor.

6.2.1 READY#

READY# is an active LOW input signal which indicates to the 82380 the completion of a bus cycle. In the Master mode (e.g., 82380 initiated DMA transfer), this signal is monitored to determine whether a peripheral or memory needs wait states inserted in the current bus cycle. In the Slave mode, it is used (together with the ADS# signal) to trace CPU bus cycles to determine if the current cycle is pipelined.

6.2.2 READYO#

READYO# (Ready Out#) is an active LOW output signal and is the output of the Wait State Generator. The number of wait states generated depends on the WSC(0-1) inputs. Note that special cases are handled for access to the 82380 internal registers and for the Refresh cycles. For 82380 internal register access, READYO# will be delayed to take into account the command recovery time of the register. One or more wait states will be generated in a pipelined cycle. During refresh, the number of wait states will be determined by the preprogrammed value in the Refresh Wait State Register.

In the simplest configuration, READYO# can be connected to the READY# input of the 82380 and the 80386 CPU. This is, however, not always the case. If external circuitry is to control the READY# inputs as well, additional logic will be required (see Application Issues).

6.2.3 WSC(0-1)

These two Wait State Control inputs select one of the three pre-programmed 8-bit Wait State Registers which determines the number of wait states to be generated. The most significant half of the three Wait State Registers corresponds to memory accesses, the least significant half to I/O accesses. The combination WSC(0-1) = 11 disables the Wait State Generator.

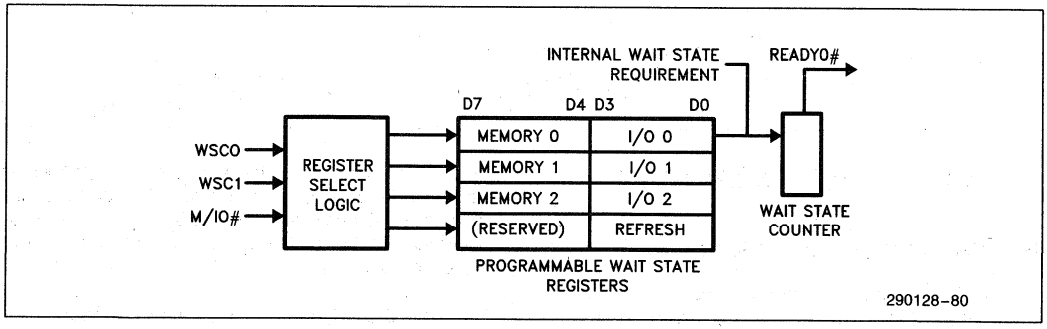


Figure 6-1. Wait State Generator Block Diagram

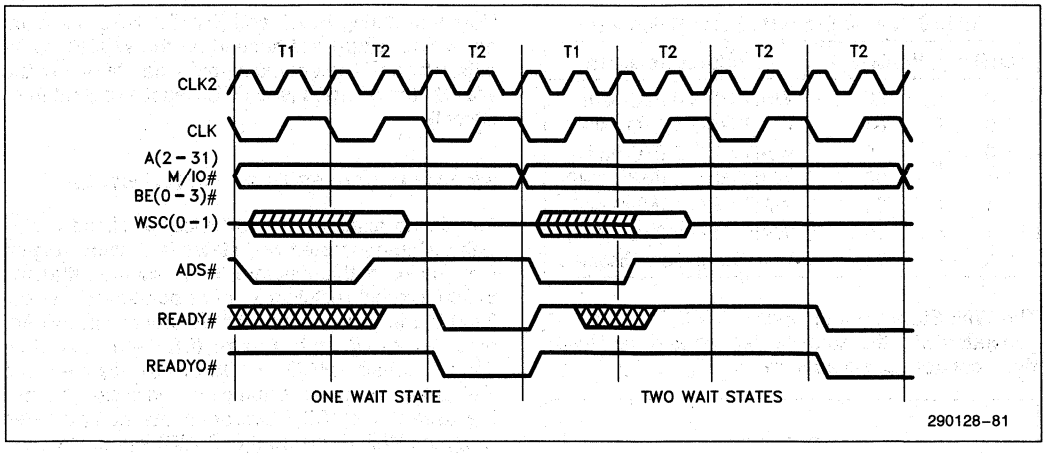


Figure 6-2. Wait States in Non-Pipelined Cycles

6.3 Bus Function

6.3.1 WAIT STATES IN NON-PIPELINED CYCLE

The timing diagram of two typical non-pipelined cycles with 82380 generated wait states is shown in Figure 6-2. In this diagram, it is assumed that the internal registers of the 82380 are not addressed. During the first T2 state of each bus cycle, the Wait State Control and the M/IO# inputs are sampled to determine which Wait State Register (if any) is selected. If the WSC inputs are active (i.e., not both are driven HIGH), the pre-programmed number of wait states corresponding to the selected Wait State Register will be requested. This is done by driving the READYO# output HIGH during the end of each T2 state.

The WSC(0–1) inputs need only be valid during the very first T2 state of each non-pipelined cycle. As a general rule, the WSC inputs are sampled on the rising edge of the next clock (82384 CLK) after the last state when ADS# (Address Status) is asserted.

The number of wait states generated depends on the type of bus cycle, and the number of wait states requested. The various combinations are discussed below.

1. Access the 82380 internal registers: 2 to 5 wait states, depending upon the specific register addressed. Some back-to-back sequences to the Interrupt Controller will require 7 wait states.
2. Interrupt Acknowledge to the 82380: 5 wait states.
3. Refresh: As programmed in the Refresh Wait State Register (see Register Set Overview). Note that if WSC(0–1) = 11, READYO# will stay inactive.
4. Other bus cycles: Depending on WSC(0–1) and M/IO# inputs, these inputs select a Wait State Register in which the number of wait states will be equal to the pre-programmed wait state count in the register plus 1. The Wait State Register selection is defined as follows (Table 6-1).

Table 6-1. Wait State Register Selection

M/IO#	WSC(1-0)	Register Selected
0	00	WAIT REG 0 (I/O half)
0	01	WAIT REG 1 (I/O half)
0	10	WAIT REG 2 (I/O half)
1	00	WAIT REG 0 (MEM half)
1	01	WAIT REG 1 (MEM half)
1	10	WAIT REG 2 (MEM half)
X	11	Wait State Gen. Disabled

The Wait State Control signals, WSC(0-1), can be generated with the address decode and the Read/Write control signals as shown in Figure 6-3.

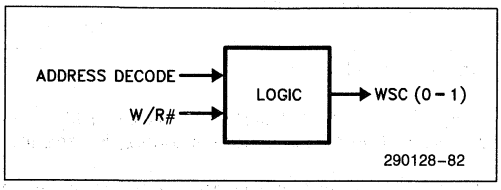


Figure 6-3. WSC(0-1) Generation

Note that during HALT and SHUTDOWN, the number of wait states will depend on the WSC(0-1) inputs, which will select the memory half of one of the Wait State Registers (see CPU Reset and Shutdown Detect).

6.3.2 WAIT STATES IN PIPELINED CYCLE

The timing diagram of two typical pipelined cycles with 82380 generated wait states is shown in Figure 6-4. Again, in this diagram, it is assumed that the 82380 internal registers are not addressed. As defined in the timing of the 80386 processor, the Address (A 2-31), Byte Enable (BE 0-3), and other control signals (M/IO#, ADS#) are asserted one T state earlier than in a non-pipelined cycle; i.e., they are asserted at T2P. Similar to the non-pipelined case, the Wait State Control (WSC) inputs are sampled in the middle of the state after the last state when the ADS# signal is asserted. Therefore, the WSC inputs should be asserted during the T1P state of each pipelined cycle (which is one T state earlier than in the non-pipelined cycle).

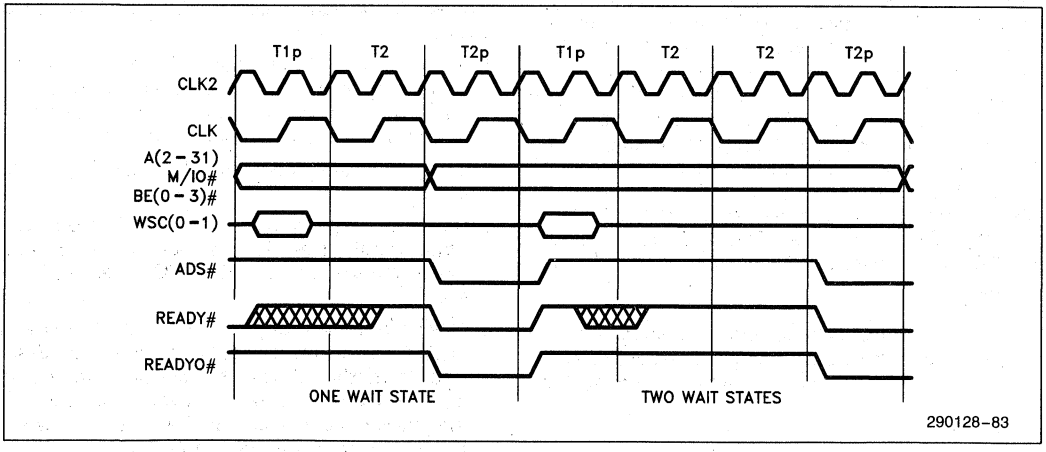


Figure 6-4. Wait State in Pipelined Cycles

The number of wait states generated in a pipelined cycle is selected in a similar manner as in the non-pipelined case discussed in the previous section. The only difference here is that the actual number of wait states generated will be one less than that of the non-pipelined cycle. This is done automatically by the Wait State Generator.

6.3.3 EXTENDING AND EARLY TERMINATING BUS CYCLE

The 82380 allows external logic to either add wait states or cause early termination of a bus cycle by controlling the READY# input to the 82380 and the host processor. A possible configuration is shown in Figure 6-5.

The EXT. RDY# (External Ready) signal of Figure 6-5 allows external devices to cause early termination of a bus cycle. When this signal is asserted LOW, the output of the circuit will also go LOW (even though the READYO# of the 82380 may still be HIGH). This output is fed to the READY# input of the 80386 and the 82380 to indicate the completion of the current bus cycle.

Similarly, the EXT. NOT READY (External Not Ready) signal is used to delay the READY# input of the processor and the 82380. As long as this signal is driven HIGH, the output of the circuit will drive the READY# input HIGH. This will effectively extend the duration of a bus cycle. However, it is important to note that if the two-level logic is not fast enough to satisfy the READY# setup time, the OR gate should be eliminated. Instead, the 82380 Wait State Generator can be disabled by driving both WSC(0-1) HIGH. In this case, the addressed memory or I/O device should activate the external READY# input whenever it is ready to terminate the current bus cycle.

Figure 6-6 and 6-7 show the timing relationships of the ready signals for the early termination and extension of the bus cycles. Section 6.7, Application Issues, contains a detailed timing analysis of the external circuit.

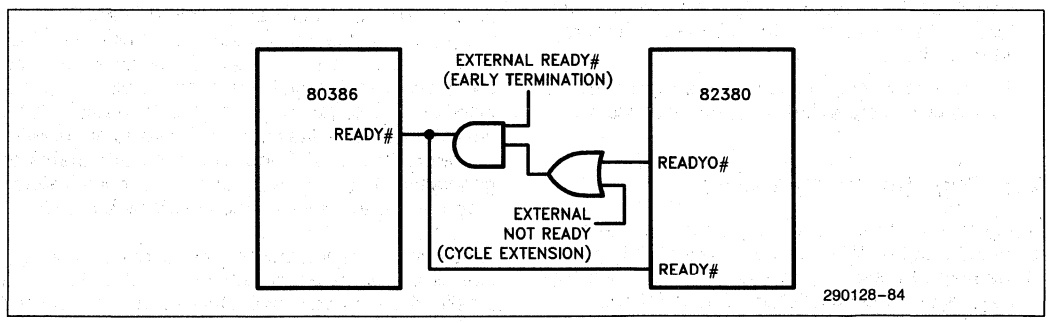


Figure 6-5. External 'READY' Control Logic

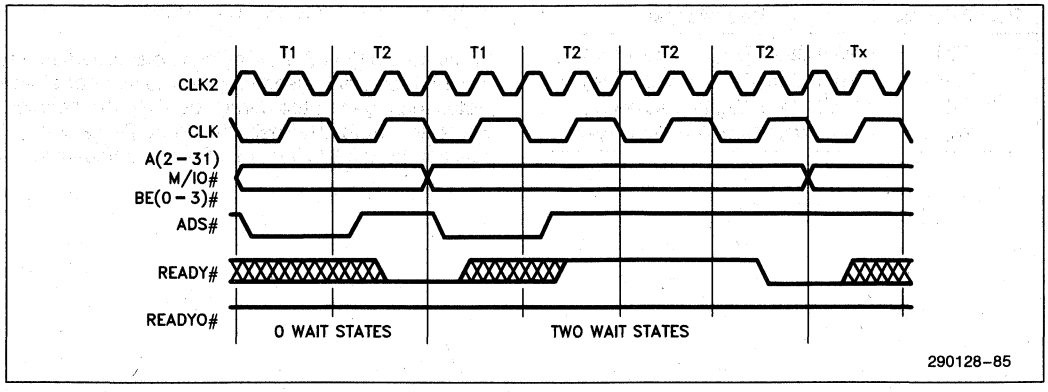


Figure 6-6. Early Termination of Bus Cycle By 'READY#'

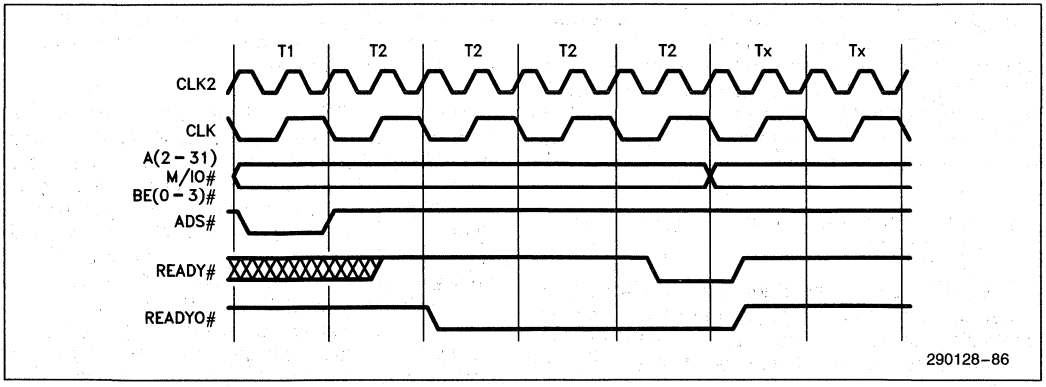


Figure 6-7. Extending Bus Cycle by 'READY#'

Due to the following implications, it should be noted that early termination of bus cycles in which 82380 internal registers are accessed is not recommended.

1. Erroneous data may be read from or written into the addressed register.
2. The 82380 must be allowed to recover either before HLDA (Hold Acknowledge) is asserted or before another bus cycle into an 82380 internal register is initiated.

   The recovery time, in bus periods, equals the remaining wait states that were avoided plus 4.

6.4 Register Set Overview

Altogether, there are four 8-bit internal registers associated with the Wait State Generator. The port address map of these registers is shown below in Table 6-2. A detailed description of each follows.

Table 6-2. Register Address Map

Port Address	Description
72H	Wait State Reg 0 (read/write)
73H	Wait State Reg 1 (read/write)
74H	Wait State Reg 2 (read/write)
75H	Ref. Wait State Reg (read/write)

WAIT STATE REGISTER 0, 1, 2

These three 8-bit read/write registers are functionally identical. They are used to store the pre-programmed wait state count. One half of each register contains the wait state count for I/O accesses while the other half contains the count for memory accesses. The total number of wait states generated will depend on the type of bus cycle. For a non-pipelined cycle, the actual number of wait states requested is equal to the wait state count plus 1. For a pipelined cycle, the number of wait states will be equal to the wait state count in the selected register. Therefore, the Wait State Generator is capable of generating 1 to 16 wait states in non-pipelined mode, and 0 to 15 wait states in pipelined mode.

Note that the minimum wait state count in each register is 0. This is equivalent to 0 wait states for a pipelined cycle and 1 wait state for a non-pipelined cycle.

REFRESH WAIT STATE REGISTER

Similar to the Wait State Registers discussed above, this 4-bit register is used to store the number of wait states to be generated during the DRAM refresh cycle. Note that the Refresh Wait State Register is not selected by the WSC inputs. It will automatically be

chosen whenever a DRAM refresh cycle occurs. If the Wait State Generator is disabled during the refresh cycle (WSC(0–1) = 11), READYO# will stay inactive and the Refresh Wait State Register is ignored.

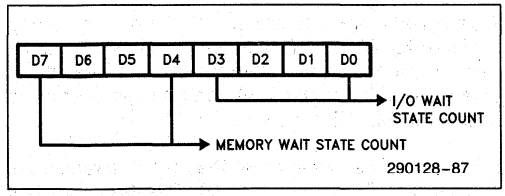
6.5 Programming

Using the Wait State Generator is relatively straightforward. No special programming sequence is required. In order to ensure the expected number of wait states will be generated when a register is selected, the registers to be used must be programmed after power-up by writing the appropriate wait state count into each register. Note that upon hardware reset, all Wait State Registers are initialized with the value FFH, giving the maximum number of wait states possible. Also, each register can be read to check the wait state count previously stored in the register.

6.6 Register Bit Definition

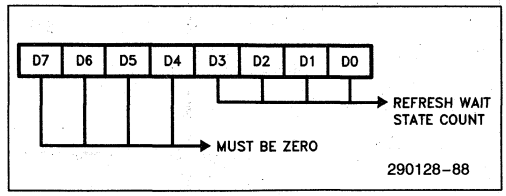
WAIT STATE REGISTER 0, 1, 2

Port Address	Description
72H	Wait State Register 0 (read/write)
73H	Wait State Register 1 (read/write)
74H	Wait State Register 2 (read/write)



REFRESH WAIT STATE REGISTER

Port Address: 75H (Read/Write)



6.7 Application Issues

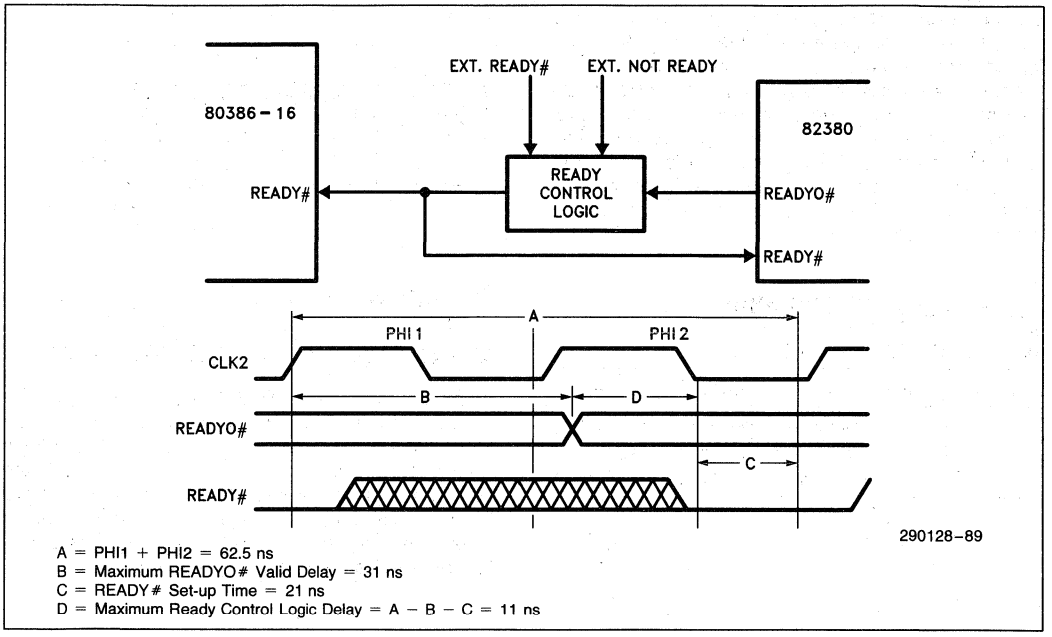
6.7.1 EXTERNAL 'READY' CONTROL LOGIC

As mentioned in section 6.3.3, wait state cycles generated by the 82380 can be terminated early or extended longer by means of additional external logic (see Figure 6-5). In order to ensure that the READY# input timing requirement of the 80386 and the 82380 is satisfied, special care must be taken when designing this external control logic. This section addresses the design requirements.

5

A simplified block diagram of the external logic along with the READY# tiiming diagram is shown in Figure 6-8. The purpose is to determine the maximum delay time allowed in the external control logic in order to satisfy the READY# setup time.

First, it will be assumed that the 80386 is running at 16 MHz (i.e., CLK2 and 32 MHz). Therefore, one bus state (two CLK2 periods) will be equivalent to 62.5 nsec. According to the AC specifications of the 82380, the maximum delay time for valid READYO# signal is 31 ns after the rising edge of CLK2 in the beginning of T2 (for non-pipelined cycle) or T2P (for pipelined cycle). Also, the minimum READY# setup time of the 80386 and the 82380 should be 20 ns before the rising edge of CLK2 at the beginning of the next bus state. This limits the total delay time for the external READY# control logic to be 11 ns (62.5-31-21) in order to meet the READY# setup timing requirement.



A = PHI1 + PHI2 = 62.5 ns
B = Maximum READYO# Valid Delay = 31 ns
C = READY# Set-up Time = 21 ns
D = Maximum Ready Control Logic Delay = A – B – C = 11 ns

Figure 6-8. 'READY' Timing Consideration

7.0 DRAM REFRESH CONTROLLER

7.1 Functional Description

The 82380 DRAM Refresh Controller consists of a 24-bit Refresh Address Counter and Refresh Request logic for DRAM refresh operations (see Figure 7-1). TIMER 1 can be used as a trigger signal to the DRAM Refresh Request logic. The Refresh Bus Size can be programmed to be 8-, 16-, or 32-bit wide. Depending on the Refresh Bus Size, the Refresh Address Counter will be incremented with the appropriate value after every refresh cycle. The internal logic of the 82380 will give the Refresh operation the highest priority in the bus control arbitration process. Bus control is not released and re-requested if the 82380 is already a bus master.

7.2 Interface Signals

7.2.1 TOUT1/REF#

The dual function output pin of TIMER 1 (TOUT1/REF#) can be programmed to generate DRAM Refresh signal. If this feature is enabled, the rising edge of TIMER 1 output (TOUT1) will trigger the DRAM Refresh Request logic. After some delay for gaining access of the bus, the 82380 DRAM Controller will generate a DRAM Refresh signal by driving REF# output LOW. This signal is cleared after the refresh cycle has taken place, or by a hardware reset.

If the DRAM Refresh feature is disabled, the TOUT1/REF# output pin is simply the TIMER 1 output. Detailed information of how TIMER 1 operates is discussed in section 6—Programmable Interval Timer, and will not be repeated here.

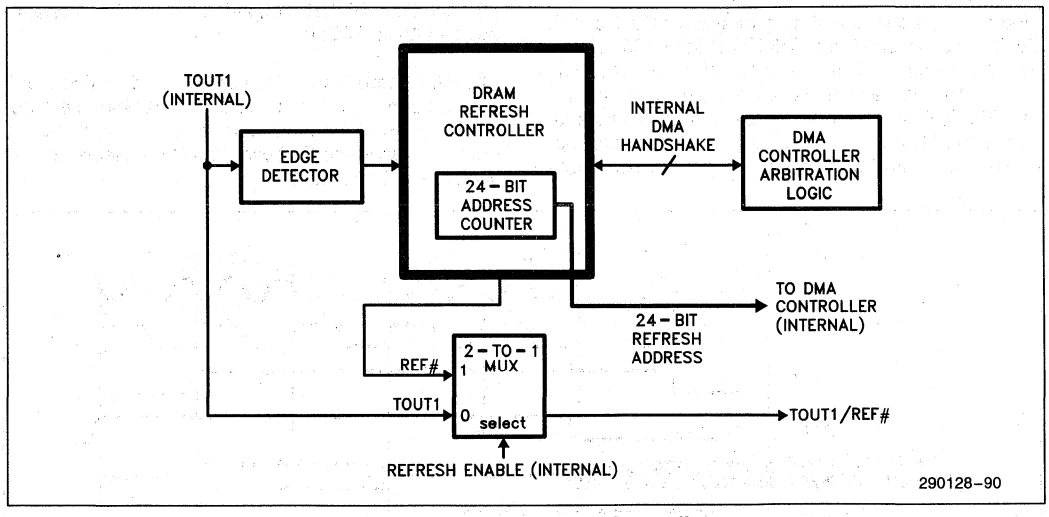


Figure 7-1. DRAM Refresh Controller

7.3 Bus Function

7.3.1 ARBITRATION

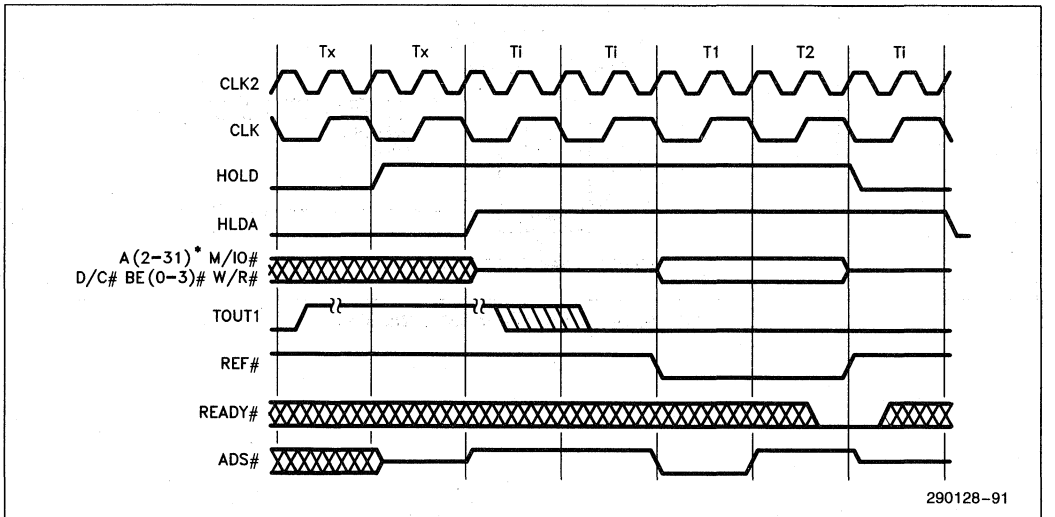
In order to ensure data integrity of the DRAMs, the 82380 gives the DRAM Refresh signal the highest priority in the arbitration logic. It allows DRAM Refresh to interrupt a DMA in progress in order to perform the DRAM Refresh cycle. The DMA service will be resumed after the refresh is done.

In case of a DRAM Refresh during a DMA process, the cascaded device will be requested to get off the bus. This is done by deasserting the EDACK signal. Once DREQn goes inactive, the 82380 will perform the refresh operation. Note that the DMA controller does not completely relinquish the system bus during refresh. The Refresh Generator simply 'steals' a bus cycle between DMA accesses.

Figure 7-2 shows the timing diagram of a Refresh Cycle. Upon expiration of TIMER 1, the 82380 will try to take control of the system bus by asserting HOLD. As soon as the 82380 see HLDA go active, the DRAM Refresh Cycle will be carried out by activating the REF# signal as well as the refresh address and control signals on the system bus (Note that REF# will not be active until two CLK periods after HLDA is asserted). The address bus will contain the 24-bit address currently in the Refresh Address Counter. The control signals are driven the same way as in a Memory Read cycle. This 'read' operation is complete when the READY# signal is driven LOW. Then, the 82380 will relinquish the bus by de-asserting HOLD. Typically, a Refresh Cycle without wait states will take five bus states to execute. If 'n' wait states are added, the Refresh Cycle will last for five plus 'n' bus states.

How often the Refresh Generation will initiate a refresh cycle depends on the frequency of CLKIN as well as TIMER1's programmed mode of operation. For this specific application, TIMER1 should be programmed to operate in Mode 2 or 3 to generate a constant clock rate. See section 6—Programmable Interval Timer for more information on programming the timer. One DRAM Refresh Cycle will be generated each time TIMER 1 expires (when TOUT1 changes to LOW to HIGH).

The Wait State Generator can be used to insert wait states during a refresh cycle. The 82380 will automatically insert the desired number of wait states as programmed in the Refresh Wait State Register (see Wait State Generator).



*NOTE:
A24–A31 = 1 during Refresh cycle.

Figure 7-2. 82380 Refresh Cycle

7.4 Modes of Operation

7.4.1 WORD SIZE AND REFRESH ADDRESS COUNTER

The 82380 supports 8-, 16- and 32-bit refresh cycle. The bus width during a refresh cycle is programmable (see Programming). The bus size can be programmed via the Refresh Control Register (see Register Overview). If the DRAM bus size is 8-, 16-, or 32-bits, the Refresh Address Counter will be incremented by 1, 2, or 4, respectively.

The Refresh Address Counter is cleared by a hardware reset.

7.5 Register Set Overview

The Refresh Generator has two internal registers to control its operation. They are the Refresh Control Register and the Refresh Wait State Register. Their port address map is shown in Table 7-1 below.

Port Address	Description
1CH	Refresh Control Reg. (read/write)
75H	Ref. Wait State Reg. (read/write)

Table 7-1. Register Address Map

The Refresh Wait State Register is not part of the Refresh Generator. It is only used to program the number of wait states to be inserted during a refresh cycle. This register is discussed in detail in section 7 (Wait State Generator) and will not be repeated here.

REFRESH CONTROL REGISTER

This 2-bit register serves two functions. First, it is used to enable/disable the DRAM Refresh function output. If disabled, the output of TIMER 1 is simply used as a general purpose timer. The second function of this register is to program the DRAM bus size for the refresh operation. The programmed bus size also determines how the Refresh Address Counter will be incremented after each refresh operation.

7.6 Programming

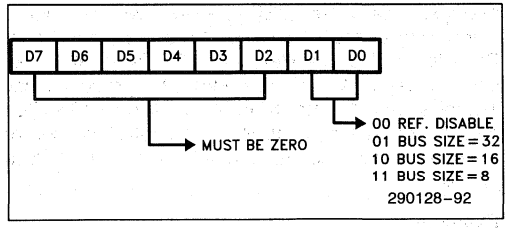
Upon hardware reset, the DRAM Refresh function is disabled (the Refresh Control Register is cleared). The following programming steps are needed before the Refresh Generator can be used. Since the rate of refresh cycles depends on how TIMER 1 is programmed, this timer must be initialized with the desired mode of operation as well as the correct refresh interval (see Programming Interval Timer).

Whether or not wait states are to be generated during a refresh cycle, the Refresh Wait State Register must also be programmed with the appropriate value. Then, the DRAM Refresh feature must be enabled and the DRAM bus width should be defined. These can be done in one step by writing the appropriate control word into the Refresh Control Register (see Register Bit Definition). After these steps are done, the refresh operation will automatically be invoked by the Refresh Generator upon expiration of Timer 1.

In addition to the above programming steps, it should be noted that after reset, although the TOUT1/REF# becomes the Timer 1 output, the state of this pin is undefined. This is because the Timer module has not been initialized yet. Therefore, if this output is used as a DRAM Refresh signal, this pin should be disqualified by external logic until the Refresh function is enabled. One simple solution is to logically AND this output with HLDA, since HLDA should not be active after reset.

7.7 Register Bit Definition

REFRESH CONTROL REGISTER
Port Address: 1CH (Read/Write)



8.0 RELOCATION REGISTER AND ADDRESS DECODE

8.1 Relocation Register

All the integrated peripheral devices in the 82380 are controlled by a set of internal registers. These registers span a total of 256 consecutive address locations (although not all the 256 locations are used). The 82380 provides a Relocation Register which allows the user to map this set of internal registers into either the memory or I/O address space. The function of the Relocation Register is to define the base address of the internal register set of the 82380 as well as if the registers are to be memory- or I/O-mapped. The format of the Relocation Register is depicted in Figure 8-1.

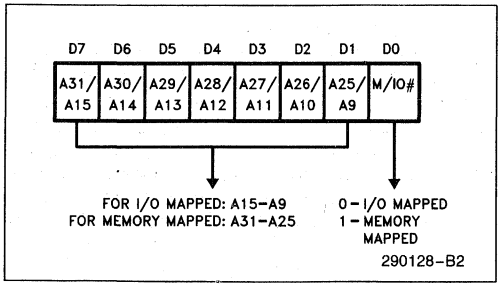


Figure 8-1. Relocation Register

Note that the Relocation Register is part of the internal register set of the 82380. It has a port address of 7FH. Therefore, any time the content of the Relocation Register is changed, the physical location of this register will also be moved. Upon reset of the 82380, the content of the Relocation Register will be cleared. This implies that the 82380 will respond to its I/O addresses in the range of 0000H to 00FFH.

8.1.1 I/O-MAPPED 82380

As shown in the figure, Bit 0 of the Relocation Register determines whether the 82380 registers are to be memory-mapped or I/O-mapped. When Bit 0 is set to '0', the 82380 will respond to I/O Addresses. Address signals BE0#–BE3#, A2–A7 will be used to select one of the internal registers to be accessed. Bit 1 to Bit 7 of the Relocation Register will correspond to A9 to A15 of the Address bus, respectively. Together with A8 implied to be '0', A15 to A8 will be fully decoded by the 82380. The following shows how the 82380 is mapped into the I/O address space.

Example

Relocation Register = 11001110 (0CEH)

82380 will respond to I/O address range from 0CE00H to 0CEFFH.

Therefore, this I/O mapping mechanism allows the 82380 internal registers to be located on any even, contiguous, 256 byte boundary of the system I/O space.

Port Address: 7FH (Read/Write)

8.1.2 MEMORY-MAPPED 82380

When Bit 0 of the Relocation Register is set to '1', the 82380 will respond to memory addresses. Again, Address signals BE0#–BE3#, A2–A7 will be used to select one of the internal registers to be accessed. Bit 1 to Bit 7 of the Relocation Register will correspond to A25–A31, respectively. A24 is assumed to be '0', and A8–A23 are ignored. Consider the following example.

Example

Relocation Register = 10100111 (0A7H)

The 82380 will respond to memory addresses in the range of 0A6XXXX00H to 0A6XXXXFFH (where 'X' is don't care).

This scheme implies that the internal register can be located in any even, contiguous, 2**24 byte page of the memory space.

8.2 Address Decoding

As mentioned previously, the 82380 internal registers do not occupy the entire contiguous 256 address locations. Some of the locations are 'unoccupied'. The 82380 always decodes the lower 8 address bits (A0–A7) to determine if any one of its registers is being accessed. If the address does not correspond to any of its registers, the 82380 will not respond. This allows external devices to be located within the 'holes' in the 82380 address space. Note that there are several unused addresses reserved for future Intel peripheral devices.

9.0 CPU RESET AND SHUTDOWN DETECT

The 82380 will activate the CPURST signal to reset the host processor when one of the following conditions occurs:

— 82380 RESET is active;

— 82380 detects a 80386 Shutdown cycle (this feature can be disabled);

— CPURST software command is issued to 80386.

Whenever the CPURST signal is activated, the 82380 will reset its own internal Slave-Bus state machine.

9.1 Hardware Reset

Following a hardware reset, the 82380 will assert its CPURST output to reset the host processor. This output will stay active for as long as the RESET input is active. During a hardware reset, the 82380 internal registers will be initialized as defined in the corresponding functional descriptions.

9.2 Software Reset

CPURST can be generated by writing the following bit pattern into 82380 register location 64H.

D7							D0
1	1	1	1	X	X	X	0

X = Don't Care

The Write operation into this port is considered as an 82380 access and the internal Wait State Generator will automatically determine the required number of wait states. The CPURST will be active following the completion of the Write cycle to this port. This signal will last for 62 CLK2 periods. The 82380 should not be accessed until the CPURST is deactivated.

This internal port is Write-Only and the 82380 will not respond to a Read operation to this location. Also, during a CPU software reset command, the 82380 will reset its Slave-Bus state machine. However, its internal registers remain unchanged. This allows the operating system to distinguish a 'warm' reset by reading any 82380 internal register previously programmed for a non-default value. The Diagnostic registers can be used or this purpose (see Internal Control and Diagnostic Ports).

9.3 Shutdown Detect

The 82380 is constantly monitoring the Bus Cycle Definition signals (M/IO#, D/C#, R/W#) and is able to detect when the 80386 executes a Shutdown bus cycle. Upon detection of a processor shutdown, the 82380 will activate the CPURST output for 62 CLK2 periods to reset the host processor. This signal is generated after the Shutdown cycle is terminated by the READY# signal.

Although the 82380 Wait State Generator will not automatically respond to a Shutdown (or Halt) cycle, the Wait State Control inputs (WSC0, WSC1) can be used to determine the number of wait states in the same manner as other non-82380 bus cycle.

This Shutdown Detect feature can be enabled or disabled by writing a control bit in the Internal Control Port at address 61H (see Internal Control and Diagnostic Ports). This feature is disabled upon a hardware reset of the 82380. As in the case of Software Reset, the 82380 will reset its Slave-Bus state machine but will not change any of its internal register contents.

10.0 INTERNAL CONTROL AND DIAGNOSTIC PORTS

10.1 Internal Control Port

The format of the Internal Control Port of the 82380 is shown in Figure 10.1. This Control Port is used to enable/disable the Processor Shutdown Detect mechanism as well as controlling the Gate inputs of the Timer 2 and 3. Note that this is a Write-Only port. Therefore, the 82380 will not respond to a read operation to this port. Upon hardware reset, this port will be cleared; i.e., the Shutdown Detect feature and the Gate inputs of Timer 2 and 3 are disabled.

10.2 Diagnostic Ports

Two 8-bit read/write Diagnostic Ports are provided in the 82380. These are two storage registers and have no effect on the operation of the 82380. They can be used to store checkpoint data or error codes in the power-on sequence and in the diagnostic service routines. As mentioned in CPU RESET AND SHUTDOWN DETECT section, these Diagnostic Ports can be used to distinguish between 'cold' and 'warm' reset. Upon hardware reset, both Diagnostic Ports are cleared. The address map of these Diagnostic Ports is shown in Figure 10-2.

Port	Address
Diagnostic Port 1 (Read/Write)	80H
Diagnostic Port 2 (Read/Write)	88H

Figure 10-2. Address Map of Diagnostic Ports

Port Address: 61H (Write Only)

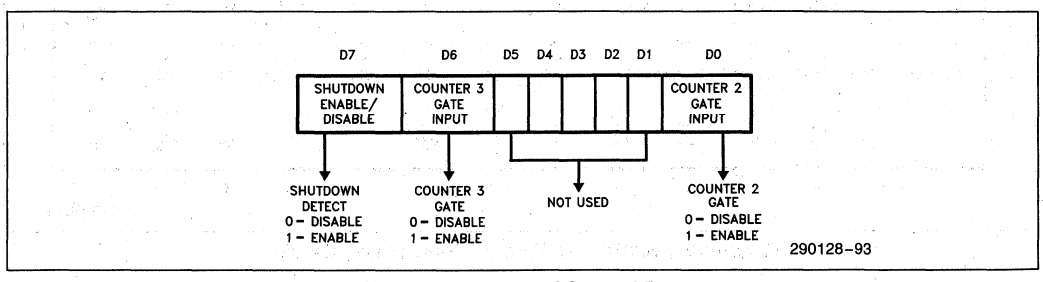


Figure 10-1. Internal Control Port

11.0 INTEL RESERVED I/O PORTS

There are eleven I/O ports in the 82380 address space which are reserved for Intel future peripheral device use only. Their address locations are: 2AH, 3DH, 3EH, 45H, 46H, 76H, 77H, 7DH, 7EH, CCH and CDH. These addresses should not be used in the system since the 82380 may respond to read/write operations to these locations and bus contention may occur if any peripheral is assigned to the same address location.

12.0 MECHANICAL DATA

12.1 Introduction

In this section, the physical package and its connections are described in detail.

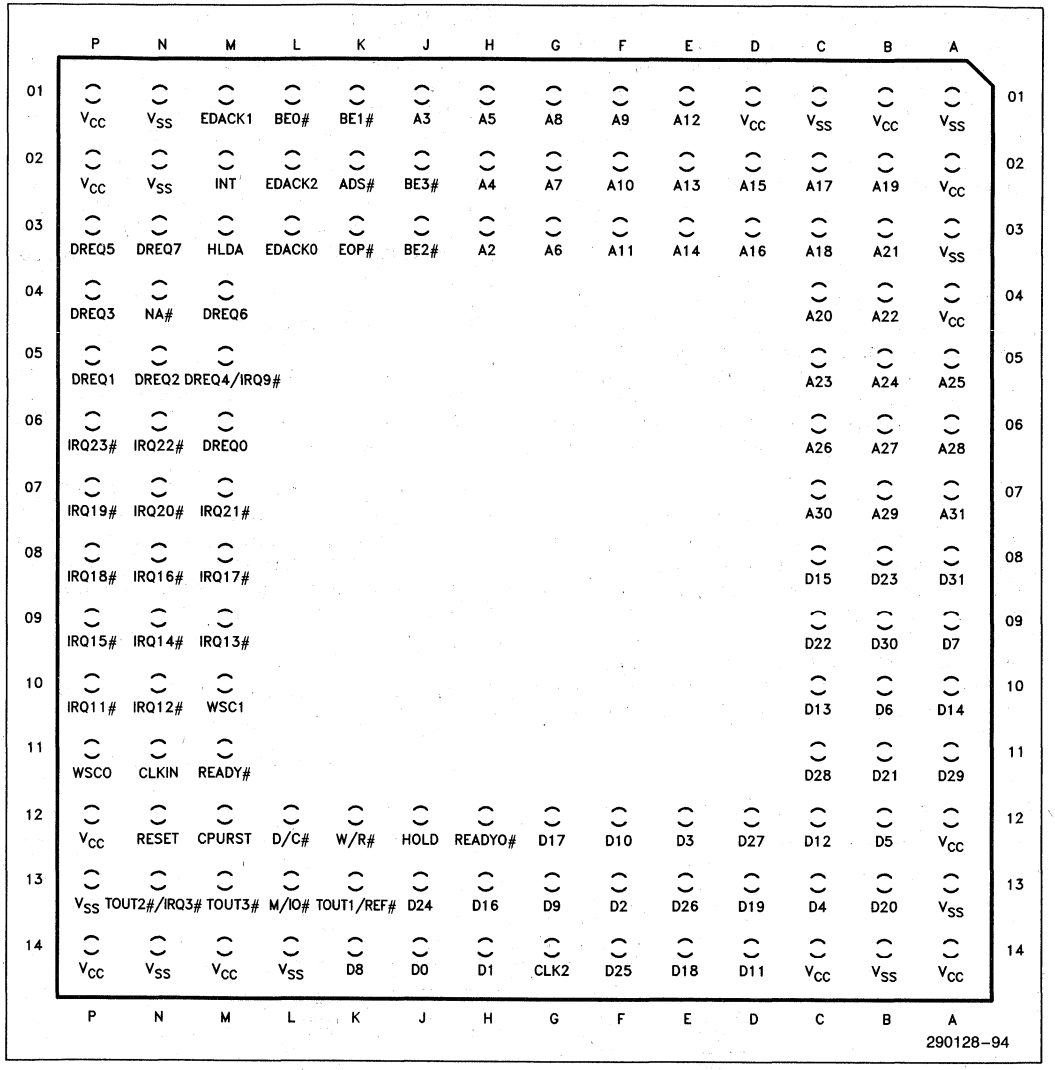


Figure 12.1. 82380 PGA Pinout—View from TOP side

12.2 Pin Assignment

The 82380 pinout as viewed from the top side of the component is shown in Figure 12.1. Its pinout as viewed from the pin side of the component is shown in Figure 12.2.

V_{CC} and GND connections must be made to multiple V_{CC} and V_{SS} (GND) pins. Each V_{CC} and V_{SS} MUST be connected to the appropriate voltage level. The circuit board should include V_{CC} and GND planes for power distribution and all V_{CC} pins must be connected to the appropriate plane.

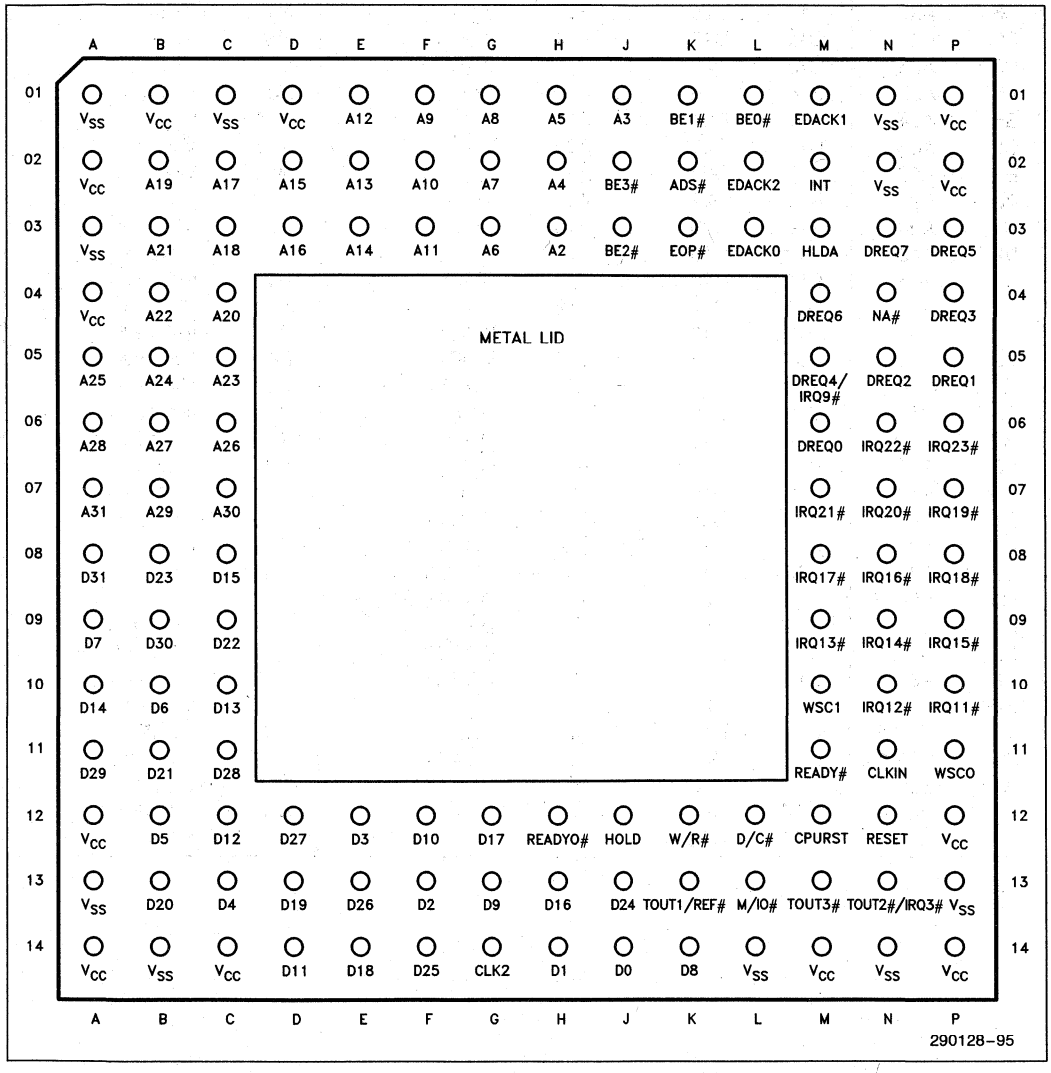


Figure 12.2. 82380 PGA Pinout—View from PIN side

Table 12-1. 82380 PGA Pinout—Functional Grouping

Pin/Signal		Pin/Signal		Pin/Signal		Pin/Signal	
A7	A31	A8	D31	P12	V_{CC}	L14	V_{SS}
C7	A30	B9	D30	M14	V_{CC}	A1	V_{SS}
B7	A29	A11	D29	P1	V_{CC}	P13	V_{SS}
A6	A28	C11	D28	P2	V_{CC}	N1	V_{SS}
B6	A27	D12	D27	P14	V_{CC}	N2	V_{SS}
C6	A26	E13	D26	D1	V_{CC}	C1	V_{SS}
A5	A25	F14	D25	C14	V_{CC}	A3	V_{SS}
B5	A24	J13	D24	B1	V_{CC}	B14	V_{SS}
C5	A23	B8	D23	A2	V_{CC}	A13	V_{SS}
B4	A22	C9	D22	A4	V_{CC}	N14	V_{SS}
B3	A21	B11	D21	A12	V_{CC}		
C4	A20	B13	D20	A14	V_{CC}	P6	IRQ23#
B2	A19	D13	D19			N6	IRQ22#
C3	A18	E14	D18	G14	CLK2	M7	IRQ21#
C2	A17	G12	D17	L12	D/C#	N7	IRQ20#
D3	A16	H13	D16	K12	W/R#	P7	IRQ19#
D2	A15	C8	D15	L13	M/IO#	P8	IRQ18#
E3	A14	A10	D14	K2	ADS#	M8	IRQ17#
E2	A13	C10	D13	N4	NA#	N8	IRQ16#
E1	A12	C12	D12	J12	HOLD	P9	IRQ15#
F3	A11	D14	D11	M3	HLDA	N9	IRQ14#
F2	A10	F12	D10	M6	DREQ0	M9	IRQ13#
F1	A9	G13	D9	P5	DREQ1	N10	IRQ12#
G1	A8	K14	D8	N5	DREQ2	P10	IRQ11#
G2	A7	A9	D7	P4	DREQ3	M2	INT
G3	A6	B10	D6	M5	DREQ4/IRQ9#		
H1	A5	B12	D5	P3	DREQ5	N11	CLKIN
H2	A4	C13	D4	M4	DREQ6	K13	TOUT1/REF#
J1	A3	E12	D3	N3	DREQ7	N13	TOUT2#/IRQ3#
H3	A2	F13	D2			M13	TOUT3#
J2	BE3#	H14	D1	K3	EOP#	M11	READY#
J3	BE2#	J14	D0	L3	EDACK0	H12	READYO#
K1	BE1#			M1	EDACK1	P11	WSC0
L1	BE0#	N12	RESET	L2	EDACK2	M10	WSC1
		M12	CPURST				

12.3 Package Dimensions and Mounting

The 82380 package is a 132-pin ceramic Pin Grid Array (PGA). The pins are arranged 0.100 inch (2.54 mm) center-to-center, in a 14 x 14 matrix, three rows around.

A wide variety of available sockets allow low insertion force or zero insertion force mountings, and a choice of terminals such as soldertail, surface mount, or wire wrap. Several applicable sockets are listed in Figure 12-4.

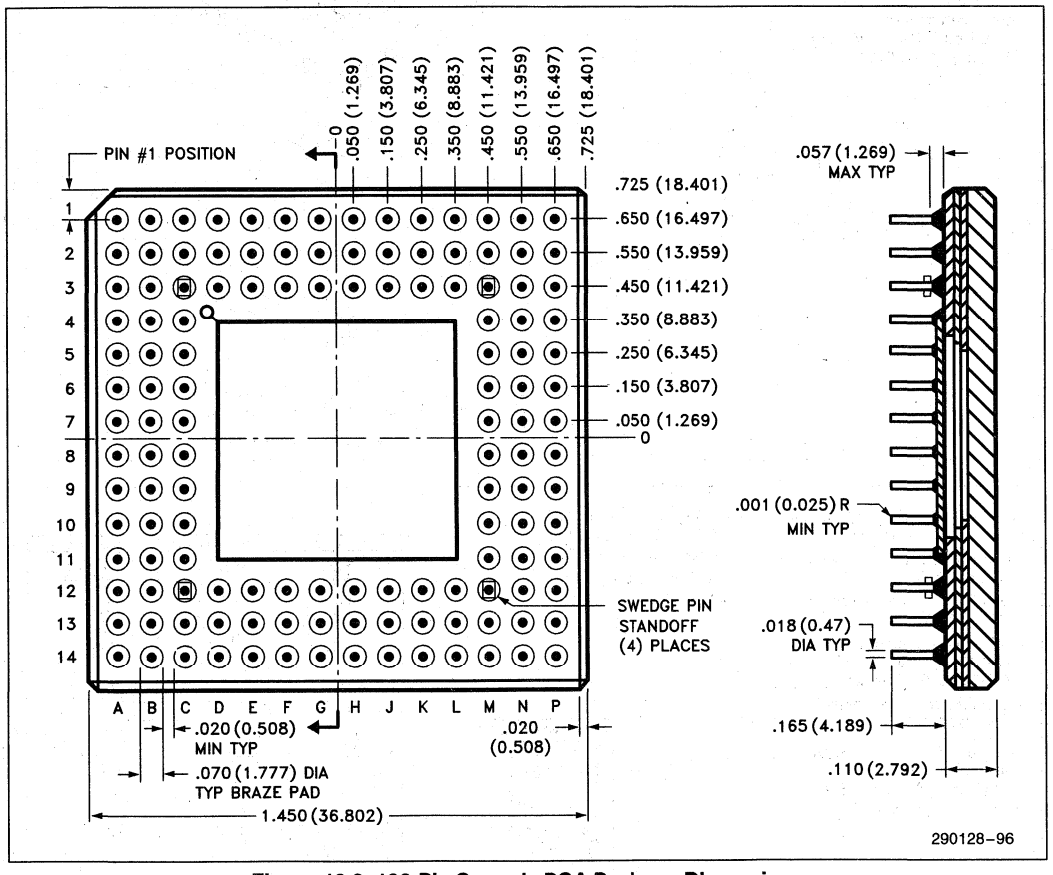


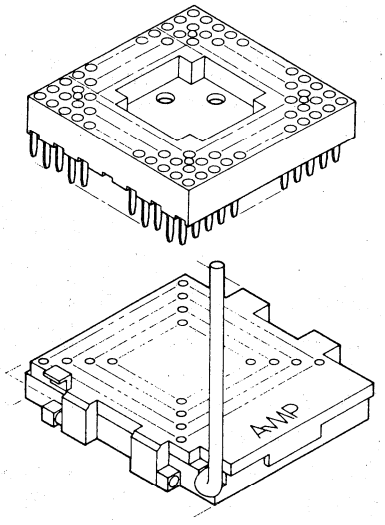
Figure 12.3. 132-Pin Ceramic PGA Package Dimensions

- Low insertion force (LIF) soldertail 55274-1
- Amp tests indicate 50% reduction in insertion force compared to machined sockets

Other socket options

- Zero insertion force (ZIF) soldertail 55583-1
- Zero insertion force (ZIF) Burn-in version 55573-2

Amp Incorporated
(Harrisburg, PA 17105 U.S.A.
Phone 717-564-0100)



290128-97

Cam handle locks in low profile position when substrate is installed (handle UP for open and DOWN for closed positions)

courtesy Amp Incorporated

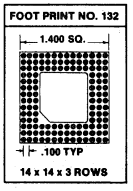
Peel-A-Way™ Mylar and Kapton Socket Terminal Carriers

- Low insertion force surface mount CS132-37TG
- Low insertion force soldertail CS132-01TG
- Low insertion force wire-wrap CS132-02TG (two level) CS132-03TG (three-level)
- Low insertion force press-fit CS132-05TG

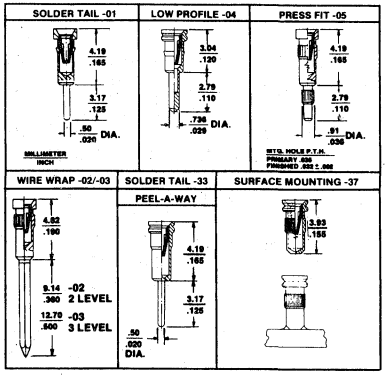
Advanced Interconnections
(5 Division Street
Warwick, RI 02818 U.S.A.
Phone 401-885-0485)

Peel-A-Way Carrier No. 132;
Kapton Carrier is KS132
Mylar Carrier is MS132

Molded Plastic Body KS132 is shown below:



290128-98



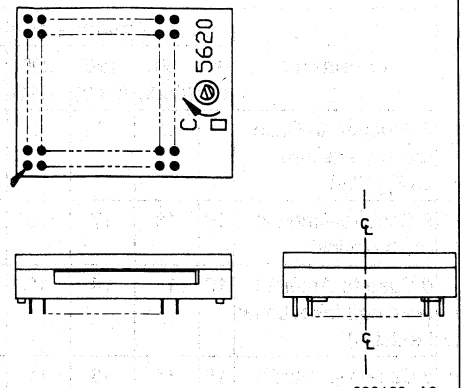
290128-99

courtesy Advanced Interconnections
(Peel-A-Way Terminal Carriers
U.S. Patent No. 4442938)

Figure 12-4. Several Socket Options for 132-pin PGA

- Low insertion force socket soldertail
  (for production use)
  2XX-6576-00-3308 (new style)
  2XX-6003-00-3302 (older style)
- Zero insertion force soldertail
  (for test and burn-in use)
  2XX-6568-00-3302

Textool Products
Electronic Products Division/3m
(1410 West Pioneer Drive
Irving, Texas 75601 U.S.A.
Phone 214-259-2676)



courtesy Textoll Products/3M

Figure 12-4. Several Socket Options for 132-pin PGA (Continued)

12.4 Package Thermal Specification

The 82380 is specified for operation when case temperature is within the range of 0°C −85°C. The case temperature may be measured in any environment, to determine whether the 82380 is within the specified operating range.

The PGA case temperature should be measured at the center of the top surface opposite the pins, as in Figure 12.5.

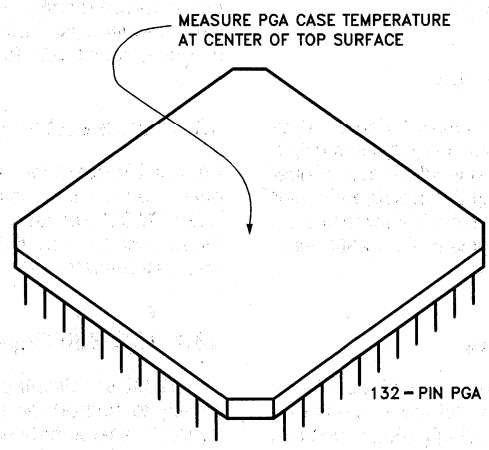
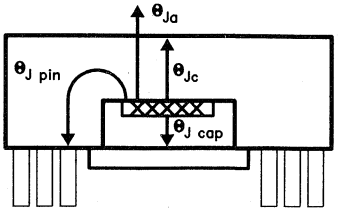


Figure 12.5. Measuring 82380 PGA Case Temperature

Thermal Resistance—°C/Watt							
Parameter	Airflow—f³/min (m³/sec)						
	0 (0)	50 (0.25)	100 (0.50)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)
θ Junction-to-Case (case measured as Fig. 6.4)	2	2	2	2	2	2	2
θ Case-to-Ambient (no heatsink)	19	18	17	15	12	10	9
θ Case-to-Ambient (with omnidirectional heatsink)	16	15	14	12	9	7	6
θ Case-to-Ambient (with unidirectional heatsink)	15	14	13	11	8	6	5



290128-A2

NOTES:
1. Table 12-6 applies to 82380 PGA plugged into socket or soldered directly into board.
2. $\theta_{JA} = \theta_{JC} + \theta_{CA}$.
3. $\theta_{J\text{-}CAP} = 4°C/W$ (approx.)
   $\theta_{J\text{-}PIN} = 4°C/W$ (inner pins) (approx.)
   $\theta_{J\text{-}PIN} = 8°C/W$ (outer pins) (approx.)

Figure 12-6. 82380 PGA Package Typical Thermal Characteristics

13.0 ELECTRICAL DATA

13.1 Power and Grounding

The large number of output buffers (address, data and control) can cause power surges as multiple output buffers drive new signal levels simultaneously. The 22 V_{CC} and V_{SS} pins of the 82380 each feed separate functional units to minimize switching induced noise effects. All V_{CC} pins of the 82380 must be connected on the circuit board.

13.2 Power Decoupling

Liberal decoupling capacitance should be placed close to the 82380. The 82380 driving its 32-bit parallel address and data buses at high frequencies can cause transient power surges when driving large capacitive loads. Low inductance capacitors and interconnects are recommended for the best reliability at high frequencies. Low inductance capacitors are available specifically for Pin Grid Array packages.

13.3 Unused Pin Recommendations

For reliable operation, ALWAYS connect unused inputs to a valid logic level. As is the case with most other CMOS processes, a floating input will increase the current consumption of the component and give an indeterminate state to the component.

13.4 ICE-386 Support

The 82380 specifications provide sufficient drive capability to support the ICE386. On the pins that are generally shared between the 80386 and the 82380, the additional loading represented by the ICE386 was allowed for in the design of the 82380.

13.5 Maximum Ratings

Storage Temperature ..... −65°C to +150°C
Case temperature Under Bias ... −65°C to +110°C
Supply Voltage with Respect to V_{SS} ..... −0.5V to +6.5V
Voltage on any other Pin ..... −0.5V to V_{CC} +0.5V

NOTE:
Stress above those listed above may cause permanent damage to the device. This is a stress rating only and functional operation at these or any other conditions above those listed in the operational sections of this specification is not implied.

Exposure to absolute maximum rating conditions for extended periods may affect device reliability. Although the 82380 contains protective circuitry to reset damage from static electric discharges, always take precautions against high static voltages or electric fields.

13.6 D.C. Specifications

T_{CASE} = 0°C to 85°C; V_{CC} = 5V ±5%; V_{SS} = 0V.

Table 13-1.

Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	0.8	V	(Note 1)
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{ILC}	CLK2 Input Low Voltage	−0.3	0.8		(Note 1)
V_{IHC}	CLK2 Input High Voltage	V_{CC} − 0.8	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage I_{OL} = 4 mA: A2–A31, D0–D31 I_{OL} = 5 mA: All Others		 0.45 0.45	 V V	
V_{OH}	Output High Voltage I_{OH} = −1 mA: A2–A31, D0–D31 I_{OH} = −0.9 mA: All Others	 2.4 2.4		 V V	
I_{LI}	Input Leakage Current for all ins except: IRQ11#–IRQ23#, TOUT2/IRQ3#, EOP#, DREQ4		±15	μA	$0V < V_{IN} < V_{CC}$
I_{LI1}	Input Leakage Current for pins: IRQ11#–IRQ23#, TOUT2#/IRQ3#, EOP#, DREQ4	10	−300	μA	$0V < V_{IN} < V_{CC}$ (Note 3)
I_{LO}	Output Leakage Current		±15	μA	$0.45 < V_{OUT} < V_{CC}$
I_{CC}	Supply Current		300 325	mA mA	CLK2 = 32 MHz = 40 MHz (Note 4)
(CAP)	Capacitance (Input/IO)		12	pF	f_c = 1 MHz (Note 2)
CCLK	CLK2 Capacitance		20	pF	f_c = 1 MHz (Note 2)

NOTES:
1. Minimum value is not 100% tested.
2. Sampled only.
3. These pins have internal pullups on them.
4. I_{CC} is specified with inputs driven to CMOS levels. I_{CC} may be higher if driven to TTL levels.

13.6 D.C. Specifications (Continued)

T_{CASE} = 0°C to 85°C; V_{CC} = 5V ±5%; V_{SS} = 0V.

Table 13-2. 82380-25 D.C. Specifications

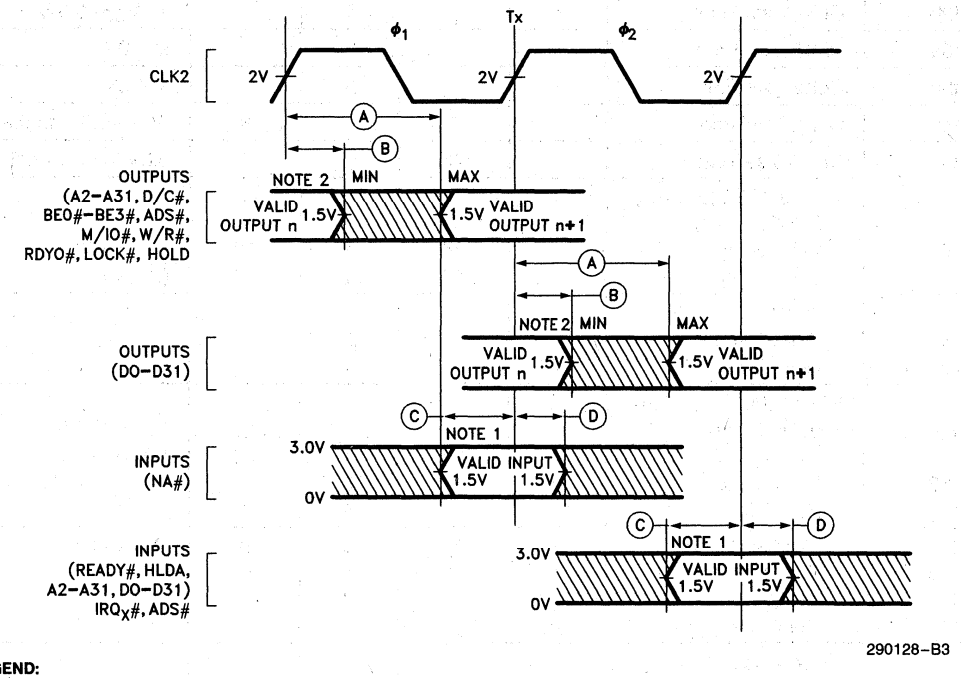
Symbol	Parameter	Min	Max	Unit	Notes
V_{IL}	Input Low Voltage	−0.3	0.8	V	(Note 1)
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{ILC}	CLK2 Input Low Voltage	−0.3	0.8	V	(Note 1)
V_{IHC}	CLK2 Input High Voltage	V_{CC} − 0.8	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage I_{OL} = 4 mA: A_2–A_{31}, D_0–D_{31} I_{OL} = 5 mA: All Others		 0.45 0.45	 V V	
V_{OH}	Output High Voltage I_{OH} = −1 mA: A_2–A_{31}, D_0–D_{31} I_{OH} = −0.9 mA: All Others	 2.4 2.4		 V V	
I_{LI}	Input Leakage Current All Inputs except: IRQ11# – IRQ23#, EOP#, TOUT2/IRQ3#, DREQ4		±15	µA	
I_{LI1}	Input Leakage Current Inputs: IRQ11# – IRQ23#, EOP#, TOUT2/IRQ3#, DREQ4	10	−300	µA	$0 < V_{IN} < V_{CC}$ (Note 3)
I_{LO}	Output Leakage Current		±15	µA	$0 < V_{IN} < V_{CC}$
I_{CC}	Supply Current (CLK2 = 50 MHz)		375	mA	(Note 4)
C_I	Input Capacitance		12	pF	(Note 2)
C_{CLK}	CLK2 Input Capacitance		20	pF	(Note 2)

NOTES:
1. Minimum value is not 100% tested.
2. f_C = 1 MHz; Sampled only.
3. These pins have weak internal pullups. They should not be left floating.
4. I_{CC} is specified with inputs driven to CMOS levels, and outputs driving CMOS loads. I_{CC} may be higher if inputs are driven to TTL levels, or if outputs are driving TTL loads.

13.7 A.C. Specifications

The A.C. specifications given in the following tables consist of output delays and input setup requirements. The A.C. diagram's purpose is to illustrate the clock edges from which the timing parameters are measured. The reader should not infer any other timing relationships from them. For specific information on timing relationships between signals, refer to the appropriate functional section.

A.C. spec measurement is defined in Figure 13.1. Inputs must be driven to the levels shown when A.C. specifications are measured. 82380 output delays are specified with minimum and maximum limits, which are measured as shown. The minimum 82380 output delay times are hold times for external circuitry. 82380 input setup and hold times are specified as minimums and define the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct 82380 operation.



LEGEND:

Ⓐ—maximum output delay spec
Ⓑ—minimum output delay spec
Ⓒ—minimum input setup spec
Ⓓ—minimum input hold spec

NOTES:

1. Input waveforms have tr ≤ 2.0 ns from 0.8V to 2.0V.
2. Under rated loading (120 pF) 82380 output tr, tf is typically ≤ 4.0 ns from 0.8V to 2.0V.

Figure 13-1. Drive Levels and Measurement Points for A.C. Specification

A.C. SPECIFICATION TABLES

Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C

Table 13-3. 82380 A.C. Characteristics

Symbol	Parameter	82380-16		82380-20		Notes
		Min	Max	Min	Max	
	Operating Frequency	4 MHz	16 MHz	4 MHz	20 MHz	Half CLK2 Frequency
t1	CLK2 Period	31 ns	125 ns	25 ns	125 ns	
t2a	CLK2 High Time	9		8		at 2.0V
t2b	CLK2 High Time	5		5		at (V_{CC}–0.8)V
t3a	CLK2 Low Time	9		8		at 2.0V
t3b	CLK2 Low Time	7		6		at 0.8V
t4	CLK2 Fall Time		8		8	(V_{CC}–0.8)V to 0.8V
t5	CLK2 Rise Time		8		8	0.8V to (V_{CC}–0.8)V
	A (2–31), BE (0–3) #, EDACK (0–2)					
t6	Valid Delay	4	36	4	30	CL = 120 pF
t7	Float Delay	4	40	4	32	(Note 1)
	A (2–31), BE (0–3) #					
t8	Setup Time	6		6		
t9	Hold Time	4		4		
	W/R#, M/IO#, D/C#,					
t10	Valid Delay	6	33	6	28	CL = 75 pF
t11	Float Delay	4	35	4	30	(Note 1)
t12	Setup Time	6		6		
t13	Hold Time	4		4		
t14	ADS# Valid Delay	6	33	6	28	CL = 75 pF
t15	Float Delay	4	35	4	30	
t16	Setup Time	21		15		
t17	Hold Time	4		4		
	Slave Mode— D(0–31) Read					
t18	Valid Delay	3	46	4	46	CL = 120 pF
t19	Float Delay	6	35	6	29	(Note 1)
	Slave Mode— D(0–31) Write					
t20	Setup Time	31		29		
t21	Hold Time	26		26		

A.C. SPECIFICATION TABLES (Continued)

Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C.

Table 13-3. 82380 A.C. Characteristics (Continued)

Symbol	Parameter	82380-16		82380-20		Notes
		Min	Max	Min	Max	
	Master Mode— D(0–31) Write					
t22	Valid Delay	4	48	4	38	CL = 120 pF
t23	Float Delay	4	35	4	27	(Note 1)
	Master Mode— D(0–31) Read					
t24	Setup Time	11		11		
t25	Hold Time	6		6		
t26	READY# Setup Time	21		12		
t27	Hold Time	4		4		
t28	WSC (0–1) Setup	6		6		
t29	Hold	21		21		
t31	RESET Setup Time	13		12		
t30	Hold Time	4		4		
t32	READYO# Valid Delay	4	31	4	23	CL = 25 pF
t33	CPU Reset From CLK2	2	18	2	16	CL = 50 pF
t34	HOLD Valid Delay	5	33	5	30	CL = 100 pF
t35	HLDA Setup Time	21		17		
t36	Hold Time	6		6		
t37a	EOP# Setup Time	21		17		Synch. EOP
t38a	EOP# Hold Time	4		4		
t37b	EOP# Setup Time	11		11		Asynch. EOP
t38b	EOP# Hold Time	11		11		
t39	EOP# Valid Delay	5	38	5	30	CL = 100 pF ('1'-> '0')
t40	EOP# Float Delay	5	40	5	32	
t41a	DREQ Setup Time	21		19		Synchronous DREQ
t42a	Hold Time	4		4		
t41b	DREQ Setup Time	11		11		Asynchronous DREQ
t42b	Hold Time	11		11		
t43	INT Valid Delay		500		500	From IRQ Input CL = 75 pF
t44	NA# Setup Time	11		10		
t45	Hold Time	15		15		

A.C. SPECIFICATION TABLES (Continued)

Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C.

Table 13-3. 82380 A.C. Characteristics (Continued)

Symbol	Parameter	82380-16		82380-20		Notes
		Min	Max	Min	Max	
t46	CLKIN Frequency	0 MHz	10 MHz	0 MHz	10 MHz	
t47	CLKIN High Time	30		30		At 1.5V
t48	CLKIN Low Time	50		50		At 1.5V
t49	CLKIN Rise Time		10		10	0.8V to 3.7V
t50	CLKIN Fall Time		10		10	3.7V to 0.8V
t51	TOUT1/REF# Valid	4	36	4	30	From CLK2, CL = 25 pF
t52	TOUT1/REF# Valid	3	93	3	93	From CLKIN, CL = 120 pF
t53	TOUT2# Valid Delay	3	93	3	93	From CLKIN, CL = 120 pF (Falling Edge Only)
t54	TOUT2# Float Delay	3	40	3	40	From CLKIN (Note 1)
t55	TOUT3# Valid Delay	3	93	3	93	From CLKIN, CL = 120 pF

NOTE:
1. Float condition occurs when the maximum output current becomes less than ILO in magnitude. Float delay is not tested. For testing purposes, the float condition occurs when the dynamic output driven voltage changes with current loads.

Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C.
A.C. timings are tested at 1.5V thresholds; except as noted.

Table 13-4. 82380-25 A.C. Characteristics

Symbol	Parameter	82380-25		Unit	Notes
		Min	Max		
	Operating Frequency 1/(t1a × 2)	4	25	MHz	
t1	CLK2 Period	20	125	ns	
t2a	CLK2 High Time	7		ns	at 2.0V
t2b	CLK2 High Time	4		ns	at 3.7V
t3a	CLK2 Low Time	7		ns	at 2.0V
t3b	CLK2 Low Time	4		ns	at 0.8V
t4	CLK2 Fall Time		7	ns	3.7V to 0.8V
t5	CLK2 Rise Time		7	ns	0.8V to 3.7V
t6	A2–A31, BE0#–BE3# EDACK0–EDACK3 Valid Delay	4	20	ns	50 pF Load
t7	A2–A31, BE0#–BE3# EDACK0–EDACK3 Float Delay	4	27	ns	50 pF Load
t8	A2–A31, BE0#–BE3# Setup Time	6		ns	
t9	A2–A31, BE0#–BE3# Hold Time	4		ns	
t10	W/R#, M/IO#, D/C# Valid Delay	4	20	ns	50 pF Load
t11	W/R#, M/IO#, D/C# Float Delay	4	29	ns	50 pF Load

A.C. SPECIFICATION TABLES (Continued)

Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C.
A.C. timings are tested at 1.5V thresholds; except as noted.

Table 13-4. 82380-25 A.C. Characteristics (Continued)

Symbol	Parameter	82380-25		Unit	Notes
		Min	Max		
t12	W/R#, M/IO#, D/C# Setup Time	6		ns	
t13	W/R#, M/IO#, D/C# Hold Time	4		ns	
t14	ADS# Valid Delay	4	19	ns	50 pF Load
t15	ADS# Float Delay	4	29	ns	50 pF Load
t16	ADS# Setup Time	12		ns	
t17	ADS# Hold Time	4		ns	
t18	Slave Mode D0–D31 Read Valid	4	31	ns	50 pF Load
t19	Slave Mode D0–D31 Read Float	6	21	ns	50 pF Load
t20	Slave Mode D0–D31 Write Setup	20		ns	
t21	Slave Mode D0–D31 Write Hold	20		ns	
t22	Master Mode D0–D31 Write Valid	8	27	ns	50 pF Load
t23	Master Mode D0–D31 Write Float	4	19	ns	50 pF Load
t24	Master Mode D0–D31 Read Setup	7		ns	
t25	Master Mode D0–D31 Read Hold	4		ns	
t26	READY# Setup Time	9		ns	
t27	READY# Hold Time	4		ns	
t28	WSC0–WSC1 Setup Time	6		ns	
t29	WSC0–WSC1 Hold Time	15		ns	
t30	RESET Hold Time	4		ns	
t31	RESET Setup Time	9		ns	
t32	READYO# Valid Delay	3	21	ns	25 pF Load
t33	CPURST Valid Delay	2	14	ns	50 pF Load
t34	HOLD Valid Delay	4	22	ns	50 pF Load
t35	HLDA Setup Time	17		ns	
t36	HLDA Hold Time	4		ns	
t37a	EOP# Setup (Synchronous)	13		ns	
t38a	EOP# Hold (Synchronous)	4		ns	
t37b	EOP# Setup (Asynchronous)	10		ns	
t38b	EOP# Hold (Asynchronous)	10		ns	
t39	EOP# Valid Delay	4	21	ns	50 pF Load
t40	EOP# Float Delay	4	21	ns	50 pF Load
t41a	DREQ Setup (Synchronous)	17		ns	
t42a	DREQ Hold (Synchronous)	4		ns	
t41b	DREQ Setup (Asynchronous)	10		ns	
t42b	DREQ Hold (Asynchronous)	10		ns	
t43	INT Valid Delay from IRQn		500	ns	50 pF Load

A.C. SPECIFICATION TABLES (Continued)

Functional Operating Range: V_{CC} = 5V ±5%; T_{CASE} = 0°C to +85°C.
A.C. timings are tested at 1.5V thresholds; except as noted.

Table 13-4. 82380-25 A.C. Characteristics (Continued)

Symbol	Parameter	82380-25		Unit	Notes
		Min	Max		
t44	NA# Setup Time	7		ns	
t45	NA# Hold Time	8		ns	
t46	CLKIN Frequency	0	10	MHz	
t47	CLKIN High Time	30		ns	2.0V
t48	CLKIN Low Time	50		ns	0.8V
t49	CLKIN Rise Time		10	ns	0.8V to 3.7V
t50	CLKIN Fall Time		10	ns	3.7V to 0.8V
	TOUT1/REF# Valid Delay				
t51	from CLK2 (Refresh)	4	20	ns	50 pF Load
t52	from CLKIN (Timer)	3	90	ns	50 pF Load
t53	TOUT2# Valid Delay (Falling Edge Only)	3	90	ns	50 pF Load
t54	TOUT2# Float Delay	3	37	ns	50 pF Load
t55	TOUT3# Valid Delay	3	90	ns	50 pF Load

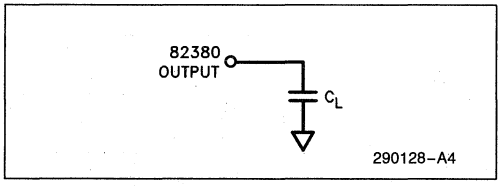


Figure 13-2. A.C. Test Load

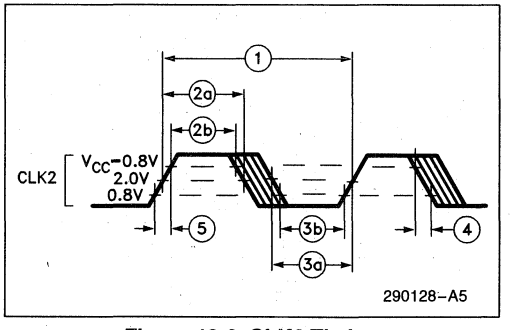


Figure 13-3. CLK2 Timing

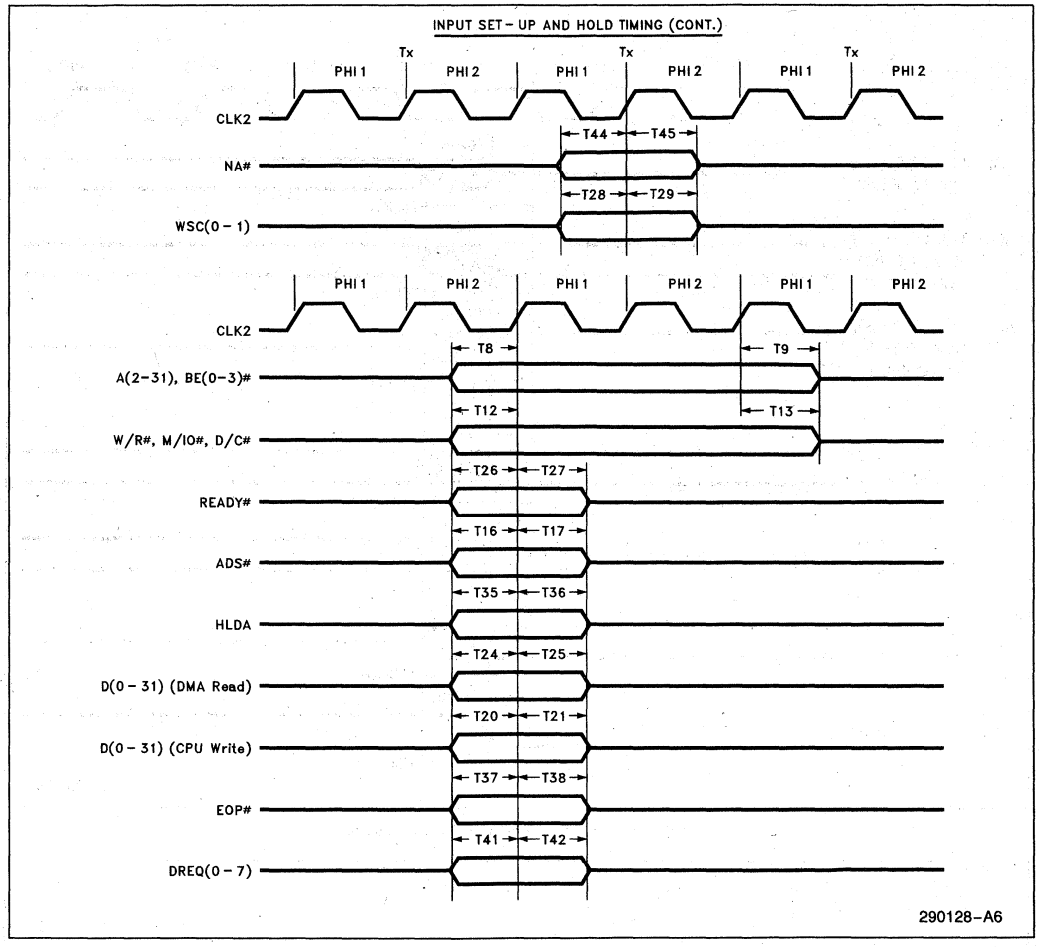


Figure 13-4. Input Setup and Hold Timing

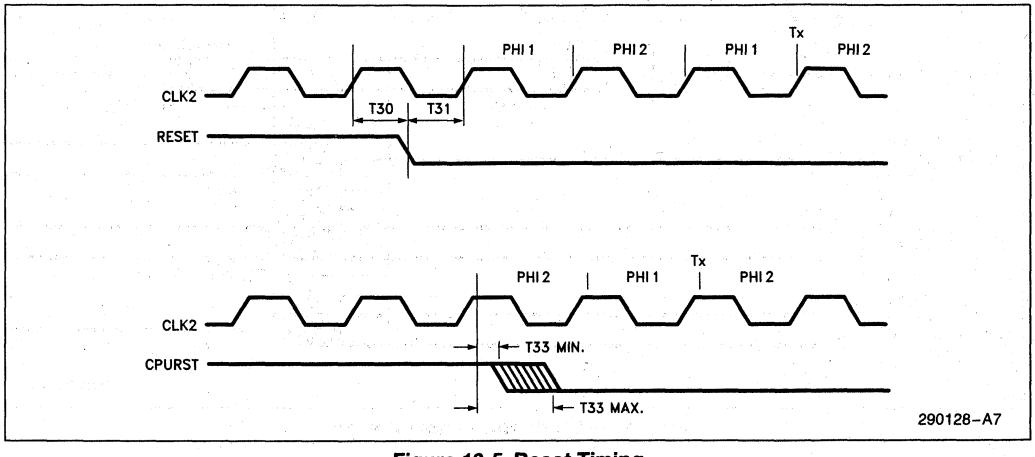


Figure 13-5. Reset Timing

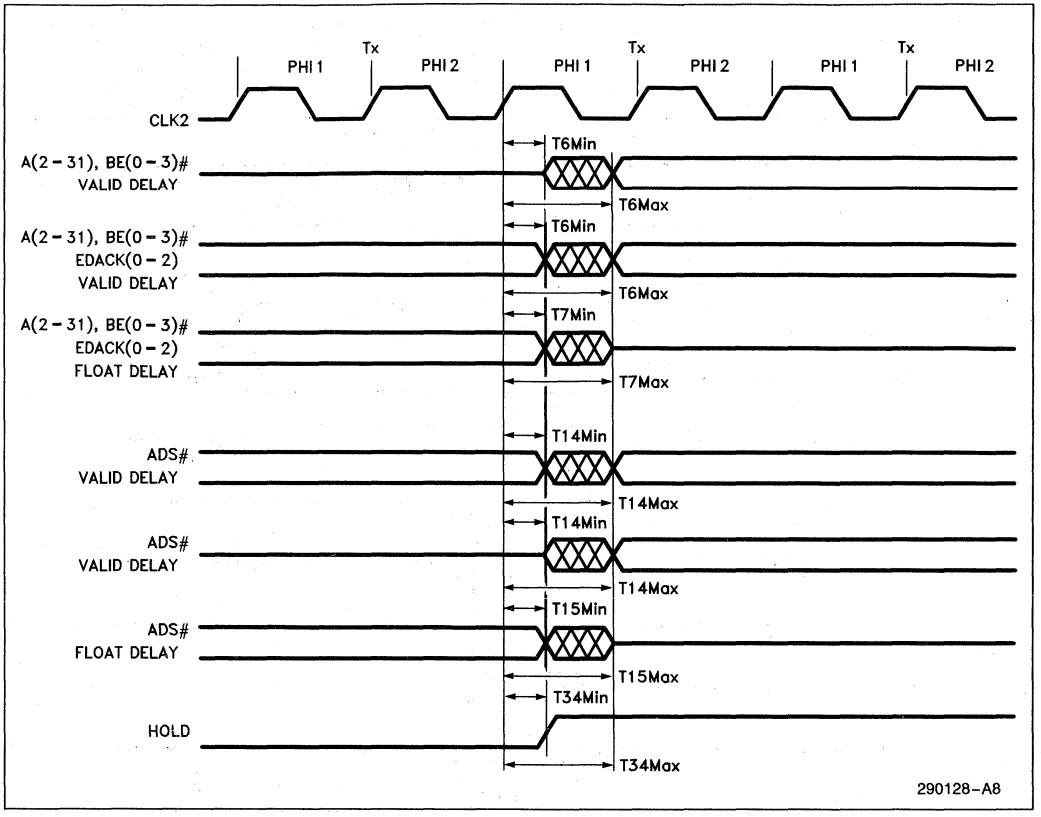


Figure 13-6. Address Output Delays

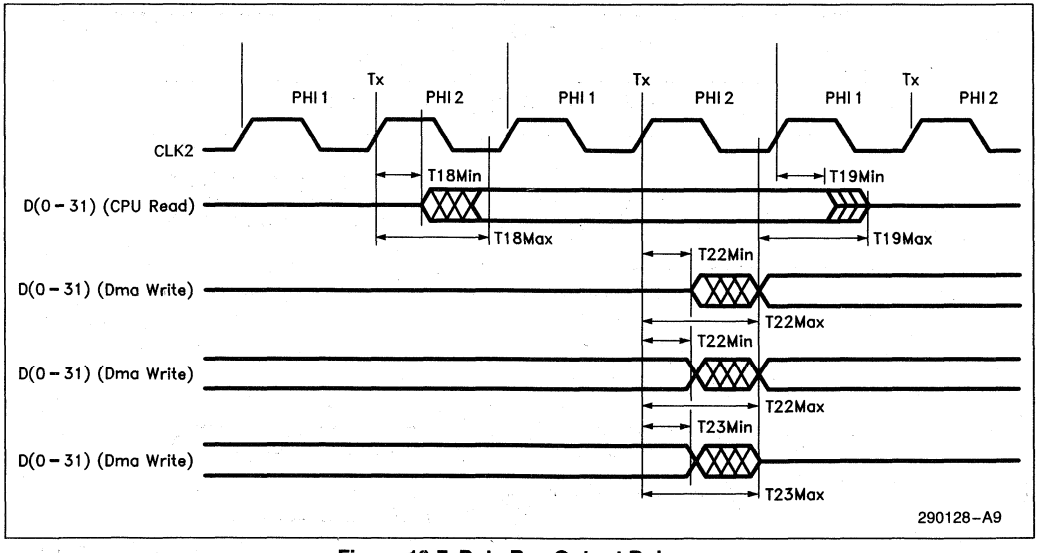


Figure 13-7. Data Bus Output Delays

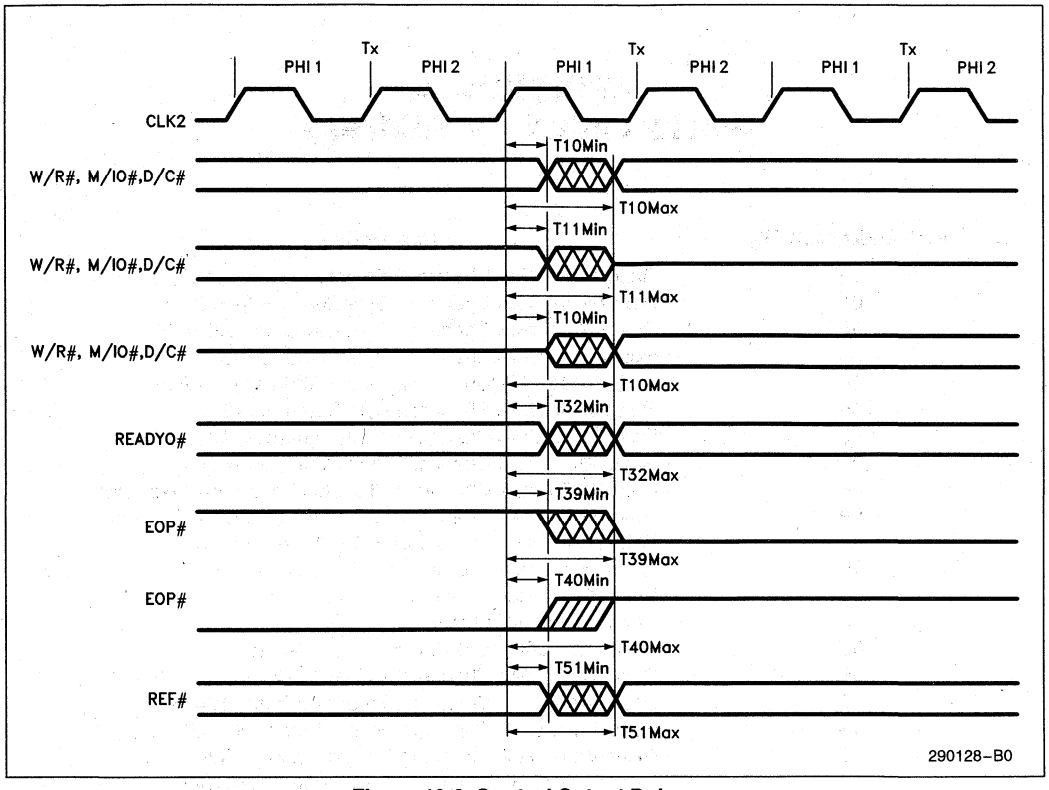


Figure 13-8. Control Output Delays

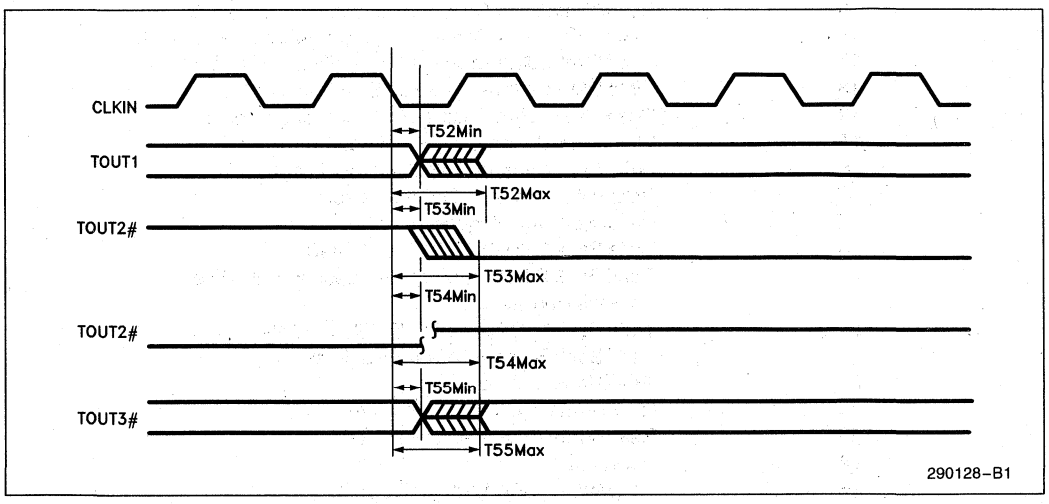


Figure 13-9. Timer Output Delays

APPENDIX A
Ports Listed by Address

Port Address (HEX)	Description
00	Read/Write DMA Channel 0 Target Address, A0–A15
01	Read/Write DMA Channel 0 Byte Count, B0–B15
02	Read/Write DMA Channel 1 Target Address, A0–A15
03	Read/Write DMA Channel 1 Byte Count, B0–B15
04	Read/Write DMA Channel 2 Target Address, A0–A15
05	Read/Write DMA Channel 2 Byte Count, B0–B15
06	Read/Write DMA Channel 3 Target Address, A0–A15
07	Read/Write DMA Channel 3 Byte Count, B0–B15
08	Read/Write DMA Channel 0–3 Status/Command I Register
09	Read/Write DMA Channel 0–3 Software Request Register
0A	Write DMA Channel 0–3 Set-Reset Mask Register
0B	Write DMA Channel 0–3 Mode Register I
0C	Write Clear Byte-Pointer FF
0D	Write DMA Master-Clear
0E	Write DMA Channel 0–3 Clear Mask Register
0F	Read/Write DMA Channel 0–3 Mask Register
10	Read/Write DMA Channel 0 Target Address, A24–A31
11	Read/Write DMA Channel 0 Byte Count, B16–B23
12	Read/Write DMA Channel 1 Target Address, A24–A31
13	Read/Write DMA Channel 1 Byte Count, B16–B23
14	Read/Write DMA Channel 2 Target Address, A24–A31
15	Read/Write DMA Channel 2 Byte Count, B16–B23
16	Read/Write DMA Channel 3 Target Address, A24–A31
17	Read/Write DMA Channel 3 Byte Count, B16–B23
18	Write DMA Channel 0–3 Bus Size Register
19	Read/Write DMA Channel 0–3 Chaining Register
1A	Write DMA Channel 0–3 Command Register II
1B	Write DMA Channel 0–3 Mode Register II
1C	Read/Write Refresh Control Register
1E	Reset Software Request Interrupt
20	Write Bank B ICW1, OCW2, or OCW3
	Read Bank B Poll, Interrupt Request or In-Service Status Register
21	Write Bank B ICW2, ICW3, ICW4 or OCW1
	Read Bank B Interrupt Mask Register
22	Read Bank B ICW2
28	Read/Write IRQ8 Vector Register
29	Read/Write IRQ9 Vector Register
2A	Reserved
2B	Read/Write IRQ11 Vector Register
2C	Read/Write IRQ12 Vector Register
2D	Read/Write IRQ13 Vector Register
2E	Read/Write IRQ14 Vector Register
2F	Read/Write IRQ15 Vector Register

APPENDIX A—Ports Listed by Address (Continued)

Port Address (HEX)	Description
30	Write Bank A ICW1, OCW2 or OCW3 Read Bank A Poll, Interrupt Request or In-Service Status Register
31	Write Bank A ICW2, ICW3, ICW4 or OCW1 Read Bank A Interrupt Mask Register
32	Read Bank A ICW2
38	Read/Write IRQ0 Vector Register
39	Read/Write IRQ1 Vector Register
3A	Read/Write IRQ1.5 Vector Register
3B	Read/Write IRQ3 Vector Register
3C	Read/Write IRQ4 Vector Register
3D	Reserved
3E	Reserved
3F	Read/Write IRQ7 Vector Register
40	Read/Write Counter 0 Register
41	Read/Write Counter 1 Register
42	Read/Write Counter 2 Register
43	Write Control Word Register I—Counter 0, 1, 2
44	Read/Write Counter 3 Register
45	Reserved
46	Reserved
47	Write Word Register II—Counter 3
61	Write Internal Control Port
64	Write CPU Reset Register (Data-1111XXX0H)
72	Read/Write Wait State Register 0
73	Read/Write Wait State Register 1
74	Read/Write Wait State Register 2
75	Read/Write Refresh Wait State Register
76	Reserved
77	Reserved
7D	Reserved
7E	Reserved
7F	Read/Write Relocation Register
80	Read/Write Internal Diagnostic Port 0
81	Read/Write DMA Channel 2 Target Address, A16–A23
82	Read/Write DMA Channel 3 Target Address, A16–A23
83	Read/Write DMA Channel 1 Target Address, A16–A23
87	Read/Write DMA Channel 0 Target Address, A16–A23
88	Read/Write Internal Diagnostic Port 1
89	Read/Write DMA Channel 6 Target Address, A16–A23
8A	Read/Write DMA Channel 7 Target Address, A16–A23
8B	Read/Write DMA Channel 5 Target Address, A16–A23
8F	Read/Write DMA Channel 4 Target Address, A16–A23

APPENDIX A—Ports Listed by Address (Continued)

Port Address (HEX)	Description
90	Read/Write DMA Channel 0 Requester Address, A0–A15
91	Read/Write DMA Channel 0 Requester Address, A16–A31
92	Read/Write DMA Channel 1 Requester Address, A0–A15
93	Read/Write DMA Channel 1 Requester Address, A16–A31
94	Read/Write DMA Channel 2 Requester Address, A0–A15
95	Read/Write DMA Channel 2 Requester Address, A16–A31
96	Read/Write DMA Channel 3 Requester Address, A0–A15
97	Read/Write DMA Channel 3 Requester Address, A16–A31
98	Read/Write DMA Channel 4 Requester Address, A0–A15
99	Read/Write DMA Channel 4 Requester Address, A16–A31
9A	Read/Write DMA Channel 5 Requester Address, A0–A15
9B	Read/Write DMA Channel 5 Requester Address, A16–A31
9C	Read/Write DMA Channel 6 Requester Address, A0–A15
9D	Read/Write DMA Channel 6 Requester Address, A16–A31
9E	Read/Write DMA Channel 7 Requester Address, A0–A15
9F	Read/Write DMA Channel 7 Requester Address, A16–A31
A0	Write Bank C ICW1, OCW2 or OCW3 Read Bank C Poll, Interrupt Request or In-Service Status Register
A1	Write Bank C ICW2, ICW3, ICW4 or OCW1 Read Bank C Interrupt Mask Register
A2	Read Bank C ICW2
A8	Read/Write IRQ16 Vector Register
A9	Read/Write IRQ17 Vector Register
AA	Read/Write IRQ18 Vector Register
AB	Read/Write IRQ19 Vector Register
AC	Read/Write IRQ20 Vector Register
AD	Read/Write IRQ21 Vector Register
AE	Read/Write IRQ22 Vector Register
AF	Read/Write IRQ23 Vector Register
C0	Read/Write DMA Channel 4 Target Address, A0–A15
C1	Read/Write DMA Channel 4 Byte Count, B0–B15
C2	Read/Write DMA Channel 5 Target Address, A0–A15
C3	Read/Write DMA Channel 5 Byte Count, B0–B15
C4	Read/Write DMA Channel 6 Target Address, A0–A15
C5	Read/Write DMA Channel 6 Byte Count, B0–B15
C6	Read/Write DMA Channel 7 Target Address, A0–A15
C7	Read/Write DMA Channel 7 Byte Count, B0–B15
C8	Read DMA Channel 4–7 Status/Command I Register
C9	Read/Write DMA Channel 4–7 Software Request Register
CA	Write DMA Channel 4–7 Set—Reset Mask Register
CB	Write DMA Channel 4–7 Mode Register I
CC	Reserved
CD	Reserved
CE	Write DMA Channel 4–7 Clear Mask Register
CF	Read/Write DMA Channel 4–7 Mask Register

APPENDIX A—Ports Listed by Address (Continued)

Port Address (HEX)	Description
D0	Read/Write DMA Channel 4 Target Address, A24–A31
D1	Read/Write DMA Channel 4 Byte Count, B16–B23
D2	Read/Write DMA Channel 5 Target Address, A24–A31
D3	Read/Write DMA Channel 5 Byte Count, B16–B23
D4	Read/Write DMA Channel 6 Target Address, A24–A31
D5	Read/Write DMA Channel 6 Byte Count, B16–B23
D6	Read/Write DMA Channel 7 Target Address, A24–A31
D7	Read/Write DMA Channel 7 Byte Count, B16–B23
D8	Write DMA Channel 4–7 Bus Size Register
D9	Read/Write DMA Channel 4–7 Chaining Register
DA	Write DMA Channel 4–7 Command Register II
DB	Write DMA Channel 4–7 Mode Register II

APPENDIX B
Ports Listed by Function

Port Address (HEX)	Description
	DMA CONTROLLER
0D	Write DMA Master-Clear
0C	Write DMA Clear Byte-Pointer FF
08	Read/Write DMA Channel 0–3 Status/Command I Register
C8	Read/Write DMA Channel 4–7 Status/Command I Register
1A	Write DMA Channel 0–3 Command Register II
DA	Write DMA Channel 4–7 Command Register II
0B	Write DMA Channel 0–3 Mode Register I
CB	Write DMA Channel 4–7 Mode Register I
1B	Write DMA Channel 0–3 Mode Register II
DB	Write DMA Channel 4–7 Mode Register II
09	Read/Write DMA Channel 0–3 Software Request Register
C9	Read/Write DMA Channel 4–7 Software Request Register
1E	Reset Software Request Interrupt
0E	Write DMA Channel 0–3 Clear Mask Register
CE	Write DMA Channel 4–7 Clear Mask Register
0F	Read/Write DMA Channel 0–3 Mask Register
CF	Read/Write DMA Channel 4–7 Mask Register
0A	Write DMA Channel 0–3 Set-Reset Mask Register
CA	Write DMA Channel 4–7 Set-Reset Mask Register
18	Write DMA Channel 0–3 Bus Size Register
D8	Write DMA Channel 4–7 Bus Size Register
19	Read/Write DMA Channel 0–3 Chaining Register
D9	Read/Write DMA Channel 4–7 Chaining Register
00	Read/Write DMA Channel 0 Target Address, A0–A15
87	Read/Write DMA Channel 0 Target Address, A16–A23
10	Read/Write DMA Channel 0 Target Address, A24–A31
01	Read/Write DMA Channel 0 Byte Count, B0–B15
11	Read/Write DMA Channel 0 Byte Count, B16–B23
90	Read/Write DMA Channel 0 Requester Address, A0–A15
91	Read/Write DMA Channel 0 Requester Address, A16–A31
02	Read/Write DMA Channel 1 Target Address, A0–A15
83	Read/Write DMA Channel 1 Target Address, A16–A23
12	Read/Write DMA Channel 1 Target Address, A24–A31
03	Read/Write DMA Channel 1 Byte Count, B0–B15
13	Read/Write DMA Channel 1 Byte Count, B16–B23
92	Read/Write DMA Channel 1 Requester Address, A0–A15
93	Read/Write DMA Channel 1 Requester Address, A16–A31

APPENDIX B—Ports Listed by Function (Continued)

Port Address (HEX)	Description
	DMA CONTROLLER
04	Read/Write DMA Channel 2 Target Address, A0–A15
81	Read/Write DMA Channel 2 Target Address, A16–A23
14	Read/Write DMA Channel 2 Target Address, A24–A31
05	Read/Write DMA Channel 2 Byte Count, B0–B15
15	Read/Write DMA Channel 2 Byte Count, B16–B23
94	Read/Write DMA Channel 2 Requester Address, A0–A15
95	Read/Write DMA Channel 2 Requester Address, A16–A31
06	Read/Write DMA Channel 3 Target Address, A0–A15
82	Read/Write DMA Channel 3 Target Address, A16–A23
16	Read/Write DMA Channel 3 Target Address, A24–A31
07	Read/Write DMA Channel 3 Byte Count, B0–B15
17	Read/Write DMA Channel 3 Byte Count, B16–B23
96	Read/Write DMA Channel 3 Requester Address, A0–A15
97	Read/Write DMA Channel 3 Requester Address, A16–A31
C0	Read/Write DMA Channel 4 Target Address, A0–A15
8F	Read/Write DMA Channel 4 Target Address, A16–A23
D0	Read/Write DMA Channel 4 Target Address, A24–A31
C1	Read/Write DMA Channel 4 Byte Count, B0–B15
D1	Read/Write DMA Channel 4 Byte Count, B16–B23
98	Read/Write DMA Channel 4 Requester Address, A0–A15
99	Read/Write DMA Channel 4 Requester Address, A16–A31
C2	Read/Write DMA Channel 5 Target Address, A0–A15
8B	Read/Write DMA Channel 5 Target Address, A16–A23
D2	Read/Write DMA Channel 5 Target Address, A24–A31
C3	Read/Write DMA Channel 5 Byte Count, B0–B15
D3	Read/Write DMA Channel 5 Byte Count, B16–B23
9A	Read/Write DMA Channel 5 Requester Address, A0–A15
9B	Read/Write DMA Channel 5 Requester Address, A16–A31
C4	Read/Write DMA Channel 6 Target Address, A0–A15
89	Read/Write DMA Channel 6 Target Address, A16–A23
D4	Read/Write DMA Channel 6 Target Address, A24–A31
C5	Read/Write DMA Channel 6 Byte Count, B0–B15
D5	Read/Write DMA Channel 6 Byte Count, B16–B23
9C	Read/Write DMA Channel 6 Requester Address, A0–A15
9D	Read/Write DMA Channel 6 Requester Address, A16–A31
C6	Read/Write DMA Channel 7 Target Address, A0–A15
8A	Read/Write DMA Channel 7 Target Address, A16–A23
D6	Read/Write DMA Channel 7 Target Address, A24–A31
C7	Read/Write DMA Channel 7 Byte Count, B0–B15
D7	Read/Write DMA Channel 7 Byte Count, B16–B23
9E	Read/Write DMA Channel 7 Requester Address, A0–A15
9F	Read/Write DMA Channel 7 Requester Address, A16–A31

APPENDIX B—Ports Listed by Function (Continued)

Port Address (HEX)	Description
	INTERRUPT CONTROLLER
20	Write Bank B ICW1, OCW2, or OCW3 Read Bank B Poll, Interrupt Request or In-Service Status Register
21	Write Bank B ICW2, ICW3, ICW4 or OCW1 Read Bank B Interrupt Mask Register
22	Read Bank B ICW2
28	Read/Write IRQ8 Vector Register
29	Read/Write IRQ9 Vector Register
2A	Reserved
2B	Read/Write IRQ11 Vector Register
2C	Read/Write IRQ12 Vector Register
2D	Read/Write IRQ13 Vector Register
2E	Read/Write IRQ14 Vector Register
2F	Read/Write IRQ15 Vector Register
A0	Write Bank C ICW1, OCW2 or OCW3 Read Bank C Poll, Interrupt Request or In-Service Status Register
A1	Write Bank C ICW2, ICW3, ICW4 or OCW1 Read Bank C Interrupt Mask Register
A2	Read Bank C ICW2
A8	Read/Write IRQ16 Vector Register
A9	Read/Write IRQ17 Vector Register
AA	Read/Write IRQ18 Vector Register
AB	Read/Write IRQ19 Vector Register
AC	Read/Write IRQ20 Vector Register
AD	Read/Write IRQ21 Vector Register
AE	Read/Write IRQ22 Vector Register
AF	Read/Write IRQ23 Vector Register
30	Write Bank A ICW1, OCW2 or OCW3 Read Bank A Poll, Interrupt Request oor In-Service Status Register
31	Write Bank A ICW2, ICW3, ICW4 or OCW1 Read Bank A Interrupt Mask Register
32	Read Bank A ICW2
38	Read/Write IRQ0 Vector Register
39	Read/Write IRQ1 Vector Register
3A	Read/Write IRQ1.5 Vector Register
3B	Read/Write IRQ3 Vector Register
3C	Read/Write IRQ4 Vector Register
3D	Reserved
3E	Reserved
3F	Read/Write IRQ7 Vector Register

APPENDIX B—Ports Listed by Function (Continued)

Port Address (HEX)	Description
	PROGRAMMABLE INTERVAL TIMER
40	Read/Write Counter 0 Register
41	Read/Write Counter 1 Register
42	Read/Write Counter 2 Register
43	Write Control Word Register I—Counter 0, 1, 2
44	Read/Write Counter 3 Register
47	Write Word Register II—Counter 3
	CPU RESET
64	Write CPU Reset Register (Data-1111XXX0H)
	WAIT STATE GENERATOR
72	Read/Write Wait State Register 0
73	Read/Write Wait State Register 1
74	Read/Write Wait State Register 2
75	Read/Write Refresh Wait State Register
	DRAM REFRESH CONTROLLER
1C	Read/Write Refresh Control Register
	INTERNAL CONTROL AND DIAGNOSTIC PORTS
61	Write Internal Control Port
80	Read/Write Internal Diagnostic Port 0
88	Read/Write Internal Diagnostic Port 1
	RELOCATION REGISTER
7F	Read/Write Relocation Register
	INTEL RESERVED PORTS
2A	Reserved
3D	Reserved
3E	Reserved
45	Reserved
46	Reserved
76	Reserved
77	Reserved
7D	Reserved
7E	Reserved
CC	Reserved
CD	Reserved

APPENDIX C
Pin Descriptions

The 82380 provides all of the signals necessary to interface it to an 80386 processor. It has separate 32-bit address and data buses. It also has a set of control signals to support operation as a bus master or a bus slave. Several special function signals exist on the 82380 for interfacing the system support peripherals to their respective system counterparts. Following are the definitions of the individual pins of the 82380. These brief descriptions are provided as a reference. Each signal is further defined within the sections which describe the associated 82380 function.

A2-A31 I/O ADDRESS BUS

This is the 32-bit address bus. The addresses are doubleword memory and I/O addresses. These are three-state signals which are active only during Master mode. The address lines should be connected directly to the 80386's local bus.

BE0# I/O BYTE-ENABLE 0

BE0# active indicates that data bits D0–D7 are being accessed or are valid. It is connected directly to the 80386's BE0#. The byte enable signals are active outputs when the 82380 is in the Master mode.

BE1# I/O BYTE-ENABLE 1

BE1# active indicates that data bits D8–D15 are being accessed or are valid. It is connected directly to the 80386's BE1#. The byte enable signals are active only when the 82380 is in the Master mode.

BE2# I/O BYTE-ENABLE 2

BE2# active indicates that data bits D15–D23 are being accessed or are valid. It is connected directly to the 80386's BE2#. The byte enable signals are active only when the 82380 is in the Master mode.

BE3# I/O BYTE-ENABLE 3

BE3# active indicates that data bits D24–D31 are being accessed or are valid. The byte enable signals are active only when the 82380 is in the Master mode. This pin should be connected directly to the 80386's BE3#. This pin is used for factory testing and must be low during reset. The 80386 drives BE3# low during reset.

D0–D31 I/O DATA BUS

This is the 32-bit data bus. These pins are active outputs during interrupt acknowledges, during Slave accesses, and when the 82380 is in the Master mode.

CLK2 I PROCESSOR CLOCK

This pin must be connected to CLK2. The 82380 monitors the phase of this clock in order to remain synchronized with the 80386. This clock drives all of the internal synchronous circuitry.

D/C# I/O DATA/CONTROL

D/C# is used to distinguish between 80386 control cycles and DMA or 80386 data access cycles. It is active as an output only in the Master mode.

W/R# I/O WRITE/READ

W/R# is used to distinguish between write and read cycles. It is active as an output only in the Master mode.

M/IO# I/O MEMORY/IO

M/IO# is used to distinguish between memory and IO accesses. It is active as an output only in the Master mode.

ADS# I/O ADDRESS STATUS

This signal indicates presence of a valid address on the address bus. It is active as output only in the Master mode. ADS# is active during the first T-state where addresses and control signals are valid.

NA# I NEXT ADDRESS

Asserted by a peripheral or memory to begin a pipelined address cycle. This pin is monitored only while the 82380 is in the Master mode. In the Slave mode, pipelining is determined by the current and past status of the ADS# and READY# signals.

HOLD O HOLD REQUEST

This is an active-high signal to the 80386 to request control of the system bus. When control is granted, the 80386 activates the hold acknowledge signal (HLDA).

HLDA I HOLD ACKNOWLEDGE

This input signal tells the DMA controller that the 80386 has relinquished control of the system bus to the DMA controller.

DREQ (0–3, 5–7) I DMA REQUEST

The DMA Request inputs monitor requests from peripherals requiring DMA service. Each of the eight DMA channels has one DREQ input. These active-high inputs are internally synchronized and prioritized. Upon reset, channel 0 has the highest priority and channel 7 the lowest.

DREQ4/IRQ9# I DMA/INTERRUPT REQUEST

This is the DMA request input for channel 4. It is also connected to the interrupt controller via interrupt request 9. This internal connection is available for DMA channel 4 only. The interrupt input is active low and can be programmed as either edge of level triggered. Either function can be masked by the appropriate mask register. Priorities of the DMA channel and the interrupt request are not related but follow the rules of the individual controllers.

Note that this pin has a weak internal pull-up. This causes the interrupt request to be inactive, but the DMA request will be active if there is no external connection made. Most applications will require that either one or the other of these functions be used, but not both. For this reason, it is advised that DMA channel 4 be used for transfers where a software request is more appropriate (such as memory-to-memory transfers). In such an application, DREQ4 can be masked by software, freeing IRQ9# for other purposes.

EOP# I/O END OF PROCESS

As an output, this signal indicates that the current Requester access is the last access of the currently operating DMA channel. It is activated when Terminal Count is reached. As an input, it signals the DMA channel to terminate the current buffer and proceed to the next buffer, if one is available. This signal may be programmed as an asynchronous or synchronous input.

EOP# must be connected to a pull-up resistor. This will prevent erroneous external requests for termination of a DMA process.

EDACK (0–2) O ENCODED DMA ACKNOWLEDGE

These signals contain the encoded acknowledgement of a request for DMA service by a peripheral. The binary code formed by the three signals indicates which channel is active. Channel 4 does not have a DMA acknowledge. The inactive state is indicated by the code 100. During a Requester access, EDACK presents the code for the active DMA channel. During a Target access, EDACK presents the inactive code 100.

IRQ (11–23)# I INTERRUPT REQUEST

These are active low interrupt request inputs. The inputs can be programmed to be edge or level sensitive. Interrupt priorities are programmable as either fixed or rotating. These inputs have weak internal pull-up resistors. Unused interrupt request inputs should be tied inactive externally.

INT O INTERRUPT OUT

INT signals the 80386 that an interrupt request is pending.

CLKIN I TIMER CLOCK INPUT

This is the clock input signal to all of the 82380's programmable timers. It is independent of the system clock input (CLK2).

TOUT1/REF# O TIMER 1 OUTPUT/REFRESH

This pin is software programmable as either the direct output of Timer 1, or as the indicator of a refresh cycle in progress. As REF#, this signal is active during the memory read cycle which occurs during refresh.

TOUT2#/IRQ3# I/O TIMER 2 OUTPUT/INTERRUPT REQUEST3

This is the inverted output of Timer 2. It is also connected directly to interrupt request 3. External hardware can use IRQ3# if Timer 2 is programmed as OUT=0 (TOUT2#=1)

TOUT3# O TIMER 3 OUTPUT

This is the inverted output of Timer 3.

READY# I READY INPUT

This active-low input indicates to the 82380 that the current bus cycle is complete. READY is sampled by the 82380 both while it is in the Master mode, and while it is in the Slave mode.

WSC (0–1) I WAIT STATE CONTROL

WSC0 AND WSC1 are inputs used by the Wait-State Generator to determine the number of wait states required by the currently accessed memory or I/O. The binary code on these ins, combined with the M/IO# signal, selects an internal register in which a wait-state count is stored. The combination WSC = 11 disables the wait-state generator.

READYO# O READY OUTPUT

This is the synchronized output of the wait-state generator. It is also valid during 80386 accesses to the 82380 in the Slave Mode when the 82380 requires wait states. READYO# should feed directly the 80386's READY# input.

RESET I RESET

This synchronous input serves to initialize the state of the 82380 and provides basis for the CPURST output. RESET must be held active for at least 15 CLK2 cycles in order to guarantee the state of the 82380. After Reset, the 82380 is in the Slave mode with all outputs except timers and interrupts in their inactive states. The state of the timers and interrupt controller must be initialized through software. This input must be active for the entire time required by the 80386 to guarantee proper reset.

CPURST O CPU RESET

CPURST provides a synchronized reset signal for the CPU. It is activated in the event of a software reset command, an 80386 shut-down detect, or a hardware reset via the RESET pin. The 82380 holds CPURST active for 62 clocks in response to either a software reset command or a shut-down detection. Otherwise CPURST reflects the RESET input.

V_{CC} +5V input power
V_{SS} Ground

Table C-1. Wait-State Select Inputs

Port Address	Wait-State Registers D7 D4	D3 D0	Select Inputs WSC1	WSC0
72H	Memory 0	I/O 0	0	0
73H	Memory 1	I/O 1	0	1
74H	Memory 2	I/O 2	1	1
	DISABLED		1	1
M/IO#	1	0		

APPENDIX D
82380 System Notes

82380 TIMER UNIT SYSTEM NOTES

The 82380 DMA controller with Integrated System Peripherals is functionally inconsistent with the data sheet. This document explains the behavior of the 82380 Timer Unit and outlines subsequent limitations of the timer unit. This document also provides recommended workarounds.

1.0 WRITE CYCLES TO THE 82380 TIMER UNIT

This errata applies only to SLAVE WRITE cycles to the 82380 timer unit. During these cycles, the data being written into the 82380 timer unit may be corrupted if CLKIN is not inhibited during a certain "window" of the write cycle.

1.1 Description

Please refer to Figure 1.

During write cycles to the 82380 timer unit, the 82380 translates the 386DX interface signals such as ADS#, W/R#, M/IO#, and D/C# into several internal signals that control the operation of the internal sub-blocks (e.g., Timer Unit).

The 82380 timer unit is controlled by such internal signals. These internal signals are generated and sampled with respect to two separate clock signals: CLK2 (the system clock) and CLKIN (the 82380 timer unit clock).

Since the CLKIN and CLK2 clock signals are used internally to generate control signals for the interface to the timer unit, some timing parameters must be met in order for the interface logic to function properly.

Those timing parameters are met by inhibiting the CLKIN signal for a specific window during Write Cycles to the 82380 Timer Unit.

The CLKIN signal must be inhibited using external logic, as the GATE function of the 82380 timer unit is not guaranteed to totally inhibit CLKIN.

1.2 Consequences

This CLKIN inhibit circuitry guarantees proper write cycles to the 82380 timer unit.

Without this solution, write cycles to the 82380 timer unit could place corrupted data into the timer unit registers. This, in turn, could yield inaccurate results and improper timer operation.

The proposed solution would involve a hardware modification for existing systems.

1.3 Solution

A timing waveform (Figure 2) shows the specific window during which CLKIN must be inhibited. Please note that CLKIN must only be inhibited during the window shown in Figure 2. This window is defined by two AC timing parameters:

t_a = 9 ns

t_b = 28 ns

The proposed solution provides a certain amount of system "guardband" to make sure that this window is avoided.

PAL equations for a suggested workaround are also included. Please refer to the comments in the PAL codes for stated assumptions of this particular workaround. A state diagram (Figure 3) is provided to help clarify how this PAL is designed.

Figure 4 shows how this PAL would fit into a system workaround. In order to show the effect of this workaround on the CLKIN signal, Figure 5 shows how CLKIN is inhibited. Note that you must still meet the CLKIN AC timing parameters (e.g., t_{47} (min), t_{48} (min)) in order for the timer unit to function properly.

Please note that this workaround has not been tested. It is provided as a suggested solution. Actual solutions will vary from system to system.

1.4 Long Term Plans

Intel has no plans to fix this behavior in the 82380 timer unit.

```
module Timer_82380_Fix
flag '-r2','-q2','-fl', '-t4', '-wl,3,6,5,4,16,7,12,17,18,15,14'
title '82380 Timer Unit CLKIN
       INHIBIT signal PAL Solution '
Timer_Unit_Fix device 'P16R6';

"This PAL inhibits the CLKIN signal (that comes from an oscillator)
"during Slave Writes to the 82380 Timer unit.
"
"ASSUMPTION:   This PAL assumes that an external system address
"              decoder provides a signal to indicate that an 82380
"              Timer Unit access is taking place. This input
"              signal is called TMR in this PAL. This PAL also
"              assumes that this TMR signal occurs during a
"              specific T-State. Please see Figure 3 of this
"              document to see when this signal is expected to
"              be active by this PAL.
"
"
"NOTE:         This PAL does not support pipelined 82380 SLAVE
"              cycles.
"
"(c) Intel Corporation 1989. This PAL is provided as a proposed
"method of solving a certain 82380 Timer Unit problem. This PAL
"has not been tested or validated. Please validate this solution
"for your system and application.
"
```

```
"Input Pins"

CLK2            pin      1; "System Clock
RESET           pin      2; "Microprocessor RESET signal
TMR             pin      3; "Input from Address Decoder, indicating
                            "an access to the timer unit of the
                            "82380.
!RDY            pin      4; "End of Cycle indicator
!ADS            pin      5; "Address and control strobe
CLK             pin      6; "PHI2 clock
W_R             pin      7; "Write/Read Signal"
nc1             pin      8; "No Connect 0"
nc3             pin      9; "No Connect 1"
GNDa            pin     10; "Tied to ground, documentation only
GNDb            pin     11; "Output enable, documentation only
CLKIN_IN        pin     12; "Input-CLKIN directly from oscillator

"Output Pins"

Q_0             pin     18; "Internal signal only, fed back to
                            "PAL logic"
CLKIN_OUT       pin     17; "CLKIN signal fed to 82380 Timer Unit
INHIBIT         pin     16; "CLKIN Inhibit signal
S0              pin     15; "Unused State Indicator Pin
S1              pin     14; "Unused State Indicator Pin

"Declarations"

Valid_ADS = ADS & CLK      ; "ADS# sampled in PHI1 of 386DX T-State

Valid_RDY = RDY & CLK      ; "RDY# sampled in PHI1 of 386DX T-State

Timer_Acc = TMR & CLK      ; "Timer Unit Access, as provided by
                             "external Address Decoder "

State_Diagram [INHIBIT, S1, S0]

state 000:      if RESET then 000
                else if Valid_ADS & W_R then 001
                else 000;

state 001:      if RESET then 000
                else if Timer_Acc then 010
                else if !Timer_Acc then 000
                else 001;

state 010:      if RESET then 000
                else if CLK then 110
                else 010;

state 110:      if RESET then 000
                else if CLK then 111
                else 110;
```

5

```
state 111:      if RESET then 000
                else if CLK then 011
                else 111;

state 011:      if RESET then 000
                else if Valid_RDY then 000
                else 011;

state 100:      if RESET then 000
                else 000;

state 101:      if RESET then 000
                else 000;

EQUATIONS

Q_0 := CLKIN_IN   ; "Latched incoming clock. This signal is used
                    "internally to feed into the MUX-ing logic"

CLKIN_OUT := (INHIBIT & CLKIN_OUT & !RESET)
           +(!INHIBIT & Q_0 & !RESET);

                        "Equation for CLKIN_OUT. This
                        "feeds directly to the 82380 Timer Unit."

END


Page 1

ABEL(tm) 3.10 - Document Generator 30-June 89 03:17
PM
82380 Timer Unit CLKIN
      INHIBIT signal PAL Solution
Equations for Module Timer_82380_Fix

Device Timer_Unit_Fix

- Reduced Equations:

    !INHIBIT := (!CLK & !INHIBIT # CLK & S0 # RESET # !S1);

    !S1 := (RESET
          # INHIBIT & !S1
          # CLK & !INHIBIT & !~RDY & S0 & S1
          # !CLK & !S1
          # !S1 & !TMR
          # !S0 & !S1);

    !S0 := (RESET
          # INHIBIT & !S1
          # CLK & !INHIBIT & !~RDY & S1
          # !CLK & !S0
          # !INHIBIT & !S0 & S1
          # S0 & !S1
          # !S1 & !W_R
          # ~ADS & !S1);
```

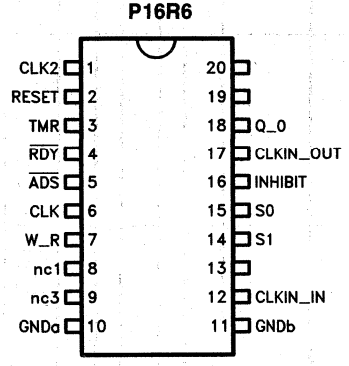
```
    !Q_0 := ( !CLKIN_IN);

    !CLKIN_OUT := (RESET # !CLKIN_OUT & INHIBIT # !INHIBIT & !Q_0);

Page 2

ABEL(tm) 3.10 - Document Generator 30-June 89 03:17
PM
82380 Timer Unit CLKIN
      INHIBIT signal PAL Solution
Chip diagram for Module Timer_82380_Fix

Device Timer_Unit_Fix
```




```
end of module Timer_82380_Fix
```

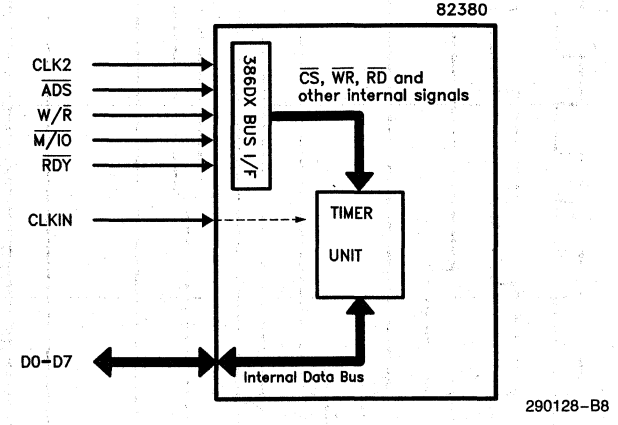


Figure 1. Translation of 386DX Signals to Internal 82380 Timer Unit Signals

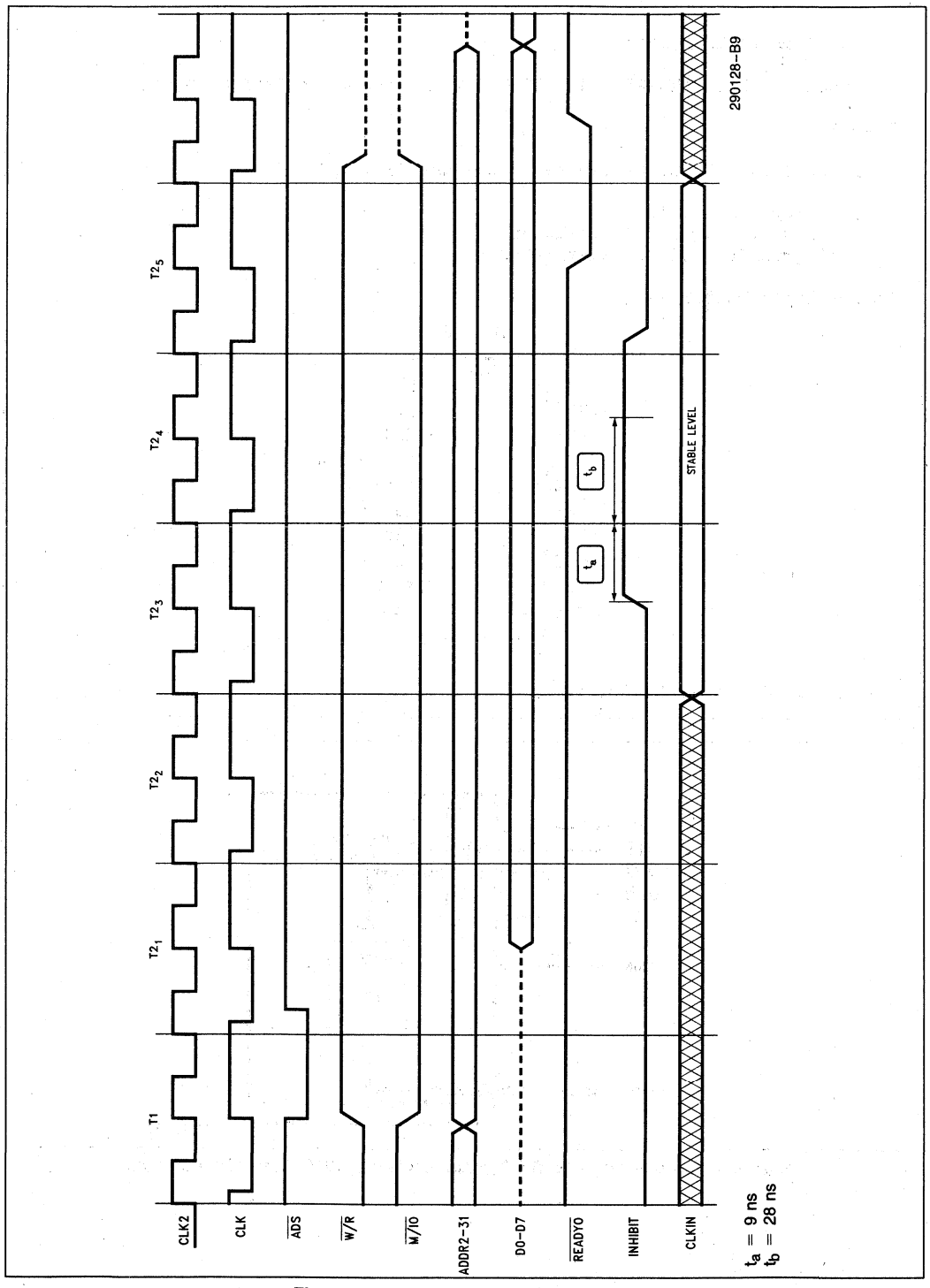


Figure 2. 82380 Timer Unit Write Cycle

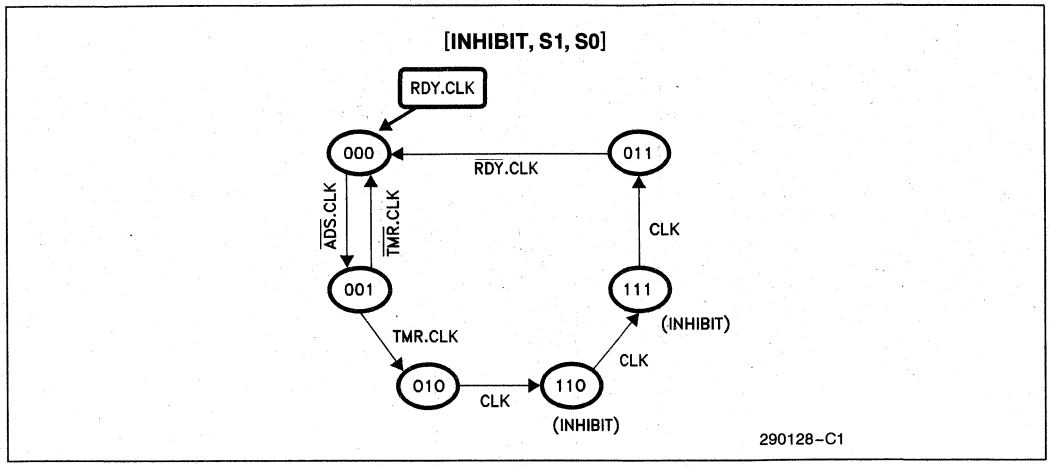
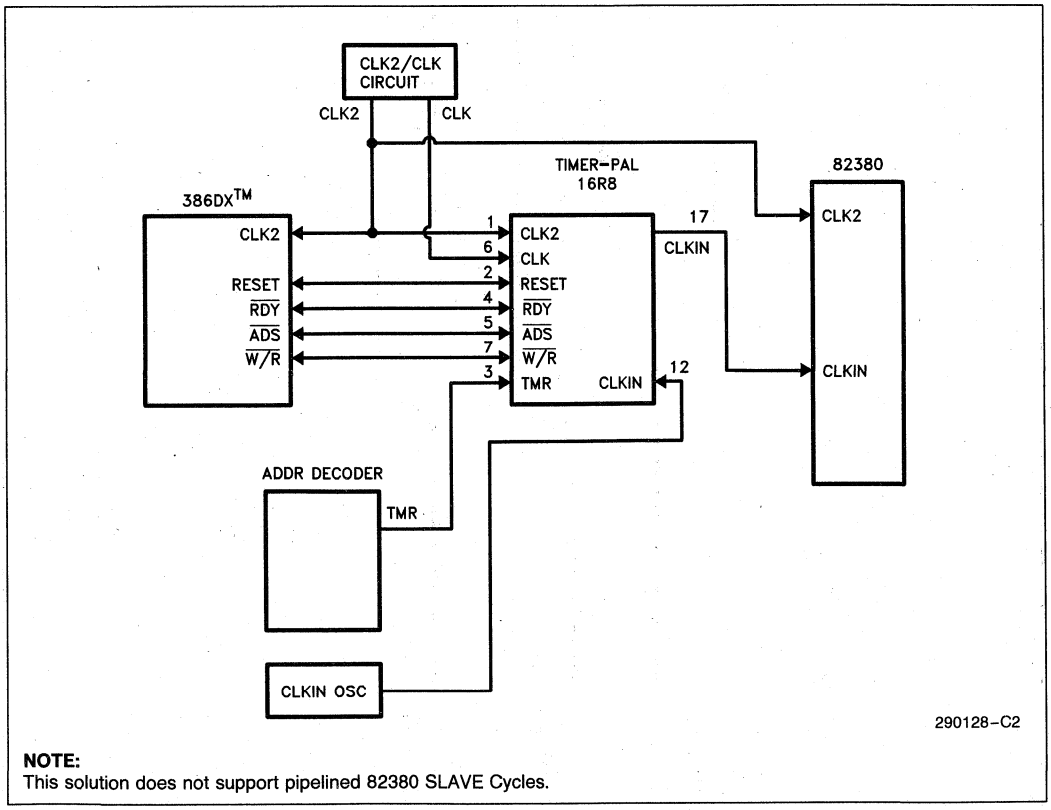


Figure 3. State Diagram for Inhibit Signal



NOTE:
This solution does not support pipelined 82380 SLAVE Cycles.

Figure 4. System with 82380 Timer Unit "Inhibit" Circuitry

5

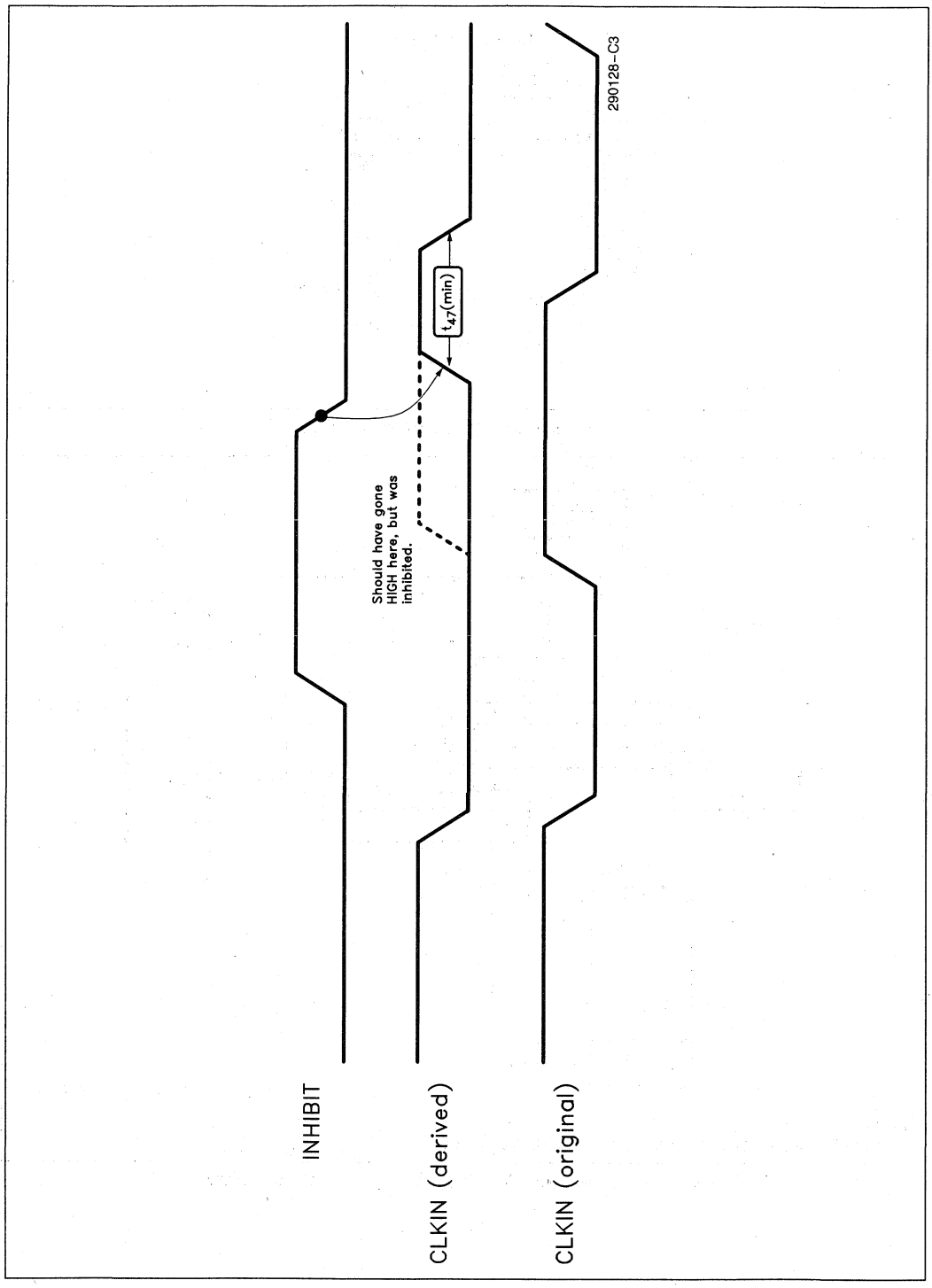


Figure 5(a). Inhibited CLKIN in an 82380 Timer Unit and CLKIN Minimum HIGH Time

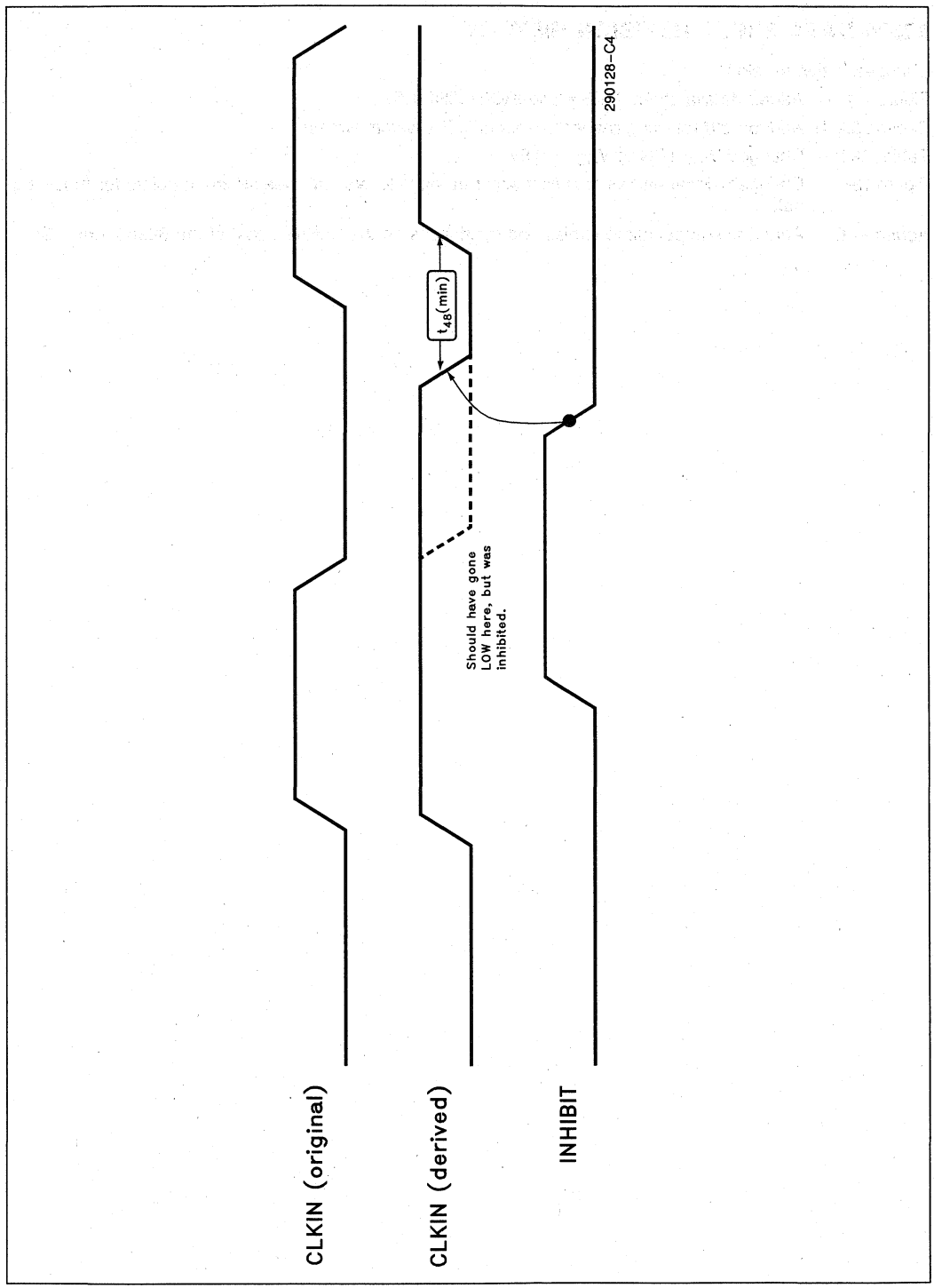


Figure 5(b). Inhibited CLKIN in an 82380 Timer Unit and CLKIN Minimum LOW Time

82380 DATA SHEET REVISION HISTORY

Changes in this revision:

Figure 4-1: Added details about IRQ3# and IRQ2#/IRQ1.5#.

Section 5.2.1: Added note referring reader to Appendix D (System Notes).

Table 13-2: Changed V_{IHC} MIN to $V_{CC} - 0.8V$.

Figure 13-1: Changed signal names to reflect accurate drive levels and measurement points for those signals.

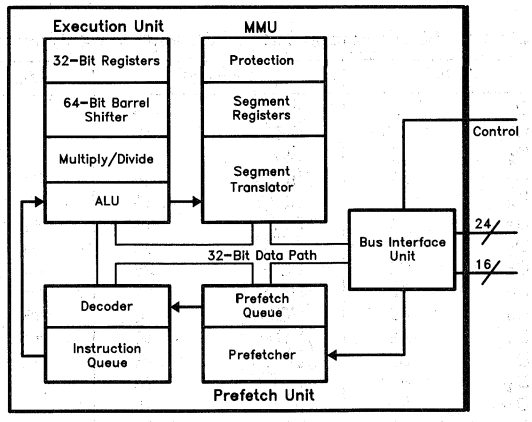
Appendix D: Added this appendix to explain the restrictions on the CLKIN signal of the 82380 Timer Unit.

376™ HIGH PERFORMANCE 32-BIT EMBEDDED PROCESSOR

- **Full 32-Bit Internal Architecture**
  - **8-, 16-, 32-Bit Data Types**
  - **8 General Purpose 32-Bit Registers**
  - **Extensive 32-Bit Instruction Set**
- **High Performance 16-Bit Data Bus**
  - **16 or 20 MHz CPU Clock**
  - **Two-Clock Bus Cycles**
  - **16 Mbytes/Sec Bus Bandwidth**
- **16 Mbyte Physical Memory Size**
- **High Speed Numerics Support with the 80387SX**
- **Low System Cost with the 82370 Integrated System Peripheral**
- **On-Chip Debugging Support Including Break Point Registers**
- **Complete Intel Development Support**
  - **C, PL/M, Assembler**
  - **ICE™-376, In-Circuit Emulator**
  - **iRMK™ Real Time Kernel**
  - **iSDM™ Debug Monitor**
  - **DOS Based Debug**
- **Extensive Third-Party Support:**
  - **Languages: C, Pascal, FORTRAN, BASIC and ADA***
  - **Hosts: VMS*, UNIX*, MS-DOS*, and Others**
  - **Real-Time Kernels**
- **High Speed CHMOS IV Technology**
- **Available in 100 Pin Plastic Quad Flat-Pack Package and 88-Pin Pin Grid Array**
  (See Packaging Outlines and Dimensions #231369)

INTRODUCTION

The 376 32-bit embedded processor is designed for high performance embedded systems. It provides the performance benefits of a highly pipelined 32-bit internal architecture with the low system cost associated with 16-bit hardware systems. The 80376 processor is based on the 80386 and offers a high degree of compatibility with the 80386. All 80386 32-bit programs not dependent on paging can be executed on the 80376 and all 80376 programs can be executed on the 80386. All 32-bit 80386 language translators can be used for software development. With proper support software, any 80386-based computer can be used to develop and test 80376 programs. In addition, any 80386-based PC-AT* compatible computer can be used for hardware prototyping for designs based on the 80376 and its companion product the 82370.



240182-48

80376 Microarchitecture

Intel, iRMK, ICE, 376, 386, Intel386, iSDM, Intel1376 are trademarks of Intel Corp.
*UNIX is a registered trademark of AT&T.
ADA is a registered trademark of the U.S. Government, Ada Joint Program Office.
PC-AT is a registered trademark of IBM Corporation.
VMS is a trademark of Digital Equipment Corporation.
MS-DOS is a trademark of MicroSoft Corporation.

December 1990
Order Number: 240182-004

1.0 PIN DESCRIPTION

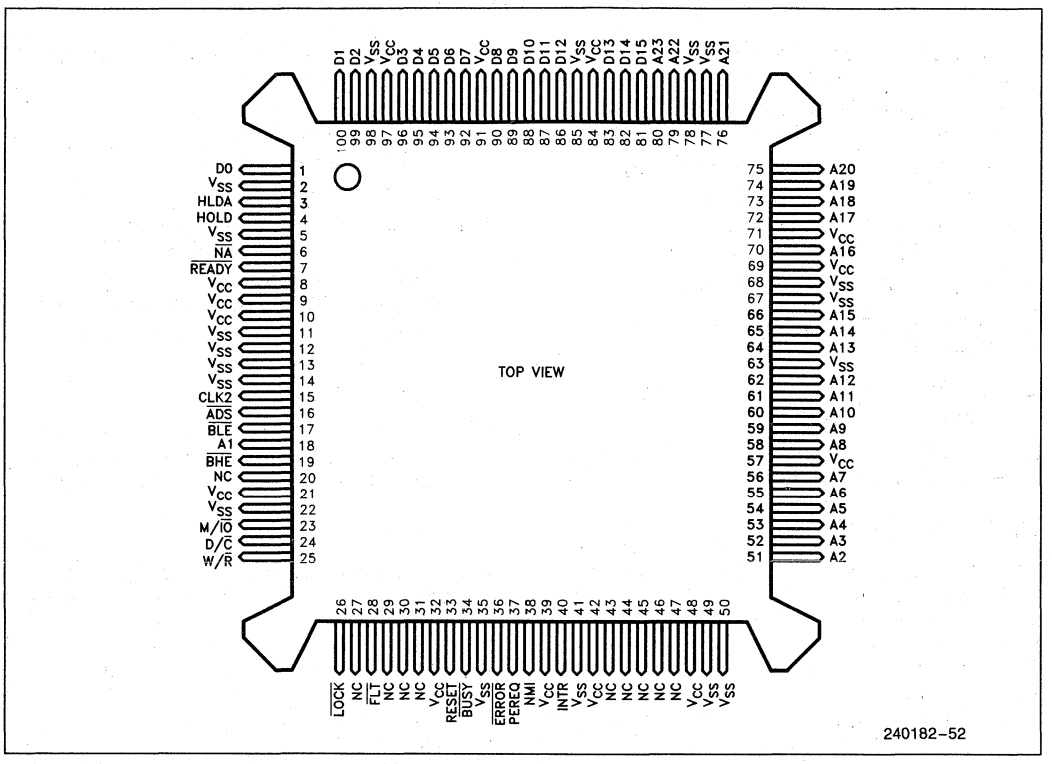
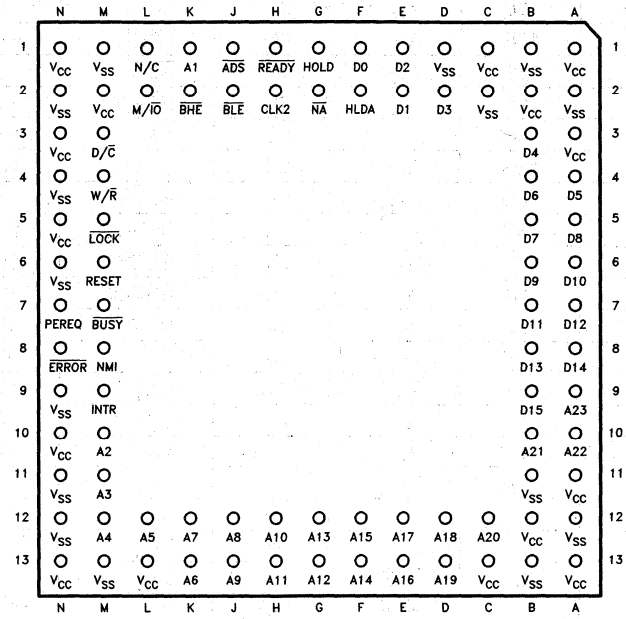


Figure 1.1. 80376 100-Pin Quad Flat-Pack Pin Out (Top View)

Table 1.1. 100-Pin Plastic Quad Flat-Pack Pin Assignments

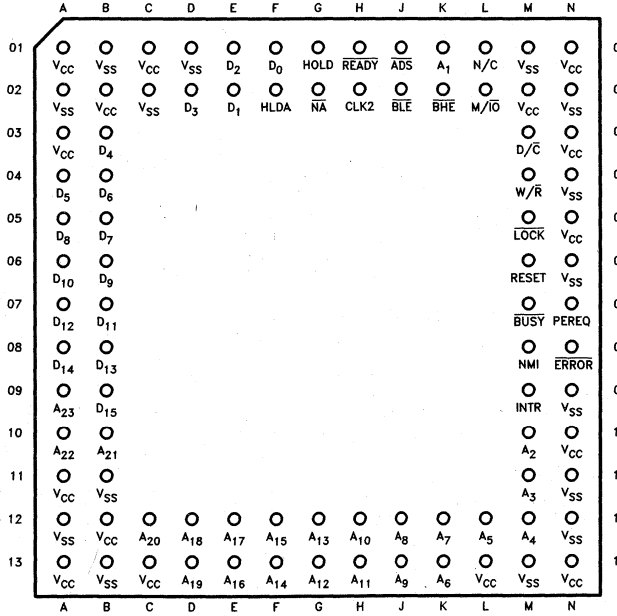
Address		Data		Control		N/C	V_{CC}	V_{SS}
A_1	18	D_0	1	$\overline{ADS}$	16	20	8	2
A_2	51	D_1	100	$\overline{BHE}$	19	27	9	5
A_3	52	D_2	99	$\overline{BLE}$	17		10	11
A_4	53	D_3	96	$\overline{BUSY}$	34	29	21	12
A_5	54	D_4	95	CLK2	15	30	32	13
A_6	55	D_5	94	D/$\overline{C}$	24	31	39	14
A_7	56	D_6	93	$\overline{ERROR}$	36	43	42	22
A_8	58	D_7	92	$\overline{FLT}$	28	44	48	35
A_9	59	D_8	90	HLDA	3	45	57	41
A_{10}	60	D_9	89	HOLD	4	46	69	49
A_{11}	61	D_{10}	88	INTR	40	47	71	50
A_{12}	62	D_{11}	87	$\overline{LOCK}$	26		84	63
A_{13}	64	D_{12}	86	M/$\overline{IO}$	23		91	67
A_{14}	65	D_{13}	83	$\overline{NA}$	6		97	68
A_{15}	66	D_{14}	82	NMI	38			77
A_{16}	70	D_{15}	81	PEREQ	37			78
A_{17}	72			$\overline{READY}$	7			85
A_{18}	73			RESET	33			98
A_{19}	74			W/$\overline{R}$	25			
A_{20}	75							
A_{21}	76							
A_{22}	79							
A_{23}	80							

Top View
(Component Side)



240182-49

Bottom View
(Pin Side)



240182-2

Figure 1.2. 80376 88-Pin Grid Array Pin Out

Table 1.2. 88-Pin Grid Array Pin Assignments

Pin	Label	Pin	Label	Pin	Label	Pin	Label
2H	CLK2	12D	A_{18}	2L	M/$\overline{IO}$	11A	V_{CC}
9B	D_{15}	12E	A_{17}	5M	$\overline{LOCK}$	13A	V_{CC}
8A	D_{14}	13E	A_{16}	1J	$\overline{ADS}$	13C	V_{CC}
8B	D_{13}	12F	A_{15}	1H	$\overline{READY}$	13L	V_{CC}
7A	D_{12}	13F	A_{14}	2G	$\overline{NA}$	1N	V_{CC}
7B	D_{11}	12G	A_{13}	1G	HOLD	13N	V_{CC}
6A	D_{10}	13G	A_{12}	2F	HLDA	11B	V_{SS}
6B	D_9	13H	A_{11}	7N	PEREQ	2C	V_{SS}
5A	D_8	12H	A_{10}	7M	$\overline{BUSY}$	1D	V_{SS}
5B	D_7	13J	A_9	8N	$\overline{ERROR}$	1M	V_{SS}
4B	D_6	12J	A_8	9M	INTR	4N	V_{SS}
4A	D_5	12K	A_7	8M	NMI	9N	V_{SS}
3B	D_4	13K	A_6	6M	RESET	11N	V_{SS}
2D	D_3	12L	A_5	2B	V_{CC}	2A	V_{SS}
1E	D_2	12M	A_4	12B	V_{CC}	12A	V_{SS}
2E	D_1	11M	A_3	1C	V_{CC}	1B	V_{SS}
1F	D_0	10M	A_2	2M	V_{CC}	13B	V_{SS}
9A	A_{23}	1K	A_1	3N	V_{CC}	13M	V_{SS}
10A	A_{22}	2J	$\overline{BLE}$	5N	V_{CC}	2N	V_{SS}
10B	A_{21}	2K	$\overline{BHE}$	10N	V_{CC}	6N	V_{SS}
12C	A_{20}	4M	W/$\overline{R}$	1A	V_{CC}	12N	V_{SS}
13D	A_{19}	3M	D/$\overline{C}$	3A	V_{CC}	1L	N/C

The following table lists a brief description of each pin on the 80376. The following definitions are used in these descriptions:

- $\overline{\ }$ The named signal is active LOW.
- I Input signal.
- O Output signal.
- I/O Input and Output signal.
- — No electrical connection.

Symbol	Type	Name and Function
CLK2	I	**CLK2** provides the fundamental timing for the 80376. For additional information see **Clock** in Section 4.1.
RESET	I	**RESET** suspends any operation in progress and places the 80376 in a known reset state. See **Interrupt Signals** in Section 4.1 for additional information.
$D_{15}-D_0$	I/O	**DATA BUS** inputs data during memory, I/O and interrupt acknowledge read cycles and outputs data during memory and I/O write cycles. See **Data Bus** in Section 4.1 for additional information.
$A_{23}-A_1$	O	**ADDRESS BUS** outputs physical memory or port I/O addresses. See **Address Bus** in Section 4.1 for additional information.
W/$\overline{R}$	O	**WRITE/READ** is a bus cycle definition pin that distinguishes write cycles from read cycles. See **Bus Cycle Definition Signals** in Section 4.1 for additional information.
D/$\overline{C}$	O	**DATA/CONTROL** is a bus cycle definition pin that distinguishes data cycles, either memory or I/O, from control cycles which are: interrupt acknowledge, halt, and instruction fetching. See **Bus Cycle Definition Signals** in Section 4.1 for additional information.
M/$\overline{IO}$	O	**MEMORY I/O** is a bus cycle definition pin that distinguishes memory cycles from input/output cycles. See **Bus Cycle Definition Signals** in Section 4.1 for additional information.
$\overline{LOCK}$	O	**BUS LOCK** is a bus cycle definition pin that indicates that other system bus masters are denied access to the system bus while it is active. See **Bus Cycle Definition Signals** in Section 4.1 for additional information.
$\overline{ADS}$	O	**ADDRESS STATUS** indicates that a valid bus cycle definition and address (W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{BHE}$, $\overline{BLE}$ and $A_{23}-A_1$) are being driven at the 80376 pins. See **Bus Control Signals** in Section 4.1 for additional information.
$\overline{NA}$	I	**NEXT ADDRESS** is used to request address pipelining. See **Bus Control Signals** in Section 4.1 for additional information.
$\overline{READY}$	I	**BUS READY** terminates the bus cycle. See **Bus Control Signals** in Section 4.1 for additional information.
$\overline{BHE}$, $\overline{BLE}$	O	**BYTE ENABLES** indicate which data bytes of the data bus take part in a bus cycle. See **Address Bus** in Section 4.1 for additional information.
HOLD	I	**BUS HOLD REQUEST** input allows another bus master to request control of the local bus. See **Bus Arbitration Signals** in Section 4.1 for additional information.

Symbol	Type	Name and Function
HLDA	O	**BUS HOLD ACKNOWLEDGE** output indicates that the 80376 has surrendered control of its local bus to another bus master. See **Bus Arbitration Signals** in Section 4.1 for additional information.
INTR	I	**INTERRUPT REQUEST** is a maskable input that signals the 80376 to suspend execution of the current program and execute an interrupt acknowledge function. See **Interrupt Signals** in Section 4.1 for additional information.
NMI	I	**NON-MASKABLE INTERRUPT REQUEST** is a non-maskable input that signals the 80376 to suspend execution of the current program and execute an interrupt acknowledge function. See **Interrupt Signals** in Section 4.1 for additional information.
$\overline{BUSY}$	I	**BUSY** signals a busy condition from a processor extension. See **Coprocessor Interface Signals** in Section 4.1 for additional information.
$\overline{ERROR}$	I	**ERROR** signals an error condition from a processor extension. See **Coprocessor Interface Signals** in Section 4.1 for additional information.
PEREQ	I	**PROCESSOR EXTENSION REQUEST** indicates that the processor extension has data to be transferred by the 80376. See **Coprocessor Interface Signals** in Section 4.1 for additional information.
$\overline{FLT}$	I	**FLOAT,** when active, forces all bidirectional and output signals, including HLDA, to the float condition. FLOAT is not available on the PGA package. See **Float** for additional information.
N/C	—	**NO CONNECT** should always remain unconnected. Connection of a N/C pin may cause the processor to malfunction or be incompatible with future steppings of the 80376.
V_{CC}	I	**SYSTEM POWER** provides the +5V nominal D.C. supply input.
V_{SS}	I	**SYSTEM GROUND** provides 0V connection from which all inputs and outputs are measured.

2.0 ARCHITECTURE OVERVIEW

The 80376 supports the protection mechanisms needed by sophisticated multitasking embedded systems and real-time operating systems. The use of these protection mechanisms is completely optional. For embedded applications not needing protection, the 80376 can easily be configured to provide a 16 Mbyte physical address space.

Instruction pipelining, high bus bandwidth, and a very high performance ALU ensure short average instruction execution times and high system throughput. The 80376 is capable of execution at sustained rates of 2.5–3.0 million instructions per second.

The 80376 offers on-chip testability and debugging features. Four break point registers allow conditional or unconditional break point traps on code execution or data accesses for powerful debugging of even ROM based systems. Other testability features include self-test and tri-stating of output buffers during RESET.

The Intel 80376 embedded processor consists of a central processing unit, a memory management unit and a bus interface. The central processing unit consists of the execution unit and instruction unit. The execution unit contains the eight 32-bit general registers which are used for both address calculation and data operations and a 64-bit barrel shifter used to speed shift, rotate, multiply, and divide operations. The instruction unit decodes the instruction opcodes and stores them in the decoded instruction queue for immediate use by the execution unit.

The Memory Management Unit (MMU) consists of a segmentation and protection unit. Segmentation allows the managing of the logical address space by providing an extra addressing component, one that allows easy code and data relocatability, and efficient sharing.

The protection unit provides four levels of protection for isolating and protecting applications and the operating system from each other. The hardware enforced protection allows the design of systems with a high degree of integrity and simplifies debugging.

Finally, to facilitate high performance system hardware designs, the 80376 bus interface offers address pipelining and direct Byte Enable signals for each byte of the data bus.

2.1 Register Set

The 80376 has twenty-nine registers as shown in Figure 2.1. These registers are grouped into the following six categories:

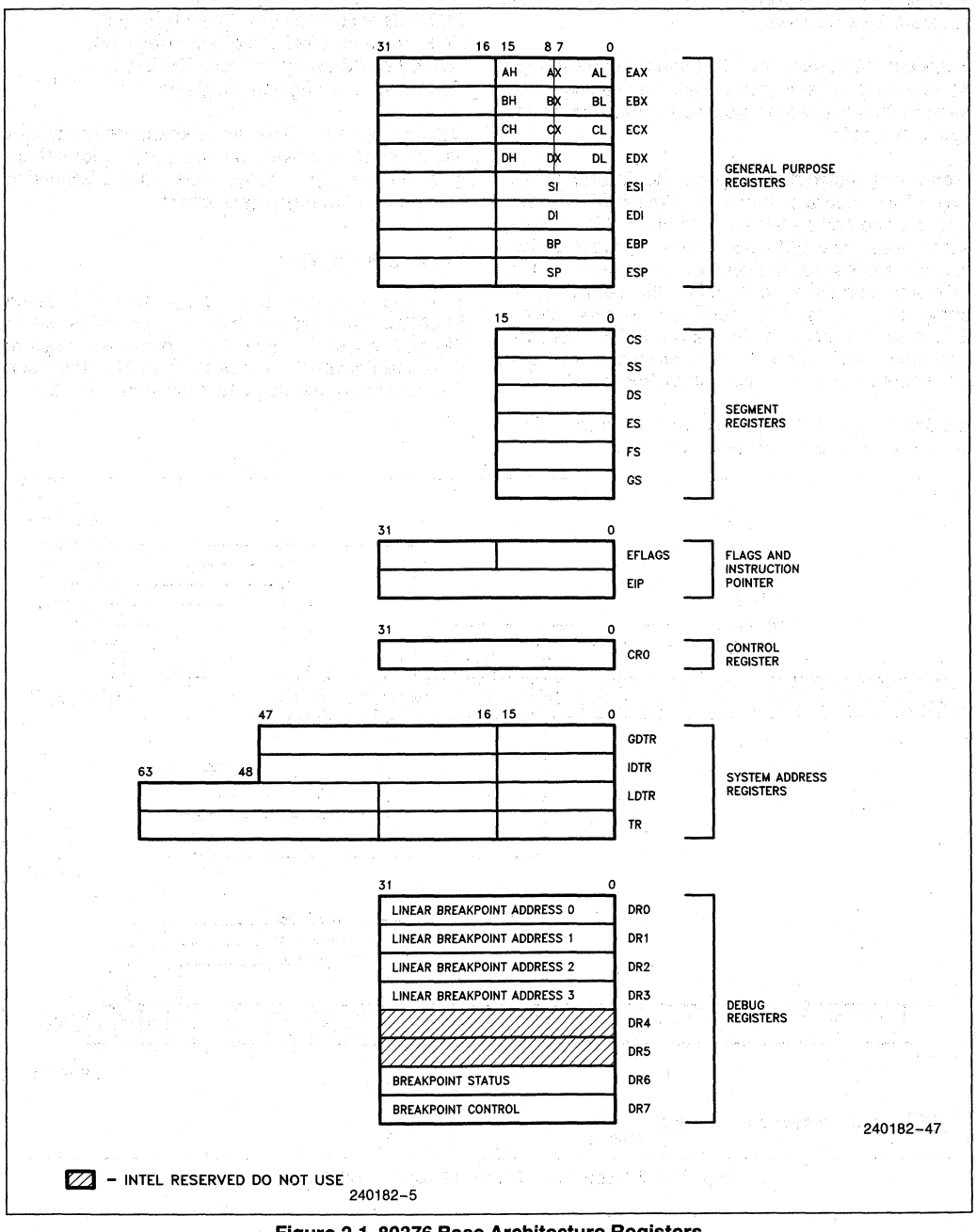


Figure 2.1. 80376 Base Architecture Registers

General Registers: The eight 32-bit general purpose registers are used to contain arithmetic and logical operands. Four of these (EAX, EBX, ECX and EDX) can be used either in their entirety as 32-bit registers, as 16-bit registers, or split into pairs of separate 8-bit registers.

Segment Registers: Six 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data.

Flags and Instruction Pointer Registers: These two 32-bit special purpose registers in Figure 2.1 record or control certain aspects of the 80376 processor state. The EFLAGS register includes status and control bits that are used to reflect the outcome of many instructions and modify the semantics of some instructions. The Instruction Pointer, called EIP, is 32 bits wide. The Instruction Pointer controls instruction fetching and the processor automatically increments it after executing an instruction.

Control Register: The 32-bit control register, CR0, is used to control Coprocessor Emulation.

System Address Registers: These four special registers reference the tables or segments supported by the 80376/80386 protection model. These tables or segments are:

GDTR (Global Descriptor Table Register),
IDTR (Interrupt Descriptor Table Register),
LDTR (Local Descriptor Table Register),
TR (Task State Segment Register).

Debug Registers: The six programmer accessible debug registers provide on-chip support for debugging. The use of the debug registers is described in Section 2.11 **Debugging Support**.

EFLAGS REGISTER

The flag Register is a 32-bit register named EFLAGS. The defined bits and bit fields within EFLAGS, shown in Figure 2.2, control certain operations and indicate the status of the 80376 processor. The function of the flag bits is given in Table 2.1.

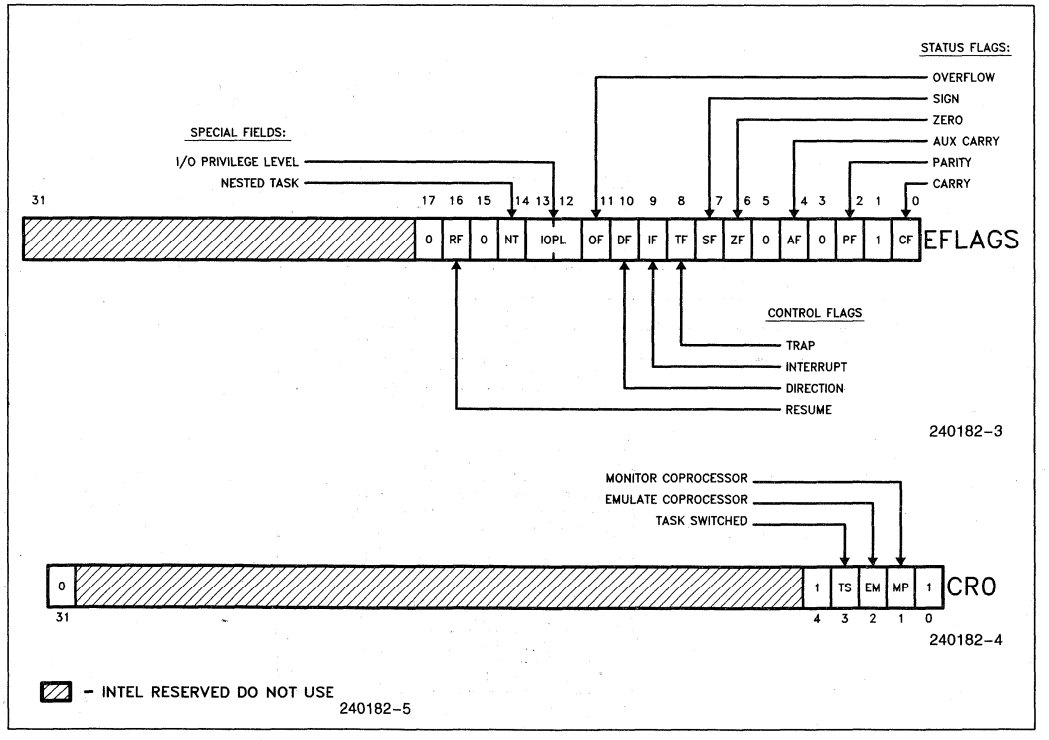


Figure 2.2. Status and Control Register Bit Functions

Table 2.1. Flag Definitions

Bit Position	Name	Function
0	CF	**Carry Flag**—Set on high-order bit carry or borrow; cleared otherwise.
2	PF	**Parity Flag**—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise.
4	AF	**Auxiliary Carry Flag**—Set on carry from or borrow to the low order four bits of AL; cleared otherwise.
6	ZF	**Zero Flag**—Set if result is zero; cleared otherwise.
7	SF	**Sign Flag**—Set equal to high-order bit of result (0 if positive, 1 if negative).
8	TF	**Single Step Flag**—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	**Interrupt-Enable Flag**—When set, external interrupts signaled on the INTR pin will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	**Direction Flag**—Causes string instructions to auto-increment (default) the appropriate index registers when cleared. Setting DF causes auto-decrement.
11	OF	**Overflow Flag**—Set if the operation resulted in a carry/borrow into the sign bit (high-order bit) of the result but did not result in a carry/borrow out of the high-order bit or vice-versa.
12, 13	IOPL	**I/O Privilege Level**—Indicates the maximum CPL permitted to execute I/O instructions without generating an exception 13 fault or consulting the I/O permission bit map. It also indicates the maximum CPL value allowing alteration of the IF bit.
14	NT	**Nested Task**—Indicates that the execution of the current task is nested within another task (see **Task Switching**).
16	RF	**Resume Flag**—Used in conjunction with debug register breakpoints. It is checked at instruction boundaries before breakpoint processing. If set, any debug fault is ignored on the next instruction. It is reset at the successful completion of any instruction except IRET, POPF, and those instructions causing task switches.

5

CONTROL REGISTER

The 80376 has a 32-bit control register called CR0 that is used to control coprocessor emulation. This register is shown in Figures, 2.1 and 2.2. The defined CR0 bits are described in Table 2.2. Bits 0, 4 and 31 of CR0 have fixed values in the 80376. These values cannot be changed. Programs that load CR0 should always load bits 0, 4 and 31 with values previously there to be compatible with the 80386.

Table 2.2. CR0 Definitions

Bit Position	Name	Function
1	MP	**Monitor Coprocessor Extension**—Allows WAIT instructions to cause a processor extension not present exception (number 7).
2	EM	**Emulate Processor Extension**—When set, this bit causes a processor extension not present exception (number 7) on ESC instructions to allow processor extension emulation.
3	TS	**Task Switched**—When set, this bit indicates the next instruction using a processor extension will cause exception 7, allowing software to test whether the current processor extension context belongs to the current task (see **Task Switching**).

2.2 Instruction Set

The instruction set is divided into nine categories of operations:

Data Transfer
Arithmetic
Shift/Rotate
String Manipulation
Bit Manipulation
Control Transfer
High Level Language Support
Operating System Support
Processor Control

These 80376 processor instructions are listed in Table 8.1 **80376 Instruction Set and Clock Count Summary**.

All 80376 processor instructions operate on either 0, 1, 2 or 3 operands; an operand resides in a register, in the instruction itself, or in memory. Most zero operand instructions (e.g. CLI, STI) take only one byte. One operand instructions generally are two bytes long. The average instruction is 3.2 bytes long. Since the 80376 has a 16-byte prefetch instruction queue an average of 5 instructions can be prefetched. The use of two operands permits the following types of common instructions:

Register to Register
Memory to Register
Immediate to Register
Memory to Memory
Register to Memory
Immediate to Memory

The operands are either 8-, 16- or 32-bit long.

2.3 Memory Organization

Memory on the 80376 is divided into 8-bit quantities (bytes), 16-bit quantities (words), and 32-bit quantities (dwords). Words are stored in two consecutive bytes in memory with the low-order byte at the lowest address. Dwords are stored in four consecutive bytes in memory with the low-order byte at the lowest address. The address of a word or Dword is the byte address of the low-order byte. For maximum performance word and dword values should be at even physical addresses.

In addition to these basic data types the 80376 processor supports segments. Memory can be divided up into one or more variable length segments, which can be shared between programs.

ADDRESS SPACES

The 80376 has three types of address spaces: **logical, linear,** and **physical.** A **logical** address (also known as a **virtual** address) consists of a selector and an offset. A selector is the contents of a segment register. An offset is formed by summing all of the addressing components (BASE, INDEX, and DISPLACEMENT), discussed in Section 2.4 **Addressing Modes,** into an effective address.

Every selector has a **logical base** address associated with it that can be up to 32 bits in length. This 32-bit **logical base** address is added to either a 32-bit offset address or a 16-bit offset address (by using the ***address length prefix***)to form a final 32-bit **linear** address. This final **linear** address is then truncated so that only the lower 24 bits of this address are used to address the 16 Mbytes physical memory address space. The **logical base** address is stored in one of two operating system tables (i.e. the Local Descriptor Table or Global Descriptor Table).

Figure 2.3 shows the relationship between the various address spaces.

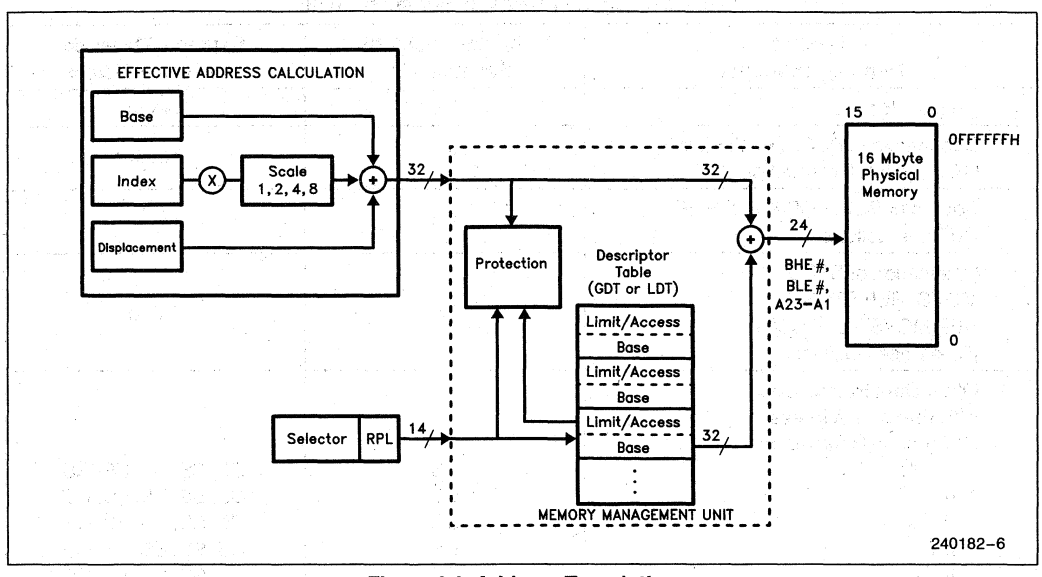


Figure 2.3. Address Translation

SEGMENT REGISTER USAGE

The main data structure used to organize memory is the segment. On the 80376, segments are variable sized blocks of linear addresses which have certain attributes associated with them. There are two main types of segments, code and data. The simplest use of segments is to have one code and data segment. Each segment is 16 Mbytes in size overlapping each other. This allows code and data to be directly addressed by the same offset.

In order to provide compact instruction encoding and increase processor performance, instructions do not need to explicitly specify which segment register is used. The segment register is automatically chosen according to the rules of Table 2.3 (Segment Register Selection Rules). In general, data references use the selector contained in the DS register, stack references use the SS register and instruction fetches use the CS register. The contents of the Instruction Pointer provide the offset. Special segment override prefixes allow the explicit use of a given segment register, and override the implicit rules listed in Table 2.3. The override prefixes also allow the use of the ES, FS and GS segment registers.

There are no restrictions regarding the overlapping of the base addresses of any segments. Thus, all 6 segments could have the base address set to zero. Further details of segmentation are discussed in Section 3.0 Architecture.

Table 2.3. Segment Register Selection Rules

Type of Memory Reference	Implied (Default) Segment Use	Segment Override Prefixes Possible
Code Fetch	CS	None
Destination of PUSH, PUSHF, INT, CALL, PUSHA Instructions	SS	None
Source of POP, POPA, POPF, IRET, RET Instructions	SS	None
Destination of STOS, MOVS, REP STOS, REP MOVS Instructions (DI is Base Register)	ES	None
Other Data References, with Effective Address Using Base Register of:		
[EAX]	DS	CS, SS, ES, FS, GS
[EBX]	DS	CS, SS, ES, FS, GS
[ECX]	DS	CS, SS, ES, FS, GS
[EDX]	DS	CS, SS, ES, FS, GS
[ESI]	DS	CS, SS, ES, FS, GS
[EDI]	DS	CS, SS, ES, FS, GS
[EBP]	SS	CS, SS, ES, FS, GS
[ESP]	SS	CS, SS, ES, FS, GS

2.4 Addressing Modes

The 80376 provides a total of 8 addressing modes for instructions to specify operands. The addressing modes are optimized to allow the efficient execution of high level languages such as C and FORTRAN, and they cover the vast majority of data references needed by high-level languages.

Two of the addressing modes provide for instructions that operate on register or immediate operands:

Register Operand Mode: The operand is located in one of the 8-, 16- or 32-bit general registers.

Immediate Operand Mode: The operand is included in the instruction as part of the opcode.

The remaining 6 modes provide a mechanism for specifying the effective address of an operand. The linear address consists of two components: the segment base address and an effective address. The effective address is calculated by summing any combination of the following three address elements (see Figure 2.3):

DISPLACEMENT: an 8-, 16- or 32-bit immediate value following the instruction.

BASE: The contents of any general purpose register. The base registers are generally used by compilers to point to the start of the local variable area. Note that if the ***Address Length Prefix*** is used, only BX and BP can be used as a BASE register.

INDEX: The contents of any general purpose register except for ESP. The index registers are used to access the elements of an array, or a string of characters. The index register's value can be multiplied by a scale factor, either 1, 2, 4 or 8. The scaled index is especially useful for accessing arrays or structures. Note that if the ***Address Length Prefix*** is used, no Scaling is available and only the registers SI and DI can be used to INDEX.

Combinations of these 3 components make up the 6 additional addressing modes. There is no performance penalty for using any of these addressing combinations, since the effective address calculation is pipelined with the execution of other instructions. The one exception is the simultaneous use of BASE and INDEX components which requires one additional clock.

As shown in Figure 2.4, the effective address (EA) of an operand is calculated according to the following formula:

$$EA = BASE_{Register} + (INDEX_{Register} \times scaling) + DISPLACEMENT$$

1. **Direct Mode:** The operand's offset is contained as part of the instruction as an 8-, 16- or 32-bit DISPLACEMENT.
2. **Register Indirect Mode:** A BASE register contains the address of the operand.
3. **Based Mode:** A BASE register's contents is added to a DISPLACEMENT to form the operand's offset.
4. **Scaled Index Mode:** An INDEX register's contents is multiplied by a SCALING factor which is added to a DISPLACEMENT to form the operand's offset.
5. **Based Scaled Index Mode:** The contents of an INDEX register is multiplied by a SCALING factor and the result is added to the contents of a BASE register to obtain the operand's offset.
6. **Based Scaled Index Mode with Displacement:** The contents of an INDEX register are multiplied by a SCALING factor, and the result is added to the contents of a BASE register and a DISPLACEMENT to form the operand's offset.

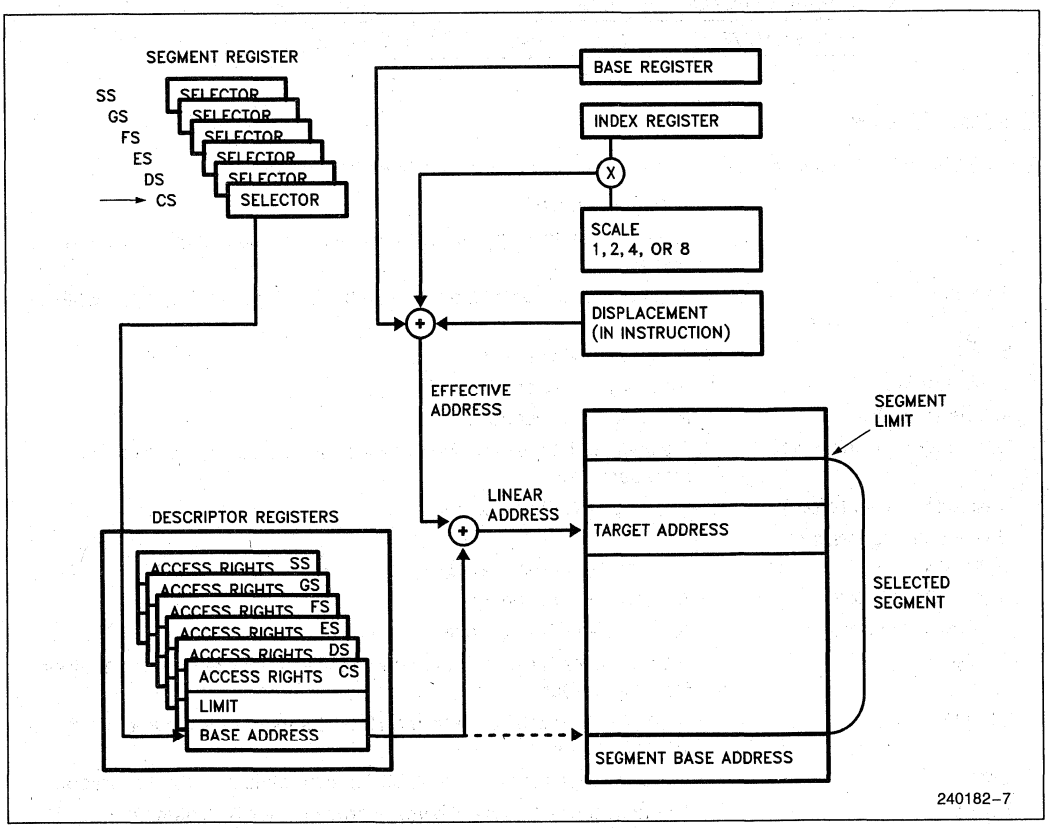


Figure 2.4. Addressing Mode Calculations

GENERATING 16-BIT ADDRESSES

The 80376 executes code with a default length for operands and addresses of 32 bits. The 80376 is also able to execute operands and addresses of 16 bits. This is specified through the use of override prefixes. Two prefixes, the **Operand Length Prefix** and the **Address Length Prefix,** override the default 32-bit length on an individual instruction basis. These prefixes are automatically added by assemblers. The Operand Length and Address Length Prefixes can be applied separately or in combination to any instruction.

The 80376 normally executes 32-bit code and uses either 8- or 32-bit displacements, and any register can be used as based or index registers. When executing 16-bit code (by prefix overrides), the displacements are either 8 or 16 bits, and the base and index register conform to the 16-bit model. Table 2.4 illustrates the differences.

Table 2.4. BASE and INDEX Registers for 16- and 32-Bit Addresses

	16-Bit Addressing	32-Bit Addressing
BASE REGISTER	BX, BP	Any 32-Bit GP Register
INDEX REGISTER	SI, DI	Any 32-Bit GP Register except ESP
SCALE FACTOR	None	1, 2, 4, 8
DISPLACMENT	0, 8, 16 Bits	0, 8, 32 Bits

2.5 Data Types

The 80376 supports all of the data types commonly used in high level languages:

Bit:	A single bit quantity.
Bit Field:	A group of up to 32 contiguous bits, which spans a maximum of four bytes.
Bit String:	A set of contiguous bits, on the 80376 bit strings can be up to 16 Mbits long.
Byte:	A signed 8-bit quantity.
Unsigned Byte:	An unsigned 8-bit quantity.
Integer (Word):	A signed 16-bit quantity.
Long Integer (Double Word):	A signed 32-bit quantity. All operations assume a 2's complement representation.
Unsigned Integer (Word):	An unsigned 16-bit quantity.
Unsigned Long Integer (Double Word):	An unsigned 32-bit quantity.
Signed Quad Word:	A signed 64-bit quantity.
Unsigned Quad Word:	An unsigned 64-bit quantity.
Pointer:	A 16- or 32-bit offset only quantity which indirectly references another memory location.
Long Pointer:	A full pointer which consists of a 16-bit segment selector and either a 16- or 32-bit offset.
Char:	A byte representation of an ASCII Alphanumeric or control character.
String:	A contiguous sequence of bytes, words or dwords. A string may contain between 1 byte and 16 Mbytes.
BCD:	A byte (unpacked) representation of decimal digits 0–9.
Packed BCD:	A byte (packed) representation of two decimal digits 0–9 storing one digit in each nibble.

When the 80376 is coupled with a numerics Coprocessor such as the 80387SX then the following common Floating Point types are supported.

Floating Point: A signed 32-, 64- or 80-bit real number representation. Floating point numbers are supported by the 80387SX numerics coprocessor.

Figure 2.5 illustrates the data types supported by the 80376 processor and the 80387SX coprocessor.

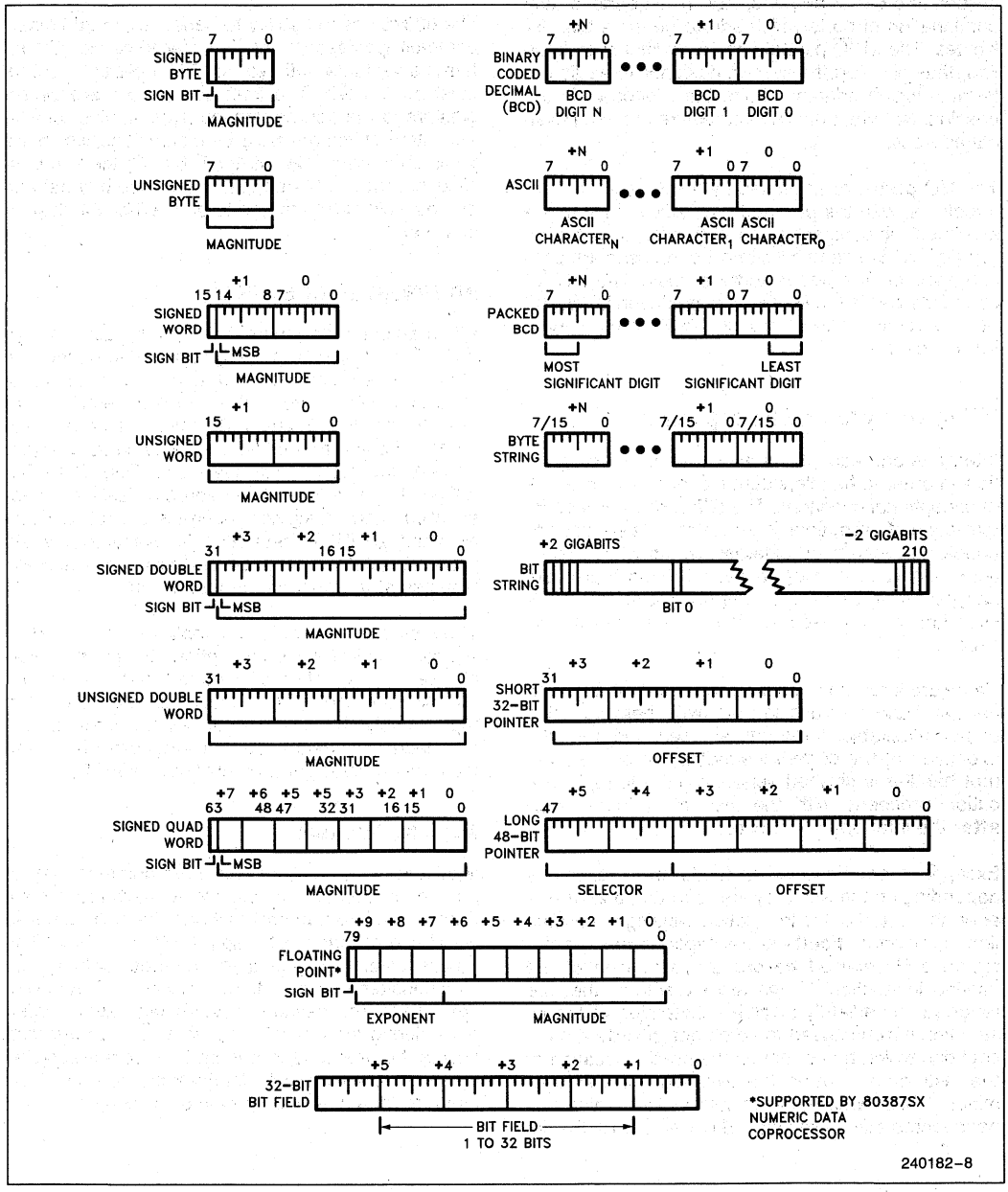


Figure 2.5. 80376 Supported Data Types

2.6 I/O Space

The 80376 has two distinct physical address spaces: physical memory and I/O. Generally, peripherals are placed in I/O space although the 80376 also supports memory-mapped peripherals. The I/O space consists of 64 Kbytes which can be divided into 64K 8-bit ports, 32K 16-bit ports, or any combination of ports which add to no more than 64 Kbytes. The M/$\overline{IO}$ pin acts as an additional address line, thus allowing the system designer to easily determine which address space the processor is accessing. Note that the I/O address refers to a physical address.

The I/O ports are accessed by the IN and OUT instructions, with the port address supplied as an immediate 8-bit constant in the instruction or in the DX register. All 8-bit and 16-bit port addresses are zero extended on the upper address lines. The I/O instructions cause the M/$\overline{IO}$ pin to be driven LOW. I/O port addresses 00F8H through 00FFH are reserved for use by Intel.

2.7 Interrupts and Exceptions

Interrupts and exceptions alter the normal program flow in order to handle external events, report errors or exceptional conditons. The difference between interrupts and exceptions is that interrupts are used to handle asynchronous external events while exceptions handle instruction faults. Although a program can generate a software interrupt via an INT N instruction, the processor treats software interrupts as exceptions.

Hardware interrupts occur as the result of an external event and are classified into two types: maskable or non-maskable. Interrupts are serviced after the execution of the current instruction. After the interrupt handler is finished servicing the interrupt, execution proceeds with the instruction immediately **after** the interrupted instruction.

Exceptions are classified as faults, traps, or aborts depending on the way they are reported, and whether or not restart of the instruction causing the exception is suported. **Faults** are exceptions that are detected and serviced **before** the execution of the faulting instruction. **Traps** are exceptions that are reported immediately **after** the execution of the instruction which caused the problem. **Aborts** are exceptions which do not permit the precise location of the instruction causing the exception to be determined. Thus, when an interrupt service routine has been completed, execution proceeds from the instruction immediately following the interrupted instruction. On the other hand the return address from an exception/fault routine will always point at the instruction causing the exception and include any leading instruction prefixes. Table 2.5 summarizes the possible interrupts for the 80376 and shows where the return address points to.

The 80376 has the ability to handle up to 256 different interrupts/exceptions. In order to service the interrupts, a table with up to 256 interrupt vectors must be defined. The interrupt vectors are simply pointers to the appropriate interrupt service routine. The interrupt vectors are 8-byte quantities, which are put in an Interrupt Descriptor Table. Of the 256 possible interrupts, 32 are reserved for use by Intel and the remaining 224 are free to be used by the system designer.

INTERRUPT PROCESSING

When an interrupt occurs the following actions happen. First, the current program address and the Flags are saved on the stack to allow resumption of the interrupted program. Next, an 8-bit vector is supplied to the 80376 which identifies the appropriate entry in the interrupt table. The table contains either an Interrupt Gate, a Trap Gate or a Task Gate that will point to an interrupt procedure or task. The user supplied interrupt service routine is executed. Finally, when an IRET instruction is executed the old processor state is restored and program execution resumes at the appropriate instruction.

The 8-bit interrupt vector is supplied to the 80376 in several different ways: exceptions supply the interrupt vector internally; software INT instructions contain or imply the vector; maskable hardware interrupts supply the 8-bit vector via the interrupt acknowledge bus sequence. Non-Maskable hardware interrupts are assigned to interrupt vector 2.

Maskable Interrupt

Maskable interrupts are the most common way to respond to asynchronous external hardware events. A hardware interrupt occurs when the INTR is pulled HIGH and the Interrupt Flag bit (IF) is enabled. The processor only responds to interrupts between instructions (string instructions have an "interrupt window" between memory moves which allows interrupts during long string moves). When an interrupt occurs the processor reads an 8-bit vector supplied by the hardware which identifies the source of the interrupt (one of 224 user defined interrupts).

Table 2.5. Interrupt Vector Assignments

Function	Interrupt Number	Instruction Which Can Cause Exception	Return Address Points to Faulting Instruction	Type
Divide Error	0	DIV, IDIV	Yes	FAULT
Debug Exception	1	Any Instruction	Yes	TRAP*
NMI Interrupt	2	INT 2 or NMI	No	NMI
One-Byte Interrupt	3	INT	No	TRAP
Interrupt on Overflow	4	INTO	No	TRAP
Array Bounds Check	5	BOUND	Yes	FAULT
Invalid OP-Code	6	Any Illegal Instruction	Yes	FAULT
Device Not Available	7	ESC, WAIT	Yes	FAULT
Double Fault	8	Any Instruction That Can Generate an Exception		ABORT
Coprocessor Segment Overrun	9	ESC	No	ABORT
Invalid TSS	10	JMP, CALL, IRET, INT	Yes	FAULT
Segment Not Present	11	Segment Register Instructions	Yes	FAULT
Stack Fault	12	Stack References	Yes	FAULT
General Protection Fault	13	Any Memory Reference	Yes	FAULT
Intel Reserved	14–15	—	—	—
Coprocessor Error	16	ESC, WAIT	Yes	FAULT
Intel Reserved	17–32			
Two-Byte Interrupt	0–255	INT n	No	TRAP

*Some debug exceptions may report both traps on the previous instruction, and faults on the next instruction.

Interrupts through Interrupt Gates automatically reset IF, disabling INTR requests. Interrupts through Trap Gates leave the state of the IF bit unchanged. Interrupts through a Task Gate change the IF bit according to the image of the EFLAGs register in the task's Task State Segment (TSS). When an IRET instruction is executed, the original state of the IF bit is restored.

Non-Maskable Interrupt

Non-maskable interrupts provide a method of servicing very high priority interrupts. When the NMI input is pulled HIGH it causes an interrupt with an internally supplied vector value of 2. Unlike a normal hardware interrupt no interrupt acknowledgement sequence is performed for an NMI.

While executing the NMI servicing procedure, the 80376 will not service any further NMI request, or INT requests, until an interrupt return (IRET) instruction is executed or the processor is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. The disabling of INTR requests depends on the gate in IDT location 2.

Software Interrupts

A third type of interrupt/exception for the 80376 is the software interrupt. An INT n instruction causes the processor to execute the interrupt service routine pointed to by the n^{th} vector in the interrupt table.

A special case of the two byte software interrupt INT n is the one byte INT 3, or breakpoint interrupt. By inserting this one byte instruction in a program, the user can set breakpoints in his program as a debugging tool.

A final type of software interrupt, is the single step interrupt. It is discussed in **Single-Step Trap** (page 22).

INTERRUPT AND EXCEPTION PRIORITIES

Interrupts are externally-generated events. Maskable Interrupts (on the INTR input) and Non-Maskable Interrupts (on the NMI input) are recognized at instruction boundaries. When NMI and maskable INTR are **both** recognized at the **same** instruction boundary, the 80376 invokes the NMI service routine first. If, after the NMI service routine has been invoked, maskable interrupts are still enabled, then the 80376 will invoke the appropriate interrupt service routine.

As the 80376 executes instructions, it follows a consistent cycle in checking for exceptions, as shown in Table 2.6. This cycle is repeated as each instruction is executed, and occurs in parallel with instruction decoding and execution.

INSTRUCTION RESTART

The 80376 fully supports restarting all instructions after faults. If an exception is detected in the instruction to be executed (exception categories 4 through 9 in Table 2.6), the 80376 device invokes the appropriate exception service routine. The 80376 is in a state that permits restart of the instruction.

DOUBLE FAULT

A Double fault (exception 8) results when the processor attempts to invoke an exception service routine for the segment exceptions (10, 11, 12 or 13), but in the process of doing so, detects an exception.

2.8 Reset and Initialization

When the processor is Reset the registers have the values shown in Table 2.7. The 80376 will then start executing instructions near the top of physical memory, at location 0FFFFF0H. A short JMP should be executed within the segment defined for power-up (see Table 2.7). The GDT should then be initialized for a start-up data and code segment followed by a far JMP that will load the segment descriptor cache with the new descriptor values. The IDT table, after reset, is located at physical address 0H, with a limit of 256 entries.

RESET forces the 80376 to terminate all execution and local bus activity. No instruction execution or bus activity will occur as long as Reset is active. Between 350 and 450 CLK2 periods after Reset becomes inactive, the 80376 will start executing instructions at the top of physical memory.

Table 2.6. Sequence of Exception Checking

Consider the case of the 80376 having just completed an instruction. It then performs the following checks before reaching the point where the next instruction is completed:

1. Check for Exception 1 Traps from the instruction just completed (single-step via Trap Flag, or Data Breakpoints set in the Debug Registers).
2. Check for external NMI and INTR.
3. Check for Exception 1 Faults in the next instruction (Instruction Execution Breakpoint set in the Debug Registers for the next instruction).
4. Check for Segmentation Faults that prevented fetching the entire next instruction (exceptions 11 or 13).
5. Check for Faults decoding the next instruction (exception 6 if illegal opcode; or exception 13 if instruction is longer than 15 bytes, or privilege violation (i.e. not at IOPL or at CPL = 0).
6. If WAIT opcode, check if TS = 1 and MP = 1 (exception 7 if both are 1).
7. If ESCape opcode for numeric coprocessor, check if EM = 1 or TS = 1 (exception 7 if either are 1).
8. If WAIT opcode or ESCape opcode for numeric coprocessor, check $\overline{\text{ERROR}}$ input signal (exception 16 if $\overline{\text{ERROR}}$ input is asserted).
9. Check for Segmentation Faults that prevent transferring the entire memory quantity (exceptions 11, 12, 13).

Table 2.7. Register Values after Reset

Flag Word (EFLAGS)	uuuu0002H	(Note 1)
Machine Status Word (CR0)	uuuuuuu1H	(Note 2)
Instruction Pointer (EIP)	0000FFF0H	
Code Segment (CS)	F000H	(Note 3)
Data Segment (DS)	0000H	(Note 4)
Stack Segment (SS)	0000H	
Extra Segment (ES)	0000H	(Note 4)
Extra Segment (FS)	0000H	
Extra Segment (GS)	0000H	
EAX Register	0000H	(Note 5)
EDX Register	Component and Stepping ID	(Note 6)
All Other Registers	Undefined	(Note 7)

NOTES:
1. EFLAG Register. The upper 14 bits of the EFLAGS register are undefined, all defined flag bits are zero.
2. CR0: The defined 4 bits in the CR0 is equal to 1H.
3. The Code Segment Register (CS) will have its Base Address set to 0FFFF0000H and Limit set to 0FFFFH.
4. The Data and Extra Segment Registers (DS and ES) will have their Base Address set to 000000000H and Limit set to 0FFFFH.
5. If self-test is selected, the EAX should contain a 0 value. If a value of 0 is not found the self-test has detected a flaw in the part.
6. EDX register always holds component and stepping identifier.
7. All unidentified bits are Intel Reserved and should not be used.

5

2.9 Initialization

Because the 80376 processor starts executing in protected mode, certain precautions need be taken during initialization. Before any far jumps can take place the GDT and/or LDT tables need to be setup and their respective registers loaded. Before interrupts can be initialized the IDT table must be setup and the IDTR must be loaded. The example code is shown below:

```
; ************************************************************
;
; This is an example of startup code to put either an 80376,
; 80386SX or 80386 into flat mode. All of memory is treated as
; simple linear RAM. There are no interrupt routines. The
; Builder creates the GDT-alias and IDT-alias and places them,
; by default, in GDT[1] and GDT[2].  Other entries in the GDT
; are specified in the Build file.  After initialization it jumps
; to a C startup routine. To use this template, change this jmp
; address to that of your code, or make the label of your code
; "c_startup".
;
; This code was assembled and built using version 1.2 of the
; Intel RLL utilities and Intel 386ASM assembler.
;
;             ***    This code was tested    ***
;
; ************************************************************
```

```
NAME FLAT                ; name of the object module

EXTRN    c_startup:near ; this is the label jmped to after init

pe_flag       equ 1
data_selc     equ 20h    ; assume code is GDT[3], data GDT[4]

INIT_CODE    SEGMENT ER PUBLIC USE32     ; Segment base at 0ffffff80h

PUBLIC GDT_DESC

gdt_desc    dq  ?

PUBLIC    START

start:
    cld                          ; clear direction flag
    smsw bx                      ; check for processor (80376) at reset
    test bl,1                    ; use SMSW rather than MOV for speed
    jnz pestart
realstart                        ; is an 80386 and in real mode
    db 66h                       ; force the next operand into 32-bit mode.
    mov eax,offset gdt_desc      ; move address of the GDT descriptor into eax
    xor ebx,ebx                  ; clear ebx
    mov bh,ah                    ; load 8 bits of address into bh
    move bl,al                   ; load 8 bits of address into bl
    db 67h
    db 66h                       ; use the 32-bit form of LGDT to load
    lgdt cs:[ebx]                ; the 32-bits of address into the GDTR
    smsw ax                      ; go into protected mode (set PE bit)
    or al,pe_flag
    lmsw ax
    jmp next                     ; flush prefetch queue
pestart:
    mov ebx,offset gdt_desc
    xor eax,eax
    mov ax,bx                    ; lower portion of address only
    lgdt cs:[eax]
    xor ebx,ebx                  ; initialize data selectors
    mov bl,data_selc             ; GDT[3]
    mov ds,bx
    mov ss,bx
    mov es,bx
    mov fs,bx
    mov gs,bx
    jmp pejump
next:
    xor ebx,ebx                  ; initialize data selectors
    mov bl,data_selc             ; GDT[3]
    mov ds,bx
    mov ss,bx
    mov es,bx
    mov fs,bx
    mov gs,bx
    db 66h                       ; for the 80386, need to make a 32-bit jump
pejump:
    jmp far ptr c_startup        ; but the 80376 is already 32-bit.

    org 70h                      ; only if segment base is at 0ffffff80h
    jmp short start
INIT_CODE ENDS
END
```

This code should be linked into your application for boot loadable code. The following build file illustrates how this is accomplished.

```
FLAT; -- build program id

SEGMENT
    *segments (dpl=0),              -- Give all user segments a DPL of 0.
    _phantom_code_ (dpl=0),         -- These two segments are created by
    _phantom_data_ (dpl=0),         -- the builder when the FLAT control is used.
    init_code  (base=0ffffff80h); -- Put startup code at the reset vector area.

GATE
    gl3 (entry=13, dpl=0, trap),        -- trap gate disables interrupts
    i32 (entry=32, dpl=0, interrupt),  -- interrupt gates doesn't

TABLE
    -- create GDT

    GDT (LOCATION = GDT_DESC,       -- In a buffer starting at GDT_DESC,
                                    -- BLD386 places the GDT base and
                                    -- GDT limit values. Buffer must be
                                    -- 6 bytes long. The base and limit
                                    -- values are places in this buffer
                                    -- as two bytes of limit plus
                                    -- four bytes of base in the format
                                    -- required for use by the LGDT
                                    -- instruction.
        ENTRY = (3:_phantom_code_,     -- Explicitly place segment
                 4:_phantom_data_,     -- entries into the GDT.
                 5:code32,
                 6:data,
                 7:init_code)
        );
TASK

    MAIN_TASK
        (
         DPL  = 0,                  -- Task privilege level is 0.
         DATA = DATA,               -- Points to a segment that
                                    -- indicates initial DS value.
         CODE = main,               -- Entry point is main, which
                                    -- must be a public id.

         STACKS = (DATA),           -- Segment id points to stack
                                    -- segment. Sets the initial SS:ESP.
         NO INTENABLED,             -- Disable interrupts.
         PRESENT                    -- Present bit in TSS set to 1.
        );

    MEMORY
        (RANGE = (EPROM = ROM(0ffff8000h..0ffffffffh),
                  DRAM = RAM(0..0ffffh)),
        ALLOCATE = (EPROM = (MAIN_TASK)));

END
```

```
asm386 flatsim.a38 debug
asm386 application.a38 debug
bnd386 application.obj,flatsim.obj nolo debug oj (application.bnd)
bld386 application.bnd bf (flatsim.bld) bl flat
```

Commands to assemble and build a boot-loadable application named "application.a38". The initialization code is called "flatsim.a38", and build file is called "application.bld".

2.10 Self-Test

The 80376 has the capability to perform a self-test. The self-test checks the function of all of the Control ROM and most of the non-random logic of the part. Approximately one-half of the 80376 can be tested during self-test.

Self-Test is initiated on the 80376 when the RESET pin transitions from HIGH to LOW, and the $\overline{BUSY}$ pin is LOW. The self-test takes about 2^{20} clocks, or approximately 33 ms with a 16 MHz 80376 processor. At the completion of self-test the processor performs reset and begins normal operation. The part has successfully passed self-test if the contents of the EAX register is zero. If the EAX register is not zero then the self-test has detected a flaw in the part. If self-test is not selected after reset, EAX may be non-zero after reset.

2.11 Debugging Support

The 80376 provides several features which simplify the debugging process. The three categories of on-chip debugging aids are:

1. The code execution breakpoint opcode (0CCH).
2. The single-step capability provided by the TF bit in the flag register, and
3. The code and data breakpoint capability provided by the Debug Registers DR0–3, DR6, and DR7.

BREAKPOINT INSTRUCTION

A single-byte software interrupt (Int 3) breakpoint instruction is available for use by software debuggers. The breakpoint opcode is 0CCh, and generates an exception 3 trap when executed.

DEBUG REGISTERS

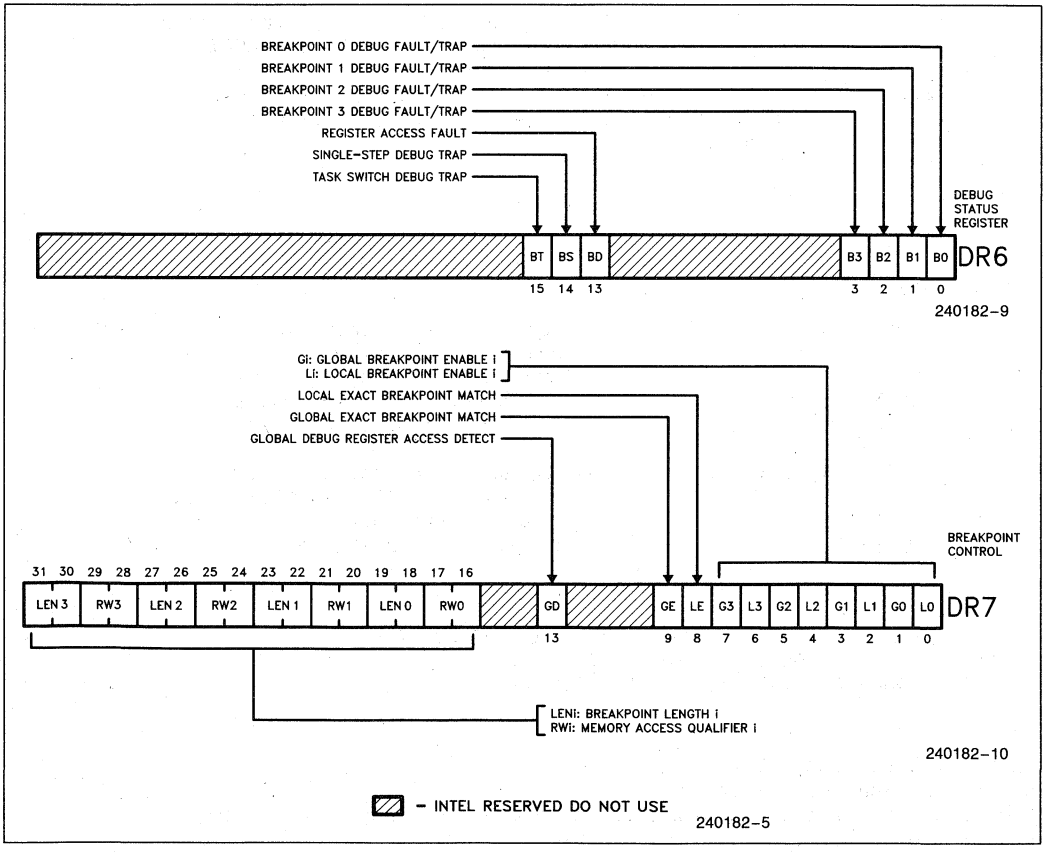


Figure 2.6. Debug Registers

SINGLE-STEP TRAP

If the single-step flag (TF, bit 8) in the EFLAG register is found to be set at the end of an instruction, a single-step exception occurs. The single-step exception is auto vectored to exception number 1.

The Debug Registers are an advanced debugging feature of the 80376. They allow data access breakpoints as well as code execution breakpoints. Since the breakpoints are indicated by on-chip registers, an instruction execution breakpoint can be placed in ROM code or in code shared by several tasks, neither of which can be supported by the INT 3 breakpoint opcode.

The 80376 contains six Debug Registers, consisting of four breakpoint address registers and two breakpoint control registers. Initially after reset, breakpoints are in the disabled state; therefore, no breakpoints will occur unless the debug registers are programmed. Breakpoints set up in the Debug Registers are auto-vectored to exception 1. Figure 2.6 shows the breakpoint status and control registers.

3.0 ARCHITECTURE

The Intel 80376 Embedded Processor has a physical address space of 16 Mbytes (2^{24} bytes) and allows the running of virtual memory programs of almost unlimited size (16 Kbytes × 16 Mbytes or 256 Gbytes (2^{38} bytes)). In addition the 80376 provides a sophisticated memory management and a hardware-assisted protection mechanism.

3.1 Addressing Mechanism

The 80376 uses two components to form the logical address, a 16-bit selector which determines the linear base address of a segment, and a 32-bit effective address. The selector is used to specify an index into an operating system defined table (see Figure 3.1). The table contains the 32-bit base address of a given segment. The linear address is formed by adding the base address obtained from the table to the 32-bit effective address. This value is truncated to 24 bits to form the physical address, which is then placed on the address bus.

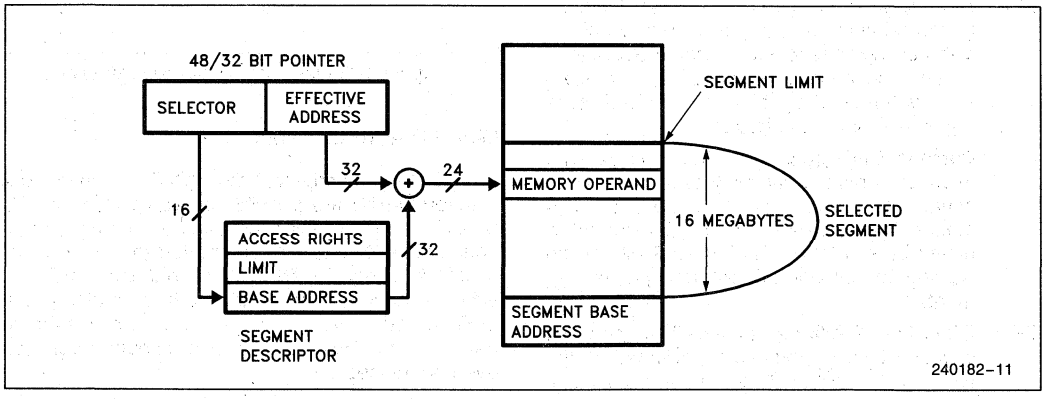


Figure 3.1. Address Calculation

3.2 Segmentation

Segmentation is one method of memory management and provides the basis for protection in the 80376. Segments are used to encapsulate regions of memory which have common attributes. For example, all of the code of a given program could be contained in a segment, or an operating system table may reside in a segment. All information about each segment, is stored in an 8-byte data structure called a descriptor. All of the descriptors in a system are contained in tables recognized by hardware.

TERMINOLOGY

The following terms are used throughout the discussion of descriptors, privilege levels and protection:

PL: **Privilege Level**—One of the four hierarchical privilege levels. Level 0 is the most privileged level and level 3 is the least privileged.

RPL: **Requestor Privilege Level**—The privilege level of the original supplier of the selector. RPL is determined by the least two significant bits of a selector.

DPL: **Descriptor Privilege Level**—This is the least privileged level at which a task may access that descriptor (and the segment associated with that descriptor). Descriptor Privilege Level is determined by bits 6:5 in the Access Right Byte of a descriptor.

CPL: **Current Privilege Level**—The privilege level at which a task is currently executing, which equals the privilege level of the code segment being executed. CPL can also be determined by examining the lowest 2 bits of the CS register, except for conforming code segments.

EPL: **Effective Privilege Level**—The effective privilege level is the least privileged of the RPL and the DPL. EPL is the numerical maximum of RPL and DPL.

Task: One instance of the execution of a program. Tasks are also referred to as processes.

DESCRIPTOR TABLES

The descriptor tables define all of the segments which are used in an 80376 system. There are three types of tables on the 80376 which hold descriptors: the Global Descriptor Table, Local Descriptor Table, and the Interrupt Decriptor Table. All of the tables are variable length memory arrays, they can range in size between 8 bytes and 64 Kbytes. Each table can hold up to 8192 8-byte descriptors. The upper 13 bits of a selector are used as an index into the descriptor table. The tables have registers associated with them which hold the 32-bit linear base address, and the 16-bit limit of each table.

Each of the tables have a register associated with it: GDTR, LDTR and IDTR; see Figure 3.2. The LGDT, LLDT and LIDT instructions load the base and limit of the Global, Local and Interrupt Descriptor Tables into the appropriate register. The SGDT, SLDT and SIDT store these base and limit values. These are privileged instructions.

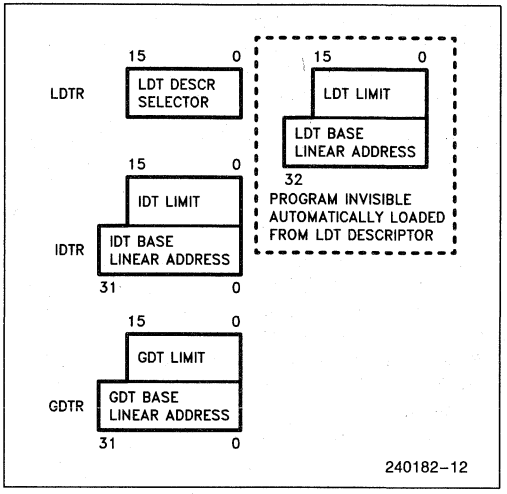


Figure 3.2. Descriptor Table Registers

Global Descriptor Table

The Global Descriptor Table (GDT) contains descriptors which are possibly available to all of the tasks in a system. The GDT can contain any type of segment descriptor except for interrupt and trap descriptors. Every 80376 system contains a GDT. A simple 80376 system contains only 2 entries in the GDT; a code and a data descriptor. For maximum performance, descriptor tables should begin on even addresses.

The first slot of the Global Descriptor Table corresponds to the null selector and is not used. The null selector defines a null pointer value.

Local Descriptor Table

LDTs contain descriptors which are associated with a given task. Generally, operating systems are designed so that each task has a separate LDT. The LDT may contain only code, data, stack, task gate, and call gate descriptors. LDTs provide a mechanism for isolating a given task's code and data segments from the rest of the operating system, while the GDT contains descriptors for segments which are common to all tasks. A segment cannot be accessed by a task if its segment descriptor does not exist in either the current LDT or the GDT. This pro-

vides both isolation and protection for a task's segments, while still allowing global data to be shared among tasks.

Unlike the 6-byte GDT or IDT registers which contain a base address and limit, the visible portion of the LDT register contains only a 16-bit selector. This selector refers to a Local Descriptor Table descriptor in the GDT (see Figure 2.1).

INTERRUPT DESCRIPTOR TABLE

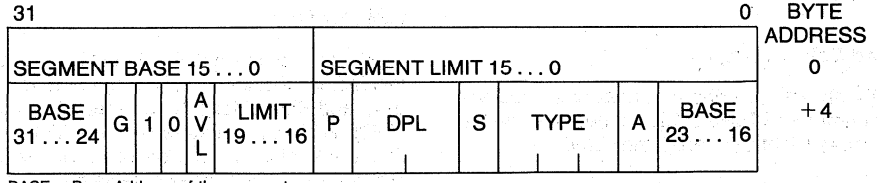
The third table needed for 80376 systems is the Interrupt Descriptor Table. The IDT contains the descriptors which point to the location of up to 256 interrupt service routines. The IDT may contain only task gates, interrupt gates and trap gates. The IDT should be at least 256 bytes in size in order to hold the descriptors for the 32 Intel Reserved Interrupts. Every interrupt used by a system must have an entry in the IDT. The IDT entries are referenced by INT instructions, external interrupt vectors, and exceptions.

DESCRIPTORS

The object to which the segment selector points to is called a descriptor. Descriptors are eight-byte quantities which contain attributes about a given region of linear address space. These attributes include the 32-bit logical base address of the segment, the 20-bit length and granularity of the segment, the protection level, read, write or execute privileges, and the type of segment. All of the attribute information about a segment is contained in 12 bits in the segment descriptor. Figure 3.3 shows the general format of a descriptor. All segments on the the 80376 have three attribute fields in common: the Present bit (P), the Descriptor Privilege Level bits (DPL) and the Segment bit (S). P=1 if the segment is loaded in physical memory, if P = 0 then any attempt to access the segment causes a not present exception (exception 11). The DPL is a two-bit field which specifies the protection level, 0–3, associated with a segment.

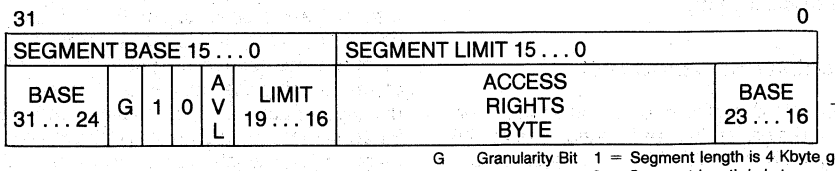
The 80376 has two main categories of segments: system segments, and non-system segments (for code and data). The segment bit, S, determines if a given segment is a system segment, a code segment or a data segment. If the S bit is 1 then the segment is either a code or data segment, if it is 0 then the segment is a system segment.

Note that although the 80376 is limited to a 16-Mbyte Physical address space (2^{24}), its base address allows a segment to be placed anywhere in a 4-Gbyte linear address space. When writing code for the 80376, users should keep code portability to an 80386 processor (or other processors with a larger physical address space) in mind. A segment base address can be placed anywhere in this 4-Gbyte linear address space, but a physical address will be



BASE Base Address of the segment
LIMIT The length of the segment
P Present Bit 1 = Present 0 = Not Present
DPL Descriptor Privilege Level 0–3
S Segment Descriptor: 0 = System Descriptor, 1 = Code or Data Descriptor
TYPE Type of Segment
A Accessed Bit
G Granularity Bit 1 = Segment length is 4 Kbyte Granular
0 = Segment length is byte granular
0 Bit must be zero (0) for compatibility with future processors
AVL Available field for user or OS

Figure 3.3. Segment Descriptors



G Granularity Bit 1 = Segment length is 4 Kbyte granular
0 = Segment length is byte granular
0 Bit must be zero (0) for compatibility with future processors
AVL Available field for user or OS

Figure 3.4. Code and Data Descriptors

Table 3.1. Access Rights Byte Definition for Code and Data Descriptors

Bit Position	Name	Function
7	Present (P)	P = 1 Segment is mapped into physical memory. P = 0 No mapping to physical memory exits
6–5	Descriptor Privilege Level (DPL)	Segment privilege attribute used in privilege tests.
4	Segment Descriptor (S)	S = 1 Code or Data (includes stacks) segment descriptor S = 0 System Segment Descriptor or Gate Descriptor
3	Executable (E)	E = 0 Descriptor type is data segment: (If Data Segment (S = 1, E = 0))
2	Expansion Direction (ED)	ED = 0 Expand up segment, offsets must be ≤ limit. ED = 1 Expand down segment, offsets must be > limit.
1	Writable (W)	W = 0 Data segment may not be written into. W = 1 Data segment may be written into.
3	Executable (E)	E = 1 Descriptor type is code segment: (If Code Segment (S = 1, E = 1))
2	Conforming (C)	C = 1 Code segment may only be executed when CPL ≥ DPL and CPL remains unchanged.
1	Readable (R)	R = 0 Code segment may not be read. R = 1 Code segment may be read.
0	Accessed (A)	A = 0 Segment has not been accessed. A = 1 Segment selector has been loaded into segment register or used by selector test instructions.

generated that is a truncated version of this linear address. Truncation will be to the maximum number of address bits. It is recommended to place EPROM at the highest physical address and DRAM at the lowest physical addresses.

Code and Data Descriptors (S = 1)

Figure 3.4 shows the general format of a code and data descriptor and Table 3.1 illustrates how the bits in the Access Right Byte are interpreted.

Code and data segments have several descriptor fields in common. The accessed bit, A, is set whenever the processor accesses a descriptor. The granularity bit, G, specifies if a segment length is 1-byte-granular or 4-Kbyte-granular. Base address bits 31–24, which are normally found in 80386 descriptors, are not made externally available on the 80376. They do not affect the operation of the 80376. The A_{31}–A_{24} field should be set to allow an 80386 to correctly execute with EPROM at the upper 4096 Mbytes of physical memory.

System Descriptor Formats (S = 0)

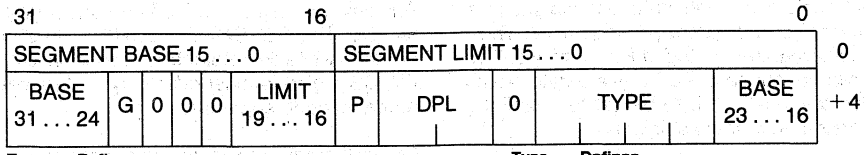
System segments describe information about operating system tables, tasks, and gates. Figure 3.5 shows the general format of system segment descriptors, and the various types of system segments. 80376 system descriptors (which are the same as 80386 descriptor types 2, 5, 9, B, C, E and F) contain a 32-bit logical base address and a 20-bit segment limit.

Selector Fields

A selector has three fields: Local or Global Descriptor Table Indicator (TI), Descriptor Entry Index (Index), and Requestor (the selector's) Privilege Level (RPL) as shown in Figure 3.6. The TI bit selects either the Global Descriptor Table or the Local Descriptor Table. The Index selects one of 8K descriptors in the appropriate descriptor table. The RPL bits allow high speed testing of the selector's privilege attributes.

Segment Descriptor Cache

In addition to the selector value, every segment register has a segment descriptor cache register associated with it. Whenever a segment register's contents are changed, the 8-byte descriptor associated with that selector is automatically loaded (cached) on the chip. Once loaded, all references to that segment use the cached descriptor information instead of reaccessing the descriptor. The contents of the descriptor cache are not visible to the programmer. Since descriptor caches only change when a segment register is changed, programs which modify the descriptor tables must reload the appropriate segment registers after changing a descriptor's value.



Type	Defines
0	Invalid
1	Reserved
2	LDT
3	Reserved
4	Reserved
5	Task Gate (80376/80386 Task)
6	Reserved
7	Reserved

Type	Defines
8	Invalid
9	Available 80376/80386 TSS
A	Undefined (Intel Reserved)
B	Busy 80376/80386 TSS
C	80376/80386 Call Gate
D	Undefined (Intel Reserved)
E	80376/80386 Interrupt Gate
F	80376/80386 Trap Gate

Figure 3.5. System Descriptors

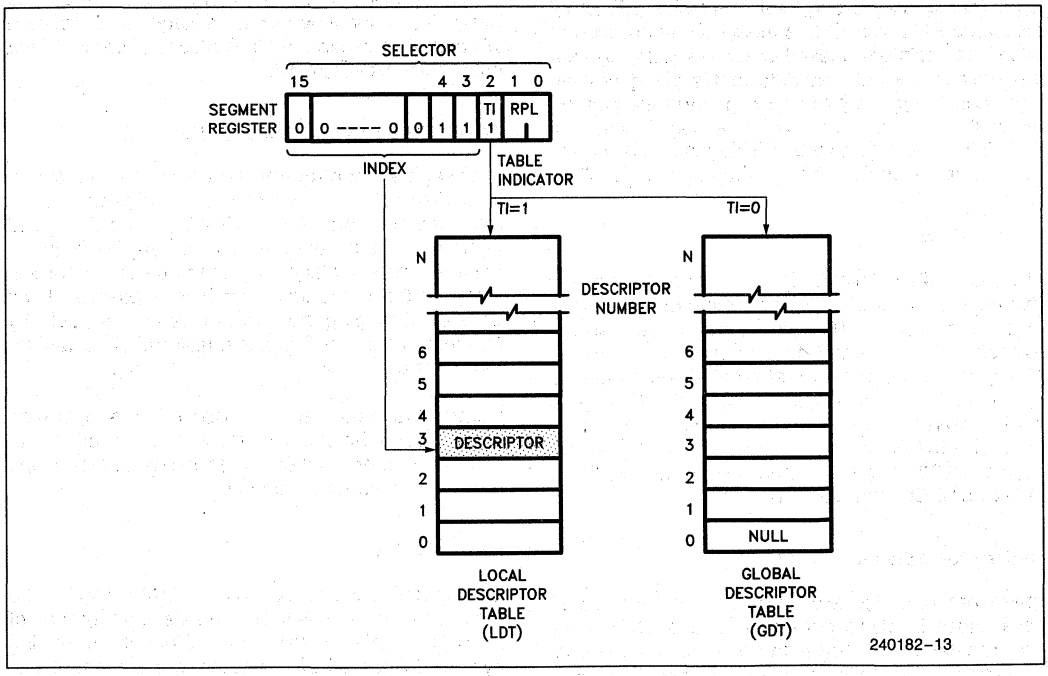


Figure 3.6. Example Descriptor Selection

3.3 Protection

The 80376 offers extensive protection features. These protection features are particularly useful in sophisticated embedded applications which use multitasking real-time operating systems. For simpler embedded applications these protection capabilities can be easily bypassed by making all applications run at privilege level (PL) 0.

RULES OF PRIVILEGE

The 80376 controls access to both data and procedures between levels of a task, according to the following rules.

—Data stored in a segment with privilege level **p** can be accessed only by code executing at a privilege level at least as privileged as **p.**

—A code segment/procedure with privilege level **p** can only be called by a task executing at the same or a lesser privilege level than **p.**

PRIVILEGE LEVELS

At any point in time, a task on the 80376 always executes at one of the four privilege levels. The Current Privilege Level (CPL) specifies what the task's privilege level is. A task's CPL may only be changed

by control transfers through gate descriptors to a code segment with a different privilege level. Thus, an application program running at PL = 3 may call an operating system routine at PL = 1 (via a gate) which would cause the task's CPL to be set to 1 until the operating system routine was finished.

Selector Privilege (RPL)

The privilege level of a selector is specified by the RPL field. The selector's RPL is only used to establish a less trusted privilege level than the current privilege level of the task for the use of a segment. This level is called the task's effective privilege level (EPL). The EPL is defined as being the least privileged (numerically larger) level of a task's CPL and a selector's RPL. The RPL is most commonly used to verify that pointers passed to an operating system procedure do not access data that is of higher privilege than the procedure that originated the pointer. Since the originator of a selector can specify any RPL value, the Adjust RPL (ARPL) instruction is provided to force the RPL bits to the originator's CPL.

I/O Privilege

The I/O privilege level (IOPL) lets the operating system code executing at CPL = 0 define the least privileged level at which I/O instructions can be used. An exception 13 (General Protection Violation) is generated if an I/O instruction is attempted when the CPL of the task is less privileged than the IOPL. The IOPL is stored in bits 13 and 14 of the EFLAGS register. The following instructions cause an exception 13 if the CPL is greater than IOPL: IN, INS, OUT, OUTS, STI, CLI and LOCK prefix.

Descriptor Access

There are basically two types of segment accessess: those involving code segments such as control transfers, and those involving data accesses. Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL as described above.

Any time an instruction loads a data segment register (DS, ES, FS, GS) the 80376 makes protection validation checks. Selectors loaded in the DS, ES, FS, GS registers must refer only to data segment or readable code segments.

Finally the privilege validation checks are performed. The CPL is compared to the EPL and if the EPL is more privileged than the CPL, an exception 13 (general protection fault) is generated.

The rules regarding the stack segment are slightly different than those involving data segments. Instructions that load selectors into SS must refer to data segment descriptors for writeable data segments. The DPL and RPL must equal the CPL of all other descriptor types or a privilege level violation will cause an exception 13. A stack not present fault causes an exception 12.

PRIVILEGE LEVEL TRANSFERS

Inter-segment control transfers occur when a selector is loaded in the CS register. For a typical system most of these transfers are simply the result of a call or a jump to another routine. There are five types of control transfers which are summarized in Table 3.2. Many of these transfers result in a privilege level transfer. Changing privilege levels is done only by control transfers, using gates, task switches, and interrupt or trap gates.

Control transfers can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules will cause an exception 13.

CALL GATES

Gates provide protected indirect CALLs. One of the major uses of gates is to provide a secure method of privilege transfers within a task. Since the operating system defines all of the gates in a system, it can ensure that all gates only allow entry into a few trusted procedures.

Table 3.2. Descriptor Types Used for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL	CALL	Call Gate	GDT/LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
	CALL, JMP	Task State Segment	GDT
Task Switch	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag register) = 0
**NT (Nested Task bit of flag register) = 1

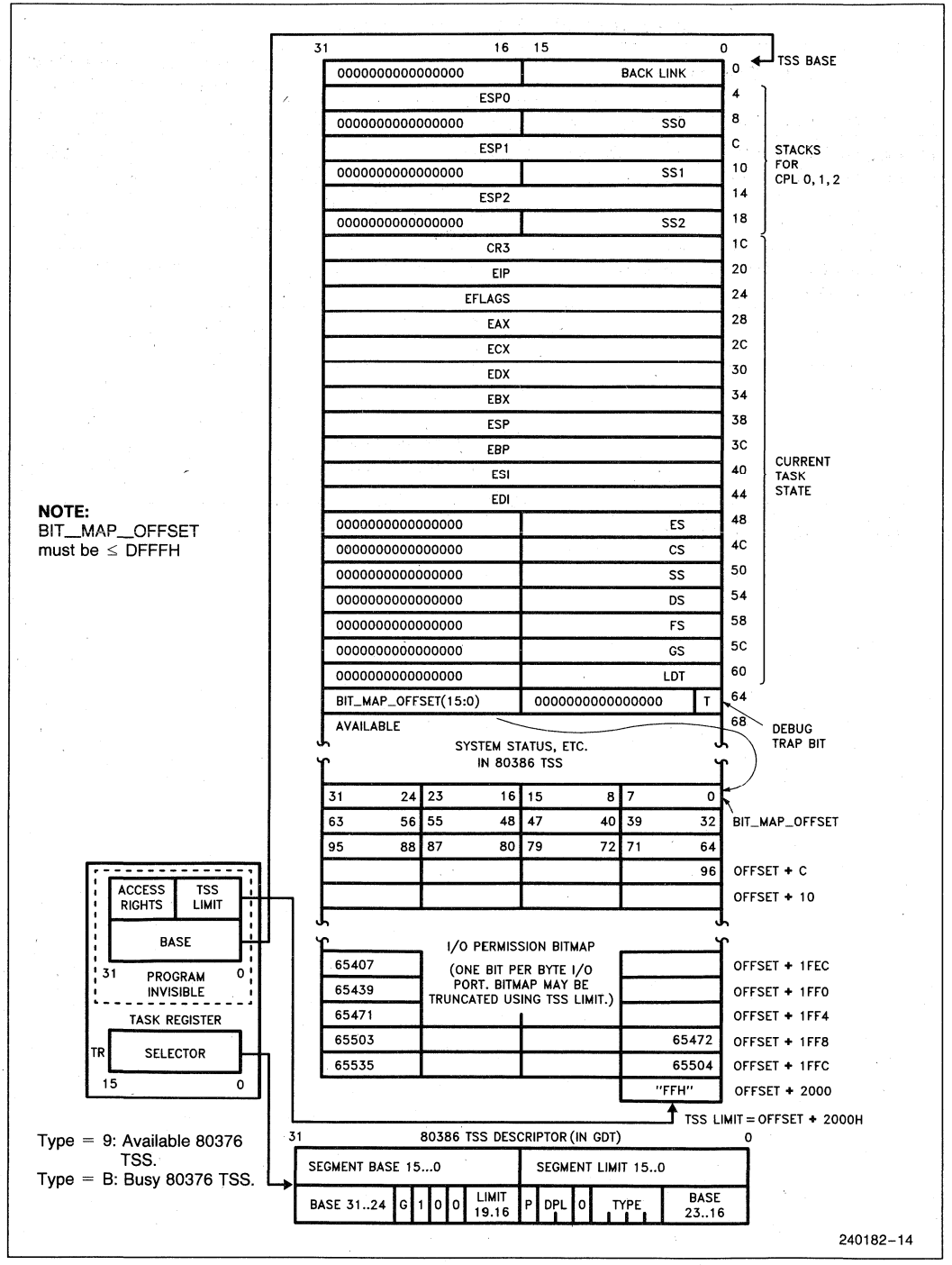


Figure 3.7. 80376 TSS And TSS Registers

TASK SWITCHING

A very important attribute of any multi-tasking operating system is its ability to rapidly switch between tasks or processes. The 80376 directly supports this operation by providing a task switch instruction in hardware. The 80376 task switch operation saves the entire state of the machine (all of the registers, address space, and a link to the previous task), loads a new execution state, performs protection checks, and commences execution in the new task. Like transfer of control by gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS), or a task gate descriptor in the GDT or LDT. An INT n instruction, exception, trap or external interrupt may also invoke the task switch operation if there is a task gate descriptor in the associated IDT descriptor slot. For simple applications, the TSS and task switching may not be used. The TSS or task switch will not be used or occur if no task gates are present in the GDT, LDT or IDT.

The TSS descriptor points to a segment (see Figure 3.7) containing the entire 80376 execution state. A task gate descriptor contains a TSS selector. The limit of an 80376 TSS must be greater than 64H, and can be as large as 16 Mbytes. In the additional TSS space, the operating system is free to store additional information as the reason the task is inactive, the time the task has spent running, and open files belonging to the task. For maximum performance, TSS should start on an even address.

Each Task must have a TSS associated with it. The current TSS is identified by a special register in the 80376 called the Task State Segment Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with the TSS descriptor is loaded whenever TR is loaded with a new selector. Returning from a task is accomplished by the IRET instruction. When IRET is executed, control is returned to the task which was interrupted. The current executing task's state is saved in the TSS and the old task state is restored from its TSS.

Several bits in the flag register and CR0 register give information about the state of a task which is useful to the operating system. The Nested Task bit, NT, controls the function of the IRET instruction. If NT = 0 the IRET instruction performs the regular return. If NT = 1, IRET performs a task switch operation back to the previous task. The NT bit is set or reset in the following fashion:

When a CALL or INT instruction initiates a task switch, the new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT (The NT bit will be restored after execution of the interrupt handler). NT may also be set or cleared by POPF or IRET instructions.

The 80376 task state segment is marked busy by changing the descriptor type field from TYPE 9 to TYPE 0BH. Use of a selector that references a busy task state segment causes an exception 13.

The coprocessor's state is not automatically saved when a task switch occurs. The Task Switched Bit, TS, in the CR0 register helps deal with the coprocessor's state in a multi-tasking environment. Whenever the 80376 switches tasks, it sets the TS bit. The 80376 detects the first use of a processor extension instruction after a task switch and causes the processor extension not available exception 7. The exception handler for exception 7 may then decide whether to save the state of the coprocessor.

The T bit in the 80376 TSS indicates that the processor should generate a debug exception when switching to a task. If T = 1 then upon entry to a new task a debug exception 1 will be generated.

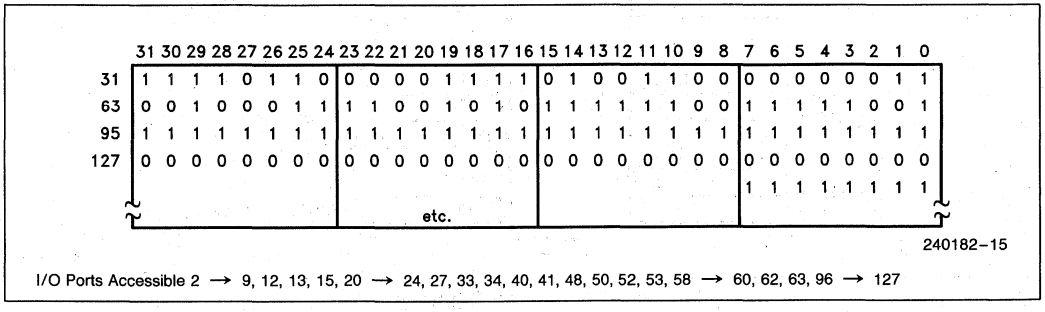


Figure 3.8. Sample I/O Permission Bit Map

PROTECTION AND I/O PERMISSION BIT MAP

The I/O instructions that directly refer to addresses in the processor's I/O space are IN, INS, OUT and OUTS. The 80376 has the ability to selectively trap references to specific I/O addresses. The structure that enables selective trapping is the ***I/O Permission Bit Map*** in the TSS segment (see Figures 3.7 and 3.8). The I/O permission map is a bit vector. The size of the map and its location in the TSS segment are variable. The processor locates the I/O permission map by means of the **I/O map base** field in the fixed portion of the TSS. The **I/O map base** field is 16 bits wide and contains the offset of the beginning of the I/O permission map.

If an I/O instruction (IN, INS, OUT or OUTS) is encountered, the processor first checks whether CPL $\leq$ IOPL. If this condition is true, the I/O operation may proceed. If not true, the processor checks the I/O permission map.

Each bit in the map corresponds to an I/O port byte address; for example, the bit for port 41 is found at **I/O map base** $+5$ linearly, ($5 \times 8 = 40$), bit offset 1. The processor tests all the bits that correspond to the I/O addresses spanned by an I/O operation; for example, a double word operation tests four bits corresponding to four adjacent byte addresses. If any tested bit is set, the processor signals a general protection exception. If all the tested bits are zero, the I/O operations may proceed.

It is not necessary for the I/O permission map to represent all the I/O addresses. I/O addresses not spanned by the map are treated as if they had one-bits in the map. The **I/O map base** should be at least one byte less than the TSS limit and the last byte beyond the I/O mapping information must contain all 1's.

Because the I/O permission map is in the TSS segment, different tasks can have different maps. Thus, the operating system can allocate ports to a task by changing the I/O permission map in the task's TSS.

IMPORTANT IMPLEMENTATION NOTE:
Beyond the last byte of I/O mapping information in the I/O permission bit map **must** be a byte containing all 1's. The byte of all 1's must be within the limit of the 80376's TSS segment (see Figure 3.7).

4.0 FUNCTIONAL DATA

The Intel 80376 embedded processor features a straightforward functional interface to the external hardware. The 80376 has separate parallel buses for data and address. The data bus is 16 bits in width, and bidirectional. The address bus outputs 24-bit address values using 23 address lines and two-byte enable signals.

The 80376 has two selectable address bus cycles: pipelined and non-pipelined. The pipelining option allows as much time as possible for data access by

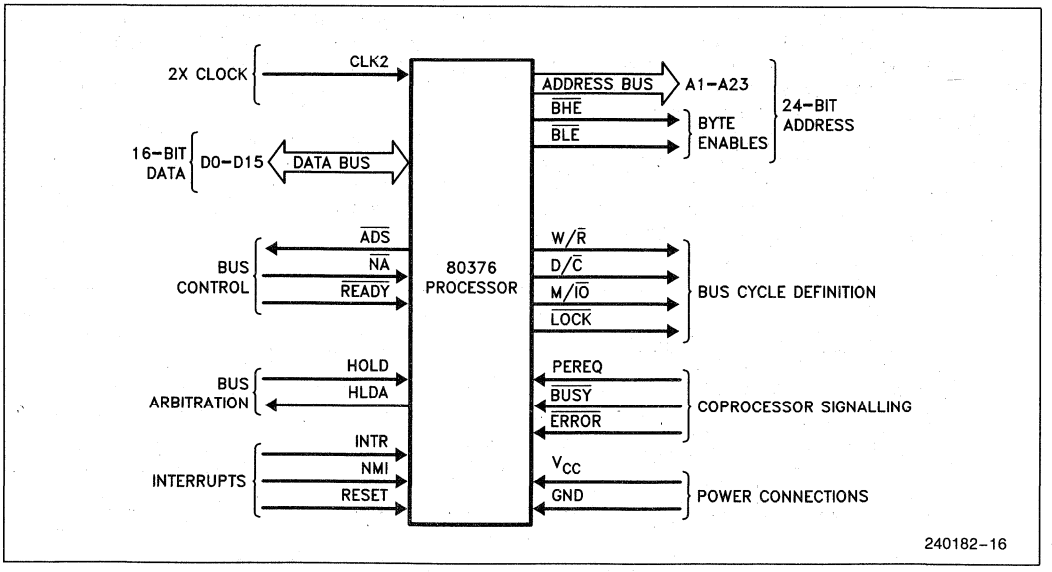


Figure 4.1. Functional Signal Groups

starting the pending bus cycle before the present bus cycle is finished. A non-pipelined bus cycle gives the highest bus performance by executing every bus cycle in two processor clock cycles. For maximum design flexibility, the address pipelining option is selectable on a cycle-by-cycle basis.

The processor's bus cycle is the basic mechanism for information transfer, either from system to processor, or from processor to system. 80376 bus cycles perform data transfer in a minimum of only two clock periods. On a 16-bit data bus, the maximum 80376 transfer bandwidth at 16 MHz is therefore 16 Mbytes/sec. However, any bus cycle will be extended for more than two clock periods if external hardware withholds acknowledgement of the cycle.

The 80376 can relinquish control of its local buses to allow mastership by other devices, such as direct memory access (DMA) channels. When relinquished, HLDA is the only output pin driven by the 80376, providing near-complete isolation of the processor from its system (all other output pins are in a float condition).

4.1 Signal Description Overview

Ahead is a brief description of the 80376 input and output signals arranged by functional groups.

The signal descriptions sometimes refer to A.C. timing parameters, such as "t_{25} Reset Setup Time" and "t_{26} Reset Hold Time." The values of these parameters can be found in Tables 6.4 and 6.5.

CLOCK (CLK2)

CLK2 provides the fundamental timing for the 80376. It is divided by two internally to generate the internal processor clock used for instruction execution. The internal clock is comprised of two

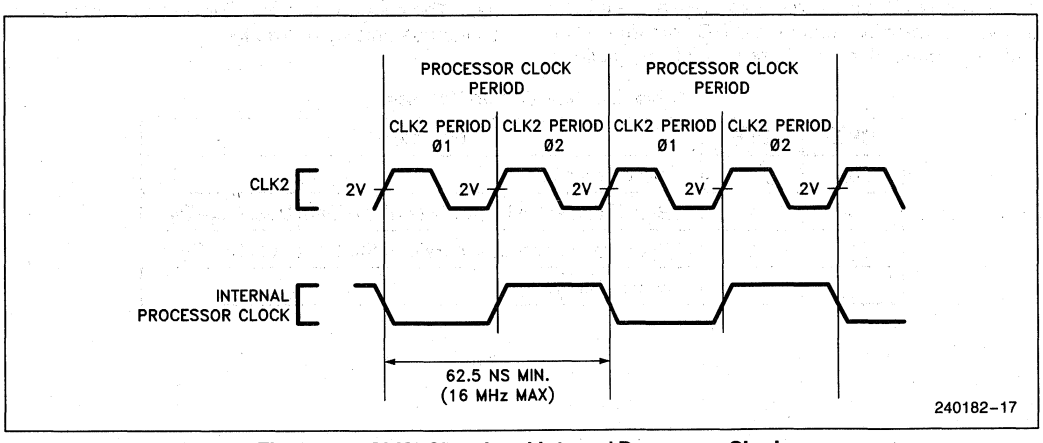


Figure 4.2. CLK2 Signal and Internal Processor Clock

phases, "phase one" and "phase two". Each CLK2 period is a phase of the internal clock. Figure 4.2 illustrates the relationship. If desired, the phase of the internal processor clock can be synchronized to a known phase by ensuring the falling edge of the RESET signal meets the applicable setup and hold times t_{25} and t_{26}.

DATA BUS (D_{15}–D_0)

These three-state bidirectional signals provide the general purpose data path between the 80376 and other devices. The data bus outputs are active HIGH and will float during bus hold acknowledge. Data bus reads require that read-data setup and hold times t_{21} and t_{22} be met relative to CLK2 for correct operation.

ADDRESS BUS ($\overline{BHE}$, $\overline{BLE}$, A_{23}–A_1)

These three-state outputs provide physical memory addresses or I/O port addresses. A_{23}–A_{16} are LOW during I/O transfers except for I/O transfers automatically generated by coprocessor instructions. During coprocessor I/O transfers, A_{22}–A_{16} are driven LOW, and A_{23} is driven HIGH so that this address line can be used by external logic to generate the coprocessor select signal. Thus, the I/O address driven by the 80376 for coprocessor commands is 8000F8H, and the I/O address driven by the 80376 processor for coprocessor data is 8000FCH or 8000FEH.

The address bus is capable of addressing 16 Mbytes of physical memory space (000000H through 0FFFFFFH), and 64 Kbytes of I/O address space (000000H through 00FFFFH) for programmed I/O. The address bus is active HIGH and will float during bus hold acknowledge.

The Byte Enable outputs $\overline{BHE}$ and $\overline{BLE}$ directly indicate which bytes of the 16-bit data bus are involved with the current transfer. $\overline{BHE}$ applies to D_{15}–D_8 and $\overline{BLE}$ applies to D_7–D_0. If both $\overline{BHE}$ and $\overline{BLE}$ are asserted, then 16 bits of data are being transferred. See Table 4.1 for a complete decoding of these signals. The byte enables are active LOW and will float during bus hold acknowledge.

Table 4.1. Byte Enable Definitions

$\overline{BHE}$	$\overline{BLE}$	Function
0	0	Word Transfer
0	1	Byte Transfer on Upper Byte of the Data Bus, D_{15}–D_8
1	0	Byte Transfer on Lower Byte of the Data Bus, D_7–D_0
1	1	Never Occurs

BUS CYCLE DEFINITION SIGNALS (W/R, D/C, M/IO, LOCK)

These three-state outputs define the type of bus cycle being performed: W/R distinguishes between write and read cycles, D/C distinguishes between data and control cycles, M/IO distinguishes between memory and I/O cycles, and LOCK distinguishes between locked and unlocked bus cycles. All of these signals are active LOW and will float during bus acknowledge.

The primary bus cycle definition signals are W/R, D/C and M/IO, since these are the signals driven valid as ADS (Address Status output) becomes active. The LOCK signal is driven valid at the same time the bus cycle begins, which due to address pipelining, could be after ADS becomes active. Exact bus cycle definitions, as a function of W/R, D/C and M/IO are given in Table 4.2.

LOCK indicates that other system bus masters are not to gain control of the system bus while it is active. LOCK is activated on the CLK2 edge that begins the first locked bus cycle (i.e., it is not active at the same time as the other bus cycle definition pins) and is deactivated when ready is returned to the end of the last bus cycle which is to be locked. The beginning of a bus cycle is determined when READY is returned in a previous bus cycle and another is pending (ADS is active) or the clock in which ADS is driven active if the bus was idle. This means that it follows more closely with the write data rules when it is valid, but may cause the bus to be locked longer than desired. The LOCK signal may be explicitly activated by the LOCK prefix on certain instructions. LOCK is always asserted when executing the XCHG instruction, during descriptor updates, and during the interrupt acknowledge sequence.

BUS CONTROL SIGNALS (ADS, READY, NA)

The following signals allow the processor to indicate when a bus cycle has begun, and allow other system hardware to control address pipelining and bus cycle termination.

Address Status (ADS)

This three-state output indicates that a valid bus cycle definition and address (W/R, D/C, M/IO, BHE, BLE and A_{23}–A_1) are being driven at the 80376 pins. ADS is an active LOW output. Once ADS is driven active, valid address, byte enables, and definition signals will not change. In addition, ADS will remain active until its associated bus cycle begins (when READY is returned for the previous bus cycle when running pipelined bus cycles). ADS will float during bus hold acknowledge. See sections **Non-Pipelined Bus Cycles** and **Pipelined Bus Cycles** for additional information on how ADS is asserted for different bus states.

Transfer Acknowledge (READY)

This input indicates the current bus cycle is complete, and the active bytes indicated by BHE and BLE are accepted or provided. When READY is sampled active during a read cycle or interrupt acknowledge cycle, the 80376 latches the input data and terminates the cycle. When READY is sampled active during a write cycle, the processor terminates the bus cycle.

5

Table 4.2. Bus Cycle Definition

M/IO	D/C	W/R	Bus Cycle Type	Locked?
0	0	0	INTERRUPT ACKNOWLEDGE	Yes
0	0	1	Does Not Occur	—
0	1	0	I/O DATA READ	No
0	1	1	I/O DATA WRITE	No
1	0	0	MEMORY CODE READ	No
1	0	1	HALT: Address = 2, BHE = 1, BLE = 0 / SHUTDOWN: Address = 0, BHE = 1, BLE = 0	No
1	1	0	MEMORY DATA READ	Some Cycles
1	1	1	MEMORY DATA WRITE	Some Cycles

READY is ignored on the first bus state of all bus cycles, and sampled each bus state thereafter until asserted. READY must eventually be asserted to acknowledge every bus cycle, including Halt Indication and Shutdown Indication bus cycles. When being sampled, READY must always meet setup and hold times t_{19} and t_{20} for correct operation.

Next Address Request (NA)

This is used to request pipelining. This input indicates the system is prepared to accept new values of BHE, BLE, A_{23}–A_1, W/R, D/C and M/IO from the 80376 even if the end of the current cycle is not being acknowledged on READY. If this input is active when sampled, the next bus cycle's address and status signals are driven onto the bus, provided the next bus request is already pending internally. NA is ignored in clock cycles in which ADS or READY is activated. This signal is active LOW and must satisfy setup and hold times t_{15} and t_{16} for correct operation. See **Pipelined Bus Cycles** and **Read and Write Cycles** for additional information.

BUS ARBITRATION SIGNALS (HOLD, HLDA)

This section describes the mechanism by which the processor relinquishes control of its local buses when requested by another bus master device. See **Entering and Exiting Hold Acknowledge** for additional information.

Bus Hold Request (HOLD)

This input indicates some device other than the 80376 requires bus mastership. When control is granted, the 80376 floats A_{23}–A_1, BHE, BLE, D_{15}–D_0, LOCK, M/IO, D/C, W/R and ADS, and then activates HLDA, thus entering the bus hold acknowledge state. The local bus will remain granted to the requesting master until HOLD becomes inactive. When HOLD becomes inactive, the 80376 will deactivate HLDA and drive the local bus (at the same time), thus terminating the hold acknowledge condition.

HOLD must remain asserted as long as any other device is a local bus master. External pull-up resistors may be required when in the hold acknowledge state since none of the 80376 floated outputs have internal pull-up resistors. See **Resistor Recommendations** for additional information. HOLD is not recognized while RESET is active but is recognized during the time between the high-to-low transistion of RESET and the first instruction fetch. If RESET is asserted while HOLD is asserted, RESET has priority and places the bus into an idle state, rather than the hold acknowledge (high-impedance) state.

HOLD is a level-sensitive, active HIGH, synchronous input. HOLD signals must always meet setup and hold times t_{23} and t_{24} for correct operation.

Bus Hold Acknowledge (HLDA)

When active (HIGH), this output indicates the 80376 has relinquished control of its local bus in response to an asserted HOLD signal, and is in the bus Hold Acknowledge state.

The Bus Hold Acknowledge state offers near-complete signal isolation. In the Hold Acknowledge state, HLDA is the only signal being driven by the 80376. The other output signals or bidirectional signals (D_{15}–D_0, BHE, BLE, A_{23}–A_1, W/R, D/C, M/IO, LOCK and ADS) are in a high-impedance state so the requesting bus master may control them. These pins remain OFF throughout the time that HLDA remains active (see Table 4.3). Pull-up resistors may be desired on several signals to avoid spurious activity when no bus master is driving them. See **Resistor Recommendations** for additional information.

When the HOLD signal is made inactive, the 80376 will deactivate HLDA and drive the bus. One rising edge on the NMI input is remembered for processing after the HOLD input is negated.

Table 4.3. Output Pin State during HOLD

Pin Value	Pin Names
1	HLDA
Float	LOCK, M/IO, D/C, W/R, ADS, A_{23}–A_1, BHE, BLE, D_{15}–D_0

Hold Latencies

The maximum possible HOLD latency depends on the software being executed. The actual HOLD latency at any time depends on the current bus activity, the state of the LOCK signal (internal to the CPU) activated by the LOCK prefix, and interrupts. The 80376 will not honor a HOLD request until the current bus operation is complete.

The 80376 breaks 32-bit data or I/O accesses into 2 internally locked 16-bit bus cycles; the LOCK signal is not asserted. The 80376 breaks unaligned 16-bit or 32-bit data or I/O accesses into 2 or 3 internally locked 16-bit bus cycles. Again the LOCK signal is not asserted but a HOLD request will not be recognized until the end of the entire transfer.

Wait states affect HOLD latency. The 80376 will not honor a HOLD request until the end of the current bus operation, no matter how many wait states are required. Systems with DMA where data transfer is critical must insure that $\overline{\text{READY}}$ returns sufficiently soon.

COPROCESSOR INTERFACE SIGNALS (PEREQ, $\overline{\text{BUSY}}$, $\overline{\text{ERROR}}$)

In the following sections are descriptions of signals dedicated to the numeric coprocessor interface. In addition to the data bus, address bus, and bus cycle definition signals, these following signals control communication between the 80376 and the 80387SX processor extension.

Coprocessor Request (PEREQ)

When asserted (HIGH), this input signal indicates a coprocessor request for a data operand to be transferred to/from memory by the 80376. In response, the 80376 transfers information between the coprocessor and memory. Because the 80376 has internally stored the coprocessor opcode being executed, it performs the requested data transfer with the correct direction and memory address.

PEREQ is a level-sensitive active HIGH asynchronous signal. Setup and hold times, t_{29} and t_{30}, relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. This signal is provided with a weak internal pull-down resistor of around 20 KΩ to ground so that it will not float active when left unconnected.

Coprocessor Busy ($\overline{\text{BUSY}}$)

When asserted (LOW), this input indicates the coprocessor is still executing an instruction, and is not yet able to accept another. When the 80376 encounters any coprocessor instruction which operates on the numerics stack (e.g. load, pop, or arithmetic operation), or the WAIT instruction, this input is first automatically sampled until it is seen to be inactive. This sampling of the $\overline{\text{BUSY}}$ input prevents overrunning the execution of a previous coprocessor instruction.

The F(N)INIT, F(N)CLEX coprocessor instructions are allowed to execute even if $\overline{\text{BUSY}}$ is active, since these instructions are used for coprocessor initialization and exception-clearing.

$\overline{\text{BUSY}}$ is an active LOW, level-sensitive asynchronous signal. Setup and hold times, t_{29} and t_{30}, relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. This pin is provided with a weak internal pull-up resistor of around 20 KΩ to V_{CC} so that it will not float active when left unconnected.

$\overline{\text{BUSY}}$ serves an additional function. If $\overline{\text{BUSY}}$ is sampled LOW at the falling edge of RESET, the 80376 processor performs an internal self-test (see **Bus Activity During and Following Reset.** If $\overline{\text{BUSY}}$ is sampled HIGH, no self-test is performed.

Coprocessor Error ($\overline{\text{ERROR}}$)

When asserted (LOW), this input signal indicates that the previous coprocessor instruction generated a coprocessor error of a type not masked by the coprocessor's control register. This input is automatically sampled by the 80376 when a coprocessor instruction is encountered, and if active, the 80376 generates exception 16 to access the error-handling software.

Several coprocessor instructions, generally those which clear the numeric error flags in the coprocessor or save coprocessor state, do execute without the 80376 generating exception 16 even if $\overline{\text{ERROR}}$ is active. These instructions are FNINIT, FNCLEX, FNSTSW, FNSTSWAX, FNSTCW, FNSTENV and FNSAVE.

$\overline{\text{ERROR}}$ is an active LOW, level-sensitive asynchronous signal. Setup and hold times t_{29} and t_{30}, relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. This pin is provided with a weak internal pull-up resistor of around 20 KΩ to V_{CC} so that it will not float active when left unconnected.

INTERRUPT SIGNALS (INTR, NMI, RESET)

The following descriptions cover inputs that can interrupt or suspend execution of the processor's current instruction stream.

Maskable Interrupt Request (INTR)

When asserted, this input indicates a request for interrupt service, which can be masked by the 80376 Flag Register IF bit. When the 80376 responds to the INTR input, it performs two interrupt acknowledge bus cycles and, at the end of the second, latches an 8-bit interrupt vector on D_7–D_0 to identify the source of the interrupt.

INTR is an active HIGH, level-sensitive asynchronous signal. Setup and hold times, t_{27} and t_{28}, relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. To assure recognition of an INTR request, INTR should remain active until the first interrupt acknowledge bus cycle begins. INTR is sampled at the beginning of every instruction. In order to be recognized at a particular instruction boundary, INTR must be active at least eight CLK2 clock periods before the beginning of the execution of the instruction. If recognized, the 80376 will begin execution of the interrupt.

Non-Maskable Interrupt Request (NMI)

This input indicates a request for interrupt service which cannot be masked by software. The non-maskable interrupt request is always processed according to the pointer or gate in slot 2 of the interrupt table. Because of the fixed NMI slot assignment, no interrupt acknowledge cycles are performed when processing NMI.

NMI is an active HIGH, rising edge-sensitive asynchronous signal. Setup and hold times, t_{27} and t_{28}, relative to the CLK2 signal must be met to guarantee recognition at a particular clock edge. To assure recognition of NMI, it must be inactive for at least eight CLK2 periods, and then be active for at least eight CLK2 periods before the beginning of the execution of an instruction.

Once NMI processing has begun, no additional NMI's are processed until after the next IRET instruction, which is typically the end of the NMI service routine. If NMI is re-asserted prior to that time, however, one rising edge on NMI will be remembered for processing after executing the next IRET instruction.

Interrupt Latency

The time that elapses before an interrupt request is serviced (interrupt latency) varies according to several factors. This delay must be taken into account by the interrupt source. Any of the following factors can affect interrupt latency:

1. If interrupts are masked, and INTR request will not be recognized until interrupts are reenabled.
2. If an NMI is currently being serviced, an incoming NMI request will not be recognized until the 80376 encounters the IRET instruction.
3. An interrupt request is recognized only on an instruction boundary of the 80376 ***Execution Unit*** except for the following cases:
   - — Repeat string instructions can be interrupted after each iteration.
   - — If the instruction loads the Stack Segment register, an interrupt is not processed until after the following instruction, which should be an ESP load. This allows the entire stack pointer to be loaded without interruption.
   - — If an instruction sets the interrupt flag (enabling interrupts), an interrupt is not processed until after the next instruction.

   The longest latency occurs when the interrupt request arrives while the 80376 processor is executing a long instruction such as multiplication, division or a task-switch.
4. Saving the Flags register and CS:EIP registers.
5. If interrupt service routine requires a task switch, time must be allowed for the task switch.
6. If the interrupt service routine saves registers that are not automatically saved by the 80376.

RESET

This input signal suspends any operation in progress and places the 80376 in a known reset state. The 80376 is reset by asserting RESET for 15 or more CLK2 periods (80 or more CLK2 periods before requesting self-test). When RESET is active, all other input pins except $\overline{FLT}$ are ignored, and all other bus pins are driven to an idle bus state as shown in Table 4.4. If RESET and HOLD are both active at a point in time, RESET takes priority even if the 80376 was in a Hold Acknowledge state prior to RESET active.

RESET is an active HIGH, level-sensitive synchronous signal. Setup and hold times, t_{25} and t_{26}, must be met in order to assure proper operation of the 80376.

Table 4.4. Pin State (Bus Idle) during RESET

Pin Name	Signal Level during RESET
$\overline{ADS}$	1
$D_{15}-D_0$	Float
$\overline{BHE}$, $\overline{BLE}$	0
$A_{23}-A_1$	1
W/$\overline{R}$	0
D/$\overline{C}$	1
M/$\overline{IO}$	0
$\overline{LOCK}$	1
HLDA	0

4.2 Bus Transfer Mechanism

All data transfers occur as a result of one or more bus cycles. Logical data operands of byte and word lengths may be transferred without restrictions on physical address alignment. Any byte boundary may be used, although two physical bus cycles are performed as required for unaligned operand transfers.

The 80376 processor address signals are designed to simplify external system hardware. $\overline{BHE}$ and $\overline{BLE}$ provide linear selects for the two bytes of the 16-bit data bus.

Byte Enable outputs $\overline{BHE}$ and $\overline{BLE}$ are asserted when their associated data bus bytes are involved with the present bus cycle, as listed in Table 4.5.

Table 4.5. Byte Enables and Associated Data and Operand Bytes

Byte Enable	Associated Data Bus Signals
$\overline{BHE}$	$D_{15}-D_8$ (Byte 1—Most Significant)
$\overline{BLE}$	D_7-D_0 (Byte 0—Least Significant)

Each bus cycle is composed of at least two bus states. Each bus state requires one processor clock period. Additional bus states added to a single bus cycle are called wait states. See **Bus Functional Description** for additional information.

4.3 Memory and I/O Spaces

Bus cycles may access physical memory space or I/O space. Peripheral devices in the system may either be memory-mapped, or I/O-mapped, or both. As shown in Figure 4.3, physical memory addresses range from 000000H to 0FFFFFFH (16 Mbytes) and I/O addresses from 000000H to 00FFFFH (64 Kbytes). Note the I/O addresses used by the automatic I/O cycles for coprocessor communication are 8000F8H to 8000FFH, beyond the address range of programmed I/O, to allow easy generation of a coprocessor chip select signal using the A_{23} and M/$\overline{IO}$ signals.

OPERAND ALIGNMENT

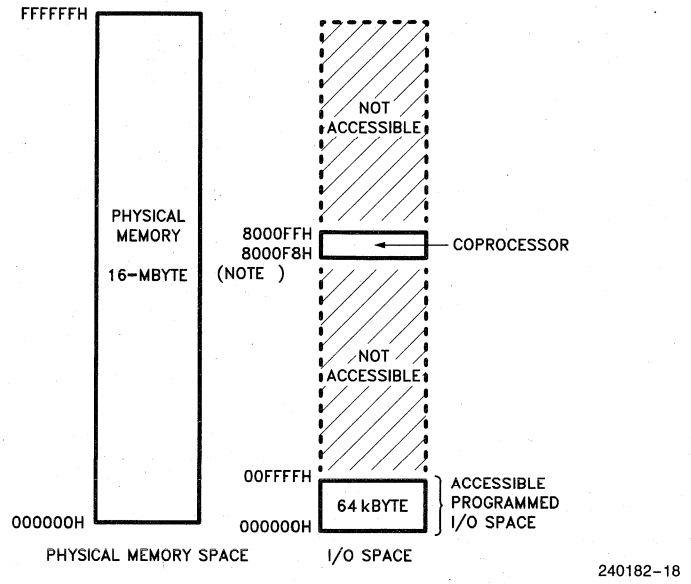
With the flexibility of memory addressing on the 80376, it is possible to transfer a logical operand that spans more than one physical Dword or word of memory or I/O. Examples are 32-bit Dword or 16-bit word operands beginning at addresses not evenly divisible by 2.

Operand alignment and size dictate when multiple bus cycles are required. Table 4.6 describes the transfer cycles generated for all combinations of logical operand lengths and alignment.

Table 4.6. Transfer Bus Cycles for Bytes, Words and Dwords

	Byte-Length of Logical Operand								
	1	2				4			
Physical Byte Address in Memory (Low-Order Bits)	xx	00	01	10	11	00	01	10	11
Transfer Cycles	b	w	lb, hb	w	hb, l,b	lw, hw	hb, lb, mw	hw, lw	mw, hb, lb

Key: b = byte transfer
w = word transfer
l = low-order portion
m = mid-order portion
x = don't care
h = high-order portion



NOTE:
Since A_{23} is HIGH during automatic communication with coprocessor, A_{23} HIGH and M/$\overline{IO}$ LOW can be used to easily generate a coprocessor select signal.

Figure 4.3. Physical Memory and I/O Spaces

4.4 Bus Functional Description

The 80376 has separate, parallel buses for data and address. The data bus is 16 bits in width, and bidirectional. The address bus provides a 24-bit value using 23 signals for the 23 upper-order address bits and 2 Byte Enable signals to directly indicate the active bytes. These buses are interpreted and controlled by several definition signals.

The definition of each bus cycle is given by three signals: M/$\overline{IO}$, W/$\overline{R}$ and D/$\overline{C}$. At the same time, a valid address is present on the byte enable signals, $\overline{BHE}$ and $\overline{BLE}$, and the other address signals A_{23}–A_1. A status signal, $\overline{ADS}$, indicates when the 80376 issues a new bus cycle definition and address.

Collectively, the address bus, data bus and all associated control signals are referred to simply as "the bus". When active, the bus performs one of the bus cycles below:

1. Read from memory space
2. Locked read from memory space
3. Write to memory space
4. Locked write to memory space
5. Read from I/O space (or coprocessor)
6. Write to I/O space (or coprocessor)
7. Interrupt acknowledge (always locked)
8. Indicate halt, or indicate shutdown

Table 4.2 shows the encoding of the bus cycle definition signals for each bus cycle. See **Bus Cycle Definition Signals** for additonal information.

When the 80376 bus is not performing one of the activities listed above, it is either Idle or in the Hold Acknowledge state, which may be detected by external circuitry. The idle state can be identified by the 80376 giving no further assertions on its address strobe output ($\overline{ADS}$) since the beginning of its most recent bus cycle, and the most recent bus cycle having been terminated. The hold acknowledge state is identified by the 80376 asserting its hold acknowledge (HLDA) output.

The shortest time unit of bus activity is a bus state. A bus state is one processor clock period (two CLK2 periods) in duration. A complete data transfer occurs during a bus cycle, composed of two or more bus states.

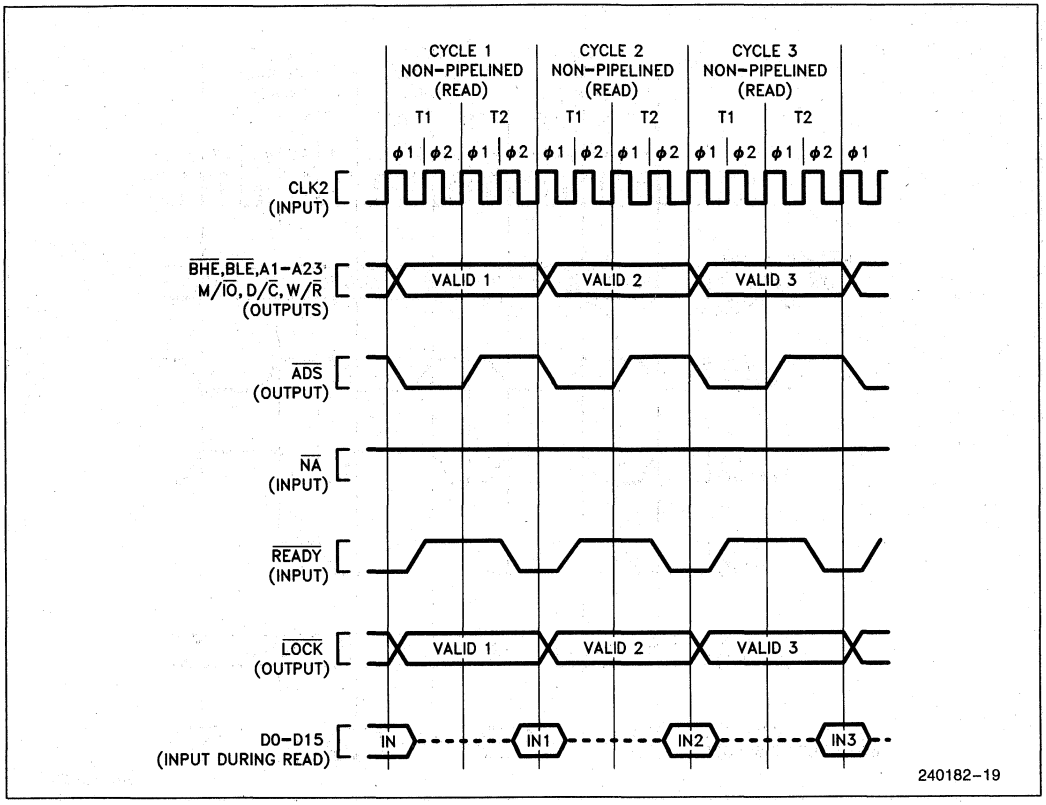


Figure 4.4. Fastest Read Cycles with Non-Pipelined Timing

The fastest 80376 bus cycle requires only two bus states. For example, three consecutive bus read cycles, each consisting of two bus states, are shown by Figure 4.4. The bus states in each cycle are named T1 and T2. Any memory or I/O address may be accessed by such a two-state bus cycle, if the external hardware is fast enough.

Every bus cycle continues until it is acknowledged by the external system hardware, using the 80376 $\overline{\text{READY}}$ input. Acknowledging the bus cycle at the end of the first T2 results in the shortest bus cycle, requiring only T1 and T2. If $\overline{\text{READY}}$ is not immediately asserted however, T2 states are repeated indefinitely until the $\overline{\text{READY}}$ input is sampled active.

The pipelining option provides a choice of bus cycle timings. Pipelined or non-pipelined cycles are selectable on a cycle-by-cycle basis with the Next Address ($\overline{\text{NA}}$) input.

When pipelining is selected the address ($\overline{\text{BHE}}$, $\overline{\text{BLE}}$ and A_{23}–A_1) and definition ($W/\overline{R}$, $D/\overline{C}$, $M/\overline{IO}$ and $\overline{\text{LOCK}}$) of the next cycle are available before the end of the current cycle. To signal their availability, the 80376 address status output ($\overline{\text{ADS}}$) is asserted. Figure 4.5 illustrates the fastest read cycles with pipelined timing.

Note from Figure 4.5 the fastest bus cycles using pipelining require only two bus states, named **T1P** and **T2P**. Therefore pipelined cycles allow the same data bandwidth as non-pipelined cycles, but address-to-data access time is increased by one T-state time compared to that of a non-pipelined cycle.

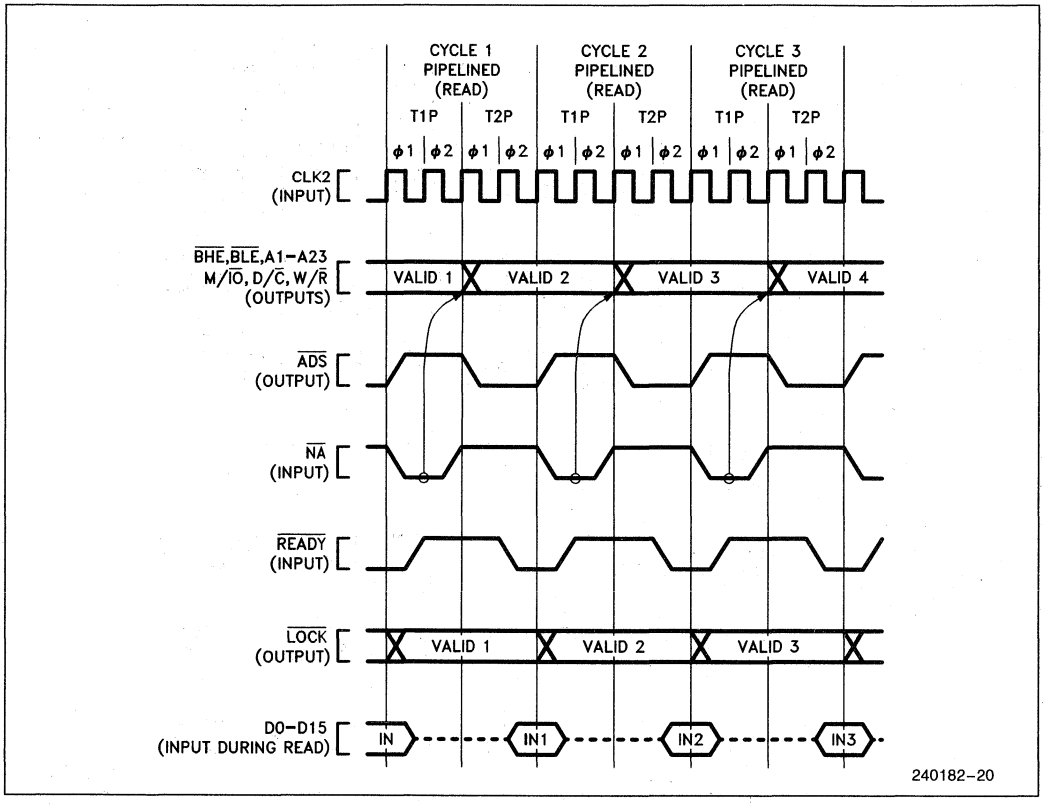


Figure 4.5. Fastest Read Cycles with Pipelined Timing

READ AND WRITE CYCLES

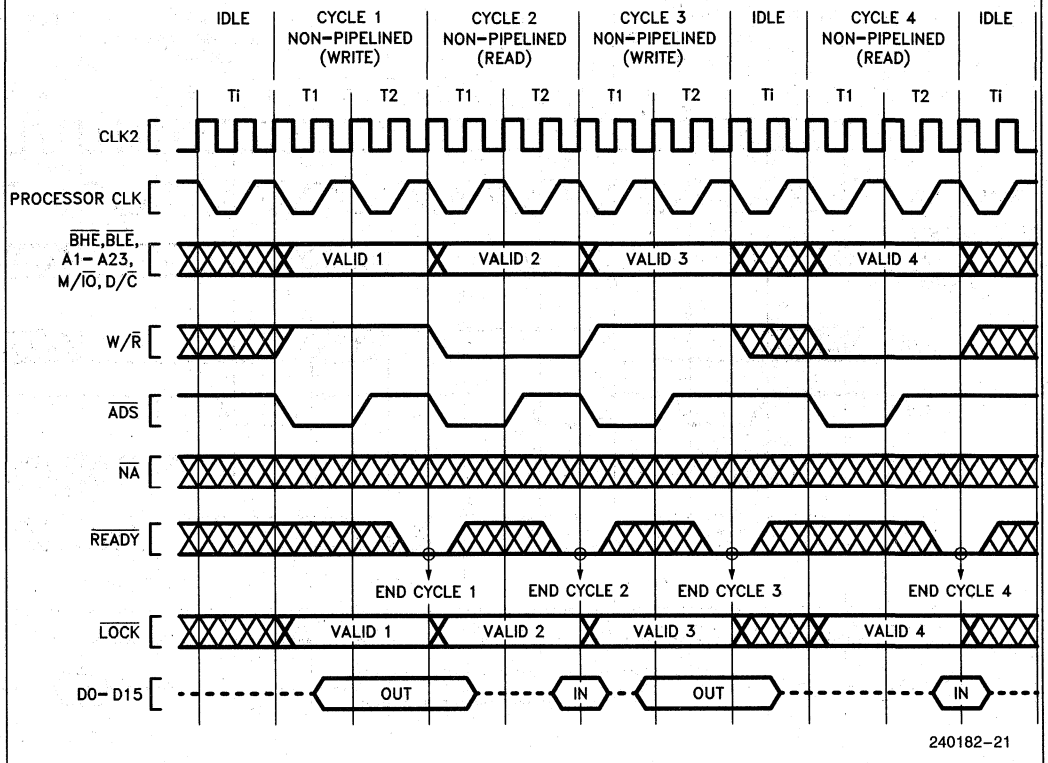
Data transfers occur as a result of bus cycles, classified as read or write cycles. During read cycles, data is transferred from an external device to the processor. During write cycles, data is transferred from the processor to an external device.

Two choices of bus cycle timing are dynamically selectable: non-pipelined or pipelined. After an idle bus state, the processor always uses non-pipelined timing. However the $\overline{NA}$ (Next Address) input may be asserted to select pipelined timing for the next bus cycle. When pipelining is selected and the 80376 has a bus request pending internally, the address and definition of the next cycle is made available even before the current bus cycle is acknowledged by $\overline{READY}$.

Terminating a read or write cycle, like any bus cycle, requires acknowledging the cycle by asserting the $\overline{READY}$ input. Until acknowledged, the processor inserts wait states into the bus cycle, to allow adjustment for the speed of any external device. External hardware, which has decoded the address and bus cycle type, asserts the $\overline{READY}$ input at the appropriate time.

At the end of the second bus state within the bus cycle, $\overline{READY}$ is sampled. At that time, if external hardware acknowledges the bus cycle by asserting $\overline{READY}$, the bus cycle terminates as shown in Figure 4.6. If $\overline{READY}$ is negated as in Figure 4.7, the 80376 executes another bus state (a wait state) and $\overline{READY}$ is sampled again at the end of that state. This continues indefinitely until the cycle is acknowledged by $\overline{READY}$ asserted.

When the current cycle is acknowledged, the 80376 terminates it. When a read cycle is acknowledged, the 80376 latches the information present at its data pins. When a write cycle is acknowledged, the write data of the 80376 remains valid throughout phase one of the next bus state, to provide write data hold time.



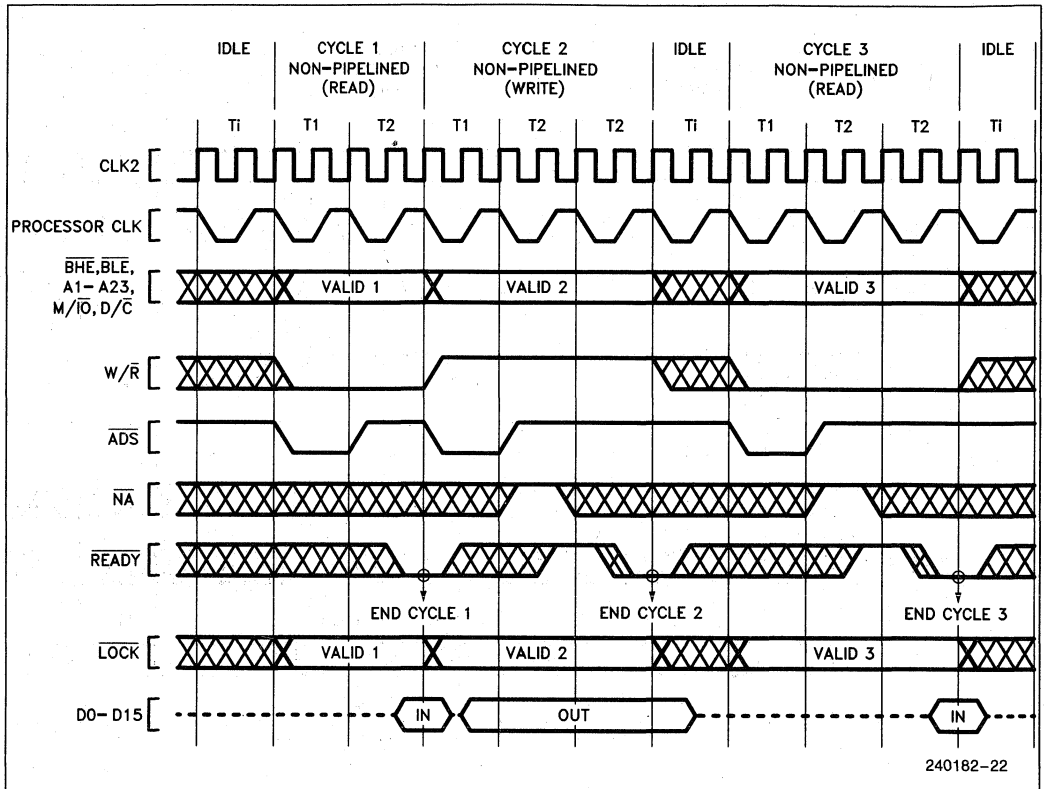
Idle states are shown here for diagram variety only. Write cycles are not always followed by an idle state. An active bus cycle can immediately follow the write cycle.

Figure 4.6. Various Non-Pipelined Bus Cycles (Zero Wait States)

Non-Pipelined Bus Cycles

Any bus cycle may be performed with non-pipelined timing. For example, Figure 4.6 shows a mixture of non-pipelined read and write cycles. Figure 4.6 shows that the fastest possible non-pipelined cycles have two bus states per bus cycle. The states are named T1 and T2. In phase one of T1, the address signals and bus cycle definition signals are driven valid and, to signal their availability, address strobe ($\overline{ADS}$) is simultaneously asserted.

During read or write cycles, the data bus behaves as follows. If the cycle is a read, the 80376 floats its data signals to allow driving by the external device being addressed. **The 80376 requires that all data bus pins be at a valid logic state (HIGH or LOW) at the end of each read cycle, when $\overline{READY}$ is asserted. The system MUST be designed to meet this requirement.** If the cycle is a write, data signals are driven by the 80376 beginning in phase two of T1 until phase one of the bus state following cycle acknowledgement.



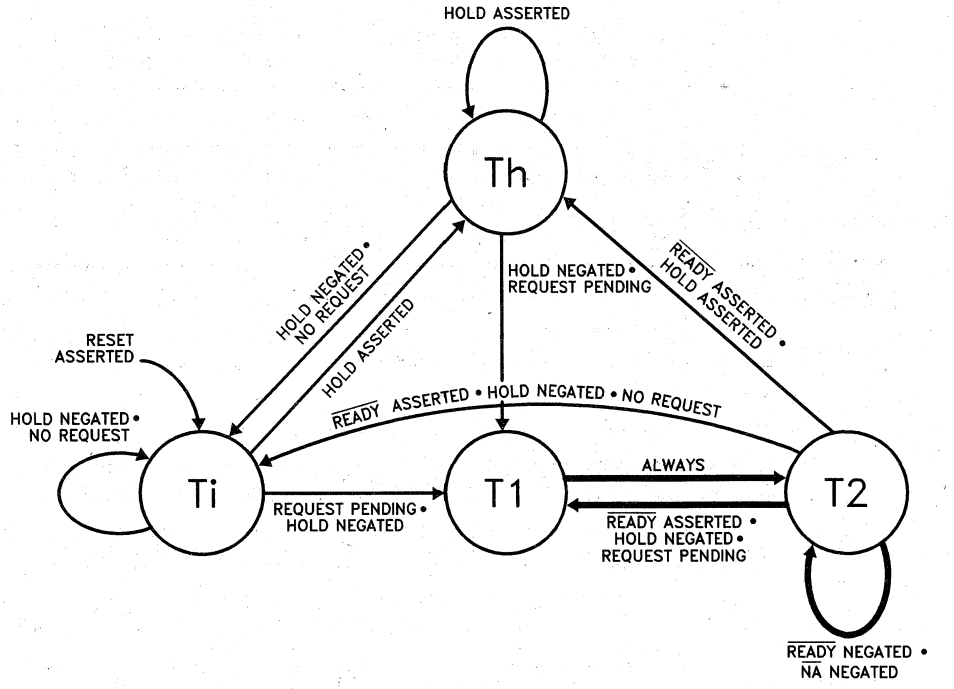
Idle states are shown here for diagram variety only. Write cycles are not always followed by an idle state. An active bus cycle can immediately follow the write cycle.

Figure 4.7. Various Non-Pipelined Bus Cycles (Various Number of Wait States)

Figure 4.7 illustrates non-pipelined bus cycles with one wait state added to Cycles 2 and 3. $\overline{\text{READY}}$ is sampled inactive at the end of the first T2 in Cycles 2 and 3. Therefore Cycles 2 and 3 have T2 repeated again. At the end of the second T2, $\overline{\text{READY}}$ is sampled active.

When address pipelining is not used, the address and bus cycle definition remain valid during all wait states. When wait states are added and it is desirable to maintain non-pipelined timing, it is necessary to negate $\overline{\text{NA}}$ during each T2 state except the last one, as shown in Figure 4.7, Cycles 2 and 3. If $\overline{\text{NA}}$ is sampled active during a T2 other than the last one, the next state would be T2I or T2P instead of another T2.

When address pipelining is not used, the bus states and transitions are completely illustrated by Figure 4.8. The bus transitions between four possible states, T1, T2, T_i, and T_h. Bus cycles consist of T1 and T2, with T2 being repeated for wait states. Otherwise the bus may be idle, T_i, or in the hold acknowledge state T_h.



Bus States:
T1—first clock of a non-pipelined bus cycle (80376 drives new address and asserts $\overline{ADS}$).
T2—subsequent clocks of a bus cycle when $\overline{NA}$ has not been sampled asserted in the current bus cycle.
Ti—idle state.
Th—hold acknowledge state (80376 asserts HLDA).
The fastest bus cycle consists of two states: T1 and T2.
Four basic bus states describe bus operation when not using pipelined address.

Figure 4.8. 80376 Bus States (Not Using Pipelined Address)

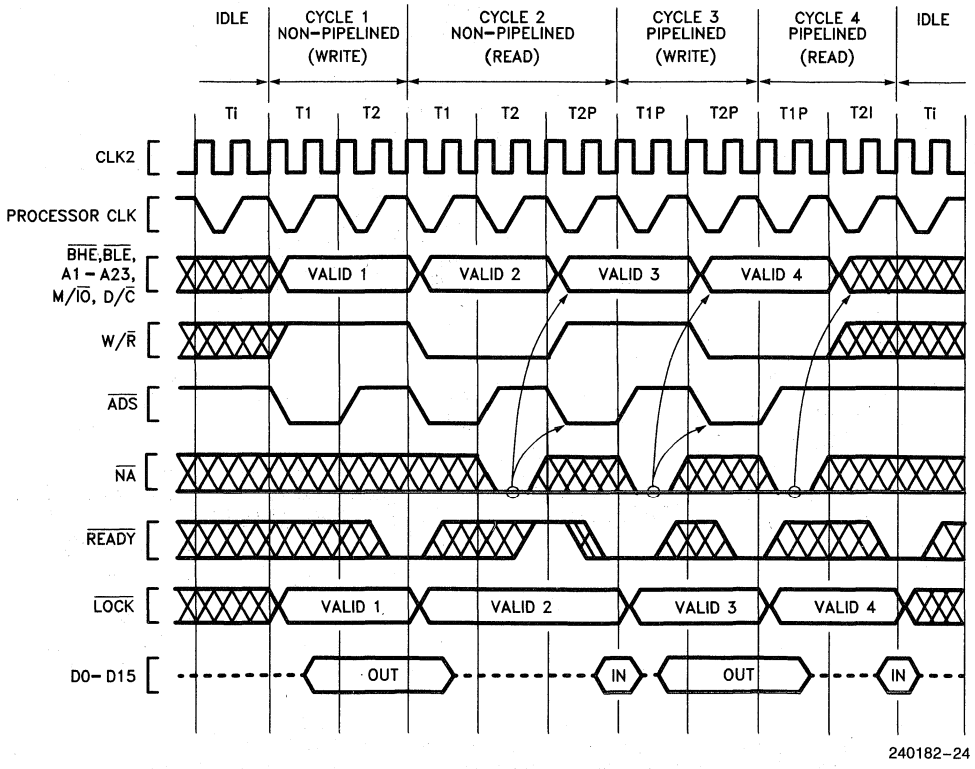
Bus cycles always begin with T1. T1 always leads to T2. If a bus cycle is not acknowledged during T2 and $\overline{NA}$ is inactive, T2 is repeated. When a cycle is acknowledged during T2, the following state will be T1 of the next bus cycle if a bus request is pending internally, or T_i if there is no bus request pending, or T_h if the HOLD input is being asserted.

Use of pipelining allows the 80376 to enter three additional bus states not shown in Figure 4.8. Figure 4.12 is the complete bus state diagram, including pipelined cycles.

Pipelined Bus Cycles

Pipelining is the option of requesting the address and the bus cycle definition of the next internally pending bus cycle before the current bus cycle is acknowledged with $\overline{READY}$ asserted. $\overline{ADS}$ is asserted by the 80376 when the next address is issued. The pipelining option is controlled on a cycle-by-cycle basis with the $\overline{NA}$ input signal.

Once a bus cycle is in progress and the current address has been valid for at least one entire bus state, the $\overline{NA}$ input is sampled at the end of every phase one until the bus cycle is acknowledged. During non-pipelined bus cycles $\overline{NA}$ is sampled at the end of phase one in every T2. An example is Cycle 2 in Figure 4.9, during which $\overline{NA}$ is sampled at the end of phase one of every T2 (it was asserted once during the first T2 and has no further effect during that bus cycle).



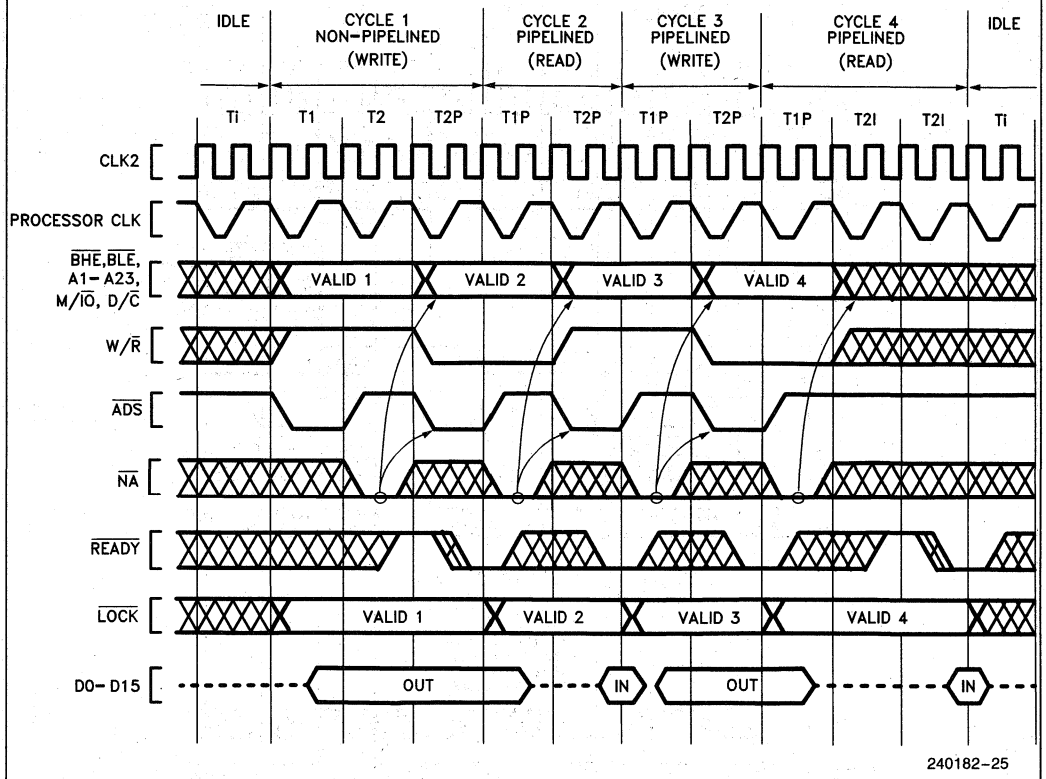
Following any idle bus state (Ti), bus cycles are non-pipelined. Within non-pipelined bus cycles, $\overline{NA}$ is only sampled during wait states. Therefore, to begin pipelining during a group of non-pipelined bus cycles requires a non-pipelined cycle with at least one wait state (Cylcle 2 above).

Figure 4.9. Transitioning to Pipelining during Burst of Bus Cycles

If $\overline{NA}$ is sampled active, the 80376 is free to drive the address and bus cycle definition of the next bus cycle, and assert $\overline{ADS}$, as soon as it has a bus request internally pending. It may drive the next address as early as the next bus state, whether the current bus cycle is acknowledged at that time or not.

Regarding the details of pipelining, the 80376 has the following characteristics:

1. The next address and status may appear as early as the bus state after $\overline{NA}$ was sampled active (see Figures 4.9 or 4.10). In that case, state T2P is entered immediately. However, when there is not an internal bus request already pending, the next address and status will not be available immediately after $\overline{NA}$ is asserted and T2I is entered instead of T2P (see Figure 4.11 Cycle 3). Provided the current bus cycle isn't yet acknowledged by $\overline{READY}$ asserted, T2P will be entered as soon as the 80376 does drive the next address and status. External hardware should therefore observe the $\overline{ADS}$ output as confirmation the next address and status are actually being driven on the bus.
2. Any address and status which are validated by a pulse on the 80376 $\overline{ADS}$ output will remain stable on the address pins for at least two processor clock periods. The 80376 cannot produce a new address and status more frequently than every two processor clock periods (see Figures 4.9, 4.10 and 4.11).
3. Only the address and bus cycle definition of the very next bus cycle is available. The pipelining capability cannot look further than one bus cycle ahead (see Figure 4.11, Cycle 1).



Following any idle bus state (Ti) the bus cycle is always non-pipelined and $\overline{NA}$ is only sampled during wait states. To start, address pipelining after an idle state requires a non-pipelined cycle with at least one wait state (cycle 1 above). The pipelined cycles (2, 3, 4 above) are shown with various numbers of wait states.

Figure 4.10. Fastest Transition to Pipelined Bus Cycle Following Idle Bus State

The complete bus state transition diagram, including pipelining is given by Figure 4.12. Note it is a superset of the diagram for non-pipelined only, and the three additional bus states for pipelining are drawn in bold.

The fastest bus cycle with pipelining consists of just two bus states, T1P and T2P (recall for non-pipelined it is T1 and T2). T1P is the first bus state of a pipelined cycle.

Initiating and Maintaining Pipelined Bus Cycles

Using the state diagram Figure 4.12, observe the transitions from an idle state, T_i, to the beginning of a pipelined bus cycle T1P. From an idle state, T_i, the first bus cycle must begin with T1, and is therefore a non-pipelined bus cycle. The next bus cycle will be pipelined, however, provided $\overline{NA}$ is asserted and the first bus cycle ends in a T2P state (the address and status for the next bus cycle is driven during T2P). The fastest path from an idle state to a pipelined bus cycle is shown in bold below:

T_i, T_i,	T1-T2-T2P,	T1P-T2P,
idle states	non-pipelined cycle	pipelined cycle

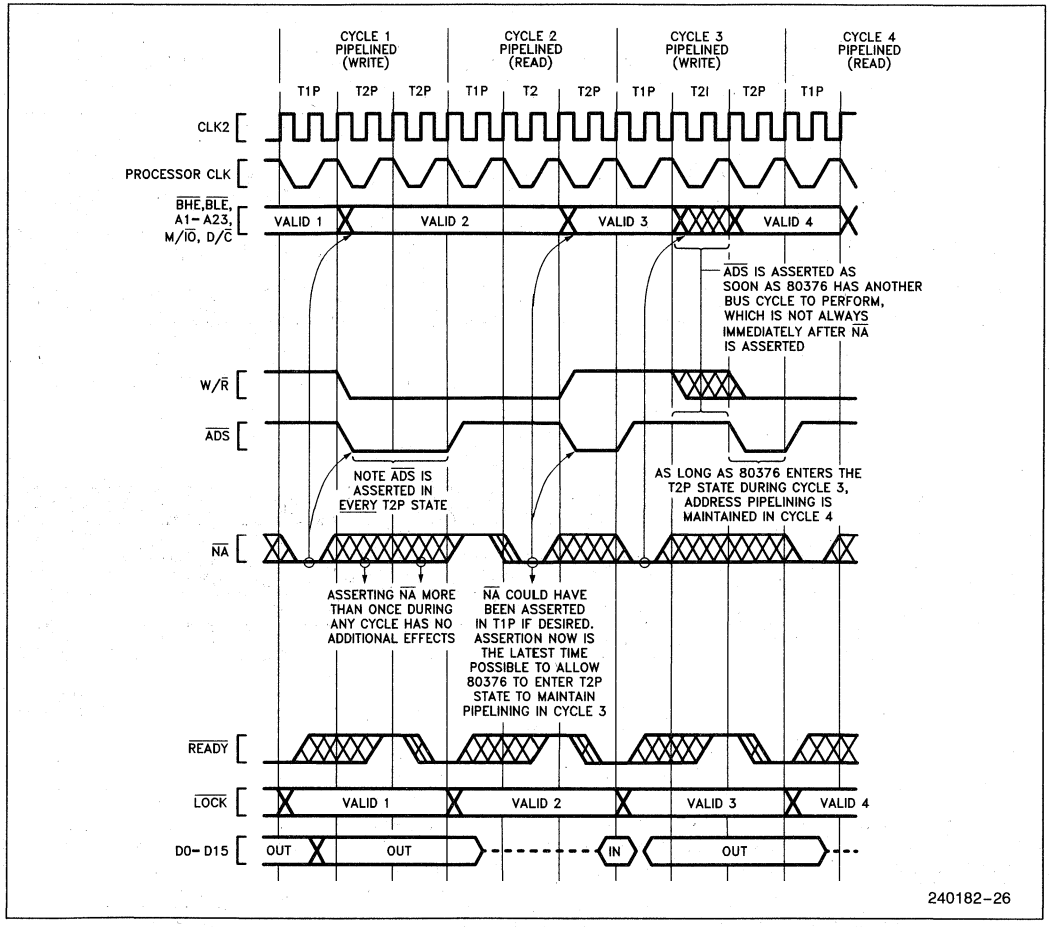


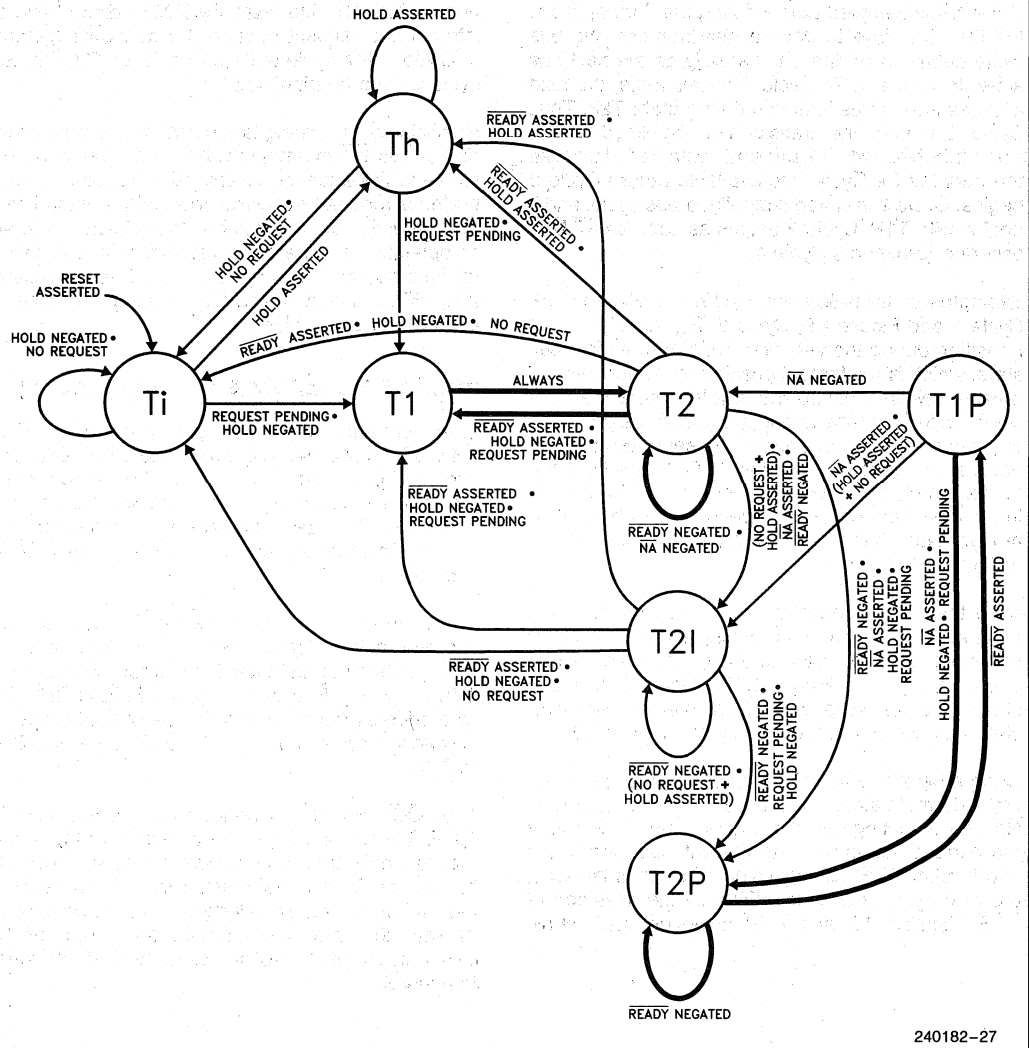
Figure 4.11. Details of Address Pipelining during Cycles with Wait States

T1–T2–T2P are the states of the bus cycle that establishes address pipelining for the next bus cycle, which begins with T1P. The same is true after a bus hold state, shown below:

T_h, T_h, T_h,	T1–T2–T2P,	T1P–T2P,
hold aknowledge states	non-pipelined cycle	pipelined cycle

The transition to pipelined address is shown functionally by Figure 4.10, Cycle 1. Note that Cycle 1 is used to transition into pipelined address timing for the subsequent Cycles 2, 3 and 4, which are pipelined. The $\overline{NA}$ input is asserted at the appropriate time to select address pipelining for Cycles 2, 3 and 4.

Once a bus cycle is in progress and the current address and status has been valid for one entire bus state, the $\overline{NA}$ input is sampled at the end of every phase one until the bus cycle is acknowledged.



Bus States:
T1—first clock of a non-pipelined bus cycle (80376 drives new address, status and asserts $\overline{ADS}$).
T2—subsequent clocks of a bus cycle when $\overline{NA}$ has not been sampled asserted in the current bus cycle.
T2I—subsequent clocks of a bus cycle when $\overline{NA}$ has been sampled asserted in the current bus cycle but there is not yet an internal bus request pending (80376 will not drive new address, status or assert $\overline{ADS}$).
T2P—subsequent clocks of a bus cycle when $\overline{NA}$ has been sampled asserted in the current bus cycle and there is an internal bus request pending (80376 drives new address, status and asserts $\overline{ADS}$).
T1P—first clock of a pipelined bus cycle.
Ti—idle state.
Th—hold acknowledge state (80376 asserts HLDA).
Asserting $\overline{NA}$ for pipelined bus cycles gives access to three more bus states: T2I, T2P and T1P.
Using pipelining the fastest bus cycle consists of T1P and T2P.

Figure 4.12. 80376 Processor Complete Bus States (Including Pipelining)

Sampling begins in T2 during Cycle 1 in Figure 4.10. Once $\overline{NA}$ is sampled active during the current cycle, the 80376 is free to drive a new address and bus cycle definition on the bus as early as the next bus state. In Figure 4.10, Cycle 1 for example, the next address and status is driven during state T2P. Thus Cycle 1 makes the transition to pipelined timing, since it begins with T1 but ends with T2P. Because the address for Cycle 2 is available before Cycle 2 begins, Cycle 2 is called a pipelined bus cycle, and it begins with T1P. Cycle 2 begins as soon as $\overline{READY}$ asserted terminates Cycle 1.

Examples of transition bus cycles are Figure 4.10, Cycle 1 and Figure 4.9, Cycle 2. Figure 4.10 shows transition during the very first cycle after an idle bus state, which is the fastest possible transition into address pipelining. Figure 4.9, Cycle 2 shows a transition cycle occurring during a burst of bus cycles. In any case, a transition cycle is the same whenever it occurs: it consists at least of T1, T2 ($\overline{NA}$ is asserted at that time), and T2P (provided the 80376 has an internal bus request already pending, which it almost always has). T2P states are repeated if wait states are added to the cycle.

Note that only three states (T1, T2 and T2P) are required in a bus cycle performing a **transition** from non-pipelined into pipelined timing, for example Figure 4.10, Cycle 1. Figure 4.10, Cycles 2, 3 and 4 show that pipelining can be maintained with two-state bus cycles consisting only of T1P and T2P.

Once a pipelined bus cycle is in progress, pipelined timing is maintained for the next cycle by asserting $\overline{NA}$ and detecting that the 80376 enters T2P during the current bus cycle. The current bus cycle must end in state T2P for pipelining to be maintained in the next cycle. T2P is identified by the assertion of $\overline{ADS}$. Figures 4.9 and 4.10 however, each show pipelining ending after Cycle 4 because Cycle 4 ends in T2I. This indicates the 80376 didn't have an internal bus request prior to the acknowledgement of Cycle 4. If a cycle ends with a T2 or T2I, the next cycle will not be pipelined.

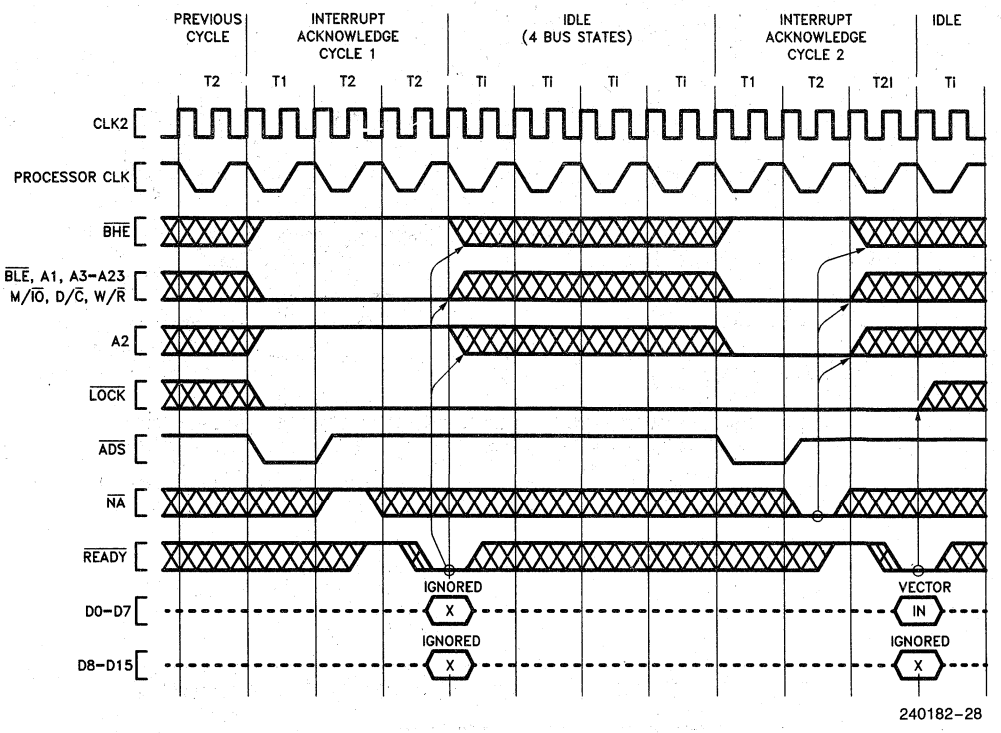
Realistically, pipelining is almost always maintained as long as $\overline{NA}$ is sampled asserted. This is so because in the absence of any other request, a code prefetch request is always internally pending until the instruction decoder and code prefetch queue are completely full. Therefore pipelining is maintained for long bursts of bus cycles, if the bus is available (i.e., HOLD inactive) and $\overline{NA}$ is sampled active in each of the bus cycles.

INTERRUPT ACKNOWLEDGE (INTA) CYCLES

In repsonse to an interrupt request on the INTR input when interrupts are enabled, the 80376 performs two interrupt acknowledge cycles. These bus cycles are similar to read cycles in that bus definition signals define the type of bus activity taking place, and each cycle continues until acknowledged by $\overline{READY}$ sampled active.

The state of A_2 distinguishes the first and second interrupt acknowledge cycles. The byte address driven during the first interrupt acknowledge cycle is 4 (A_{23}–A_3, A_1, $\overline{BLE}$ LOW, A_2 and $\overline{BHE}$ HIGH). The byte address driven during the second interrupt acknowledge cycle is 0 (A_{23}–A_1, $\overline{BLE}$ LOW and $\overline{BHE}$ HIGH).

The $\overline{LOCK}$ output is asserted from the beginning of the first interrupt acknowledge cycle until the end of the second interrupt acknowledge cycle. Four idle bus states, T_i, are inserted by the 80376 between the two interrupt acknowledge cycles for compatibility with the interrupt specification T_{RHRL} of the 8259A Interrupt Controller and the 82370 Integrated Peripheral.



Interrupt Vector (0–255) is read on D0–D7 at end of second Interrupt Acknowledge bus cycle.
Because each Interrupt Acknowledge bus cycle is followed by idle bus states, asserting $\overline{NA}$ has no practical effect. Choose the approach which is simplest for your system hardware design.

Figure 4.13. Interrupt Acknowledge Cycles

During both interrupt acknowledge cycles, D_{15}–D_0 float. No data is read at the end of the first interrupt acknowledge cycle. At the end of the second interrupt acknowledge cycle, the 80376 will read an external interrupt vector from D_7–D_0 of the data bus. The vector indicates the specific interrupt number (from 0–255) requiring service.

HALT INDICATION CYCLE

The 80376 execution unit halts as a result of executing a HLT instruction. Signaling its entrance into the halt state, a halt indication cycle is performed. The halt indication cycle is identified by the state of the bus definition signals and a byte address of 2. See the **Bus Cycle Definition Signals** section. The halt indication cycle must be acknowledged by $\overline{READY}$ asserted. A halted 80376 resumes execution when INTR (if interrupts are enabled), NMI or RESET is asserted.

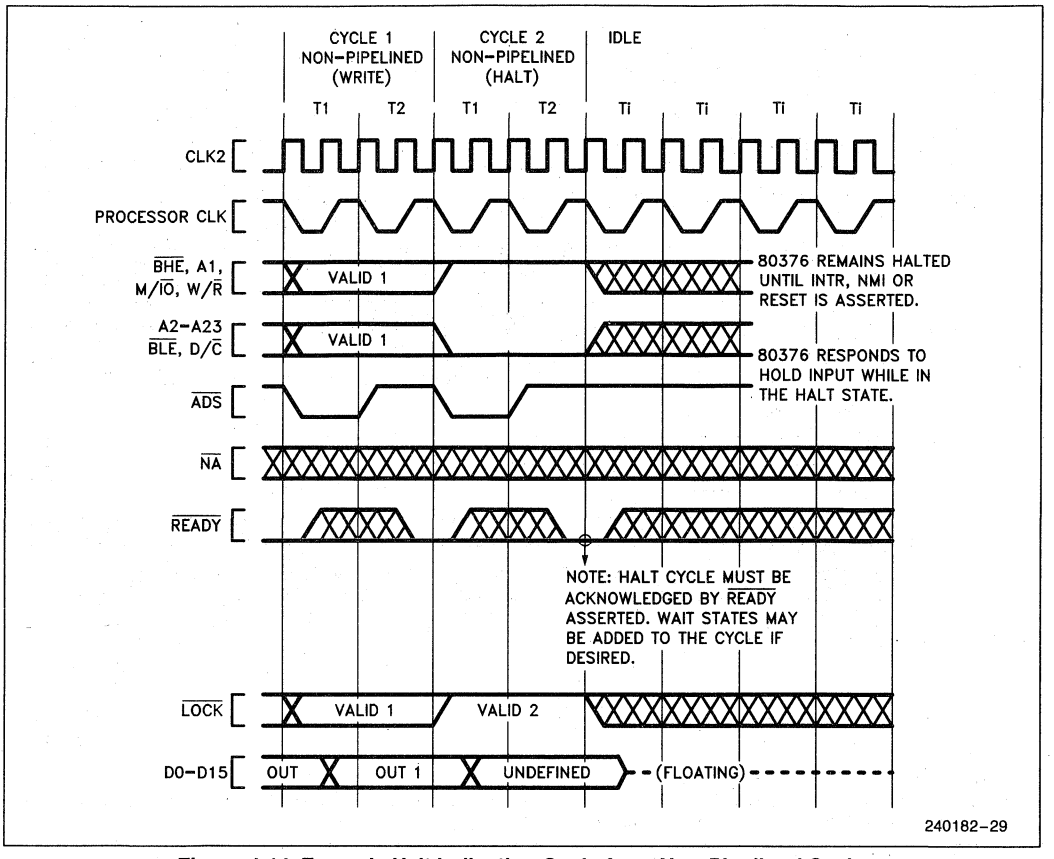


Figure 4.14. Example Halt Indication Cycle from Non-Pipelined Cycle

SHUTDOWN INDICATION CYCLE

The 80376 shuts down as a result of a protection fault while attempting to process a double fault. Signaling its entrance into the shutdown state, a shutdown indication cycle is performed. The shutdown indication cycle is identified by the state of the bus definition signals shown in **Bus Cycle Definition Signals** and a byte address of 0. The shutdown indication cycle must be acknowledged by $\overline{READY}$ asserted. A shutdown 80376 resumes execution when NMI or RESET is asserted.

ENTERING AND EXITING HOLD ACKNOWLEDGE

The bus hold acknowledge state, T_h, is entered in response to the HOLD input being asserted. In the bus hold acknowledge state, the 80376 floats all outputs or bidirectional signals, except for HLDA. HLDA is asserted as long as the 80376 remains in the bus hold acknowledge state. In the bus hold acknowledge state, all inputs except HOLD and RESET are ignored.

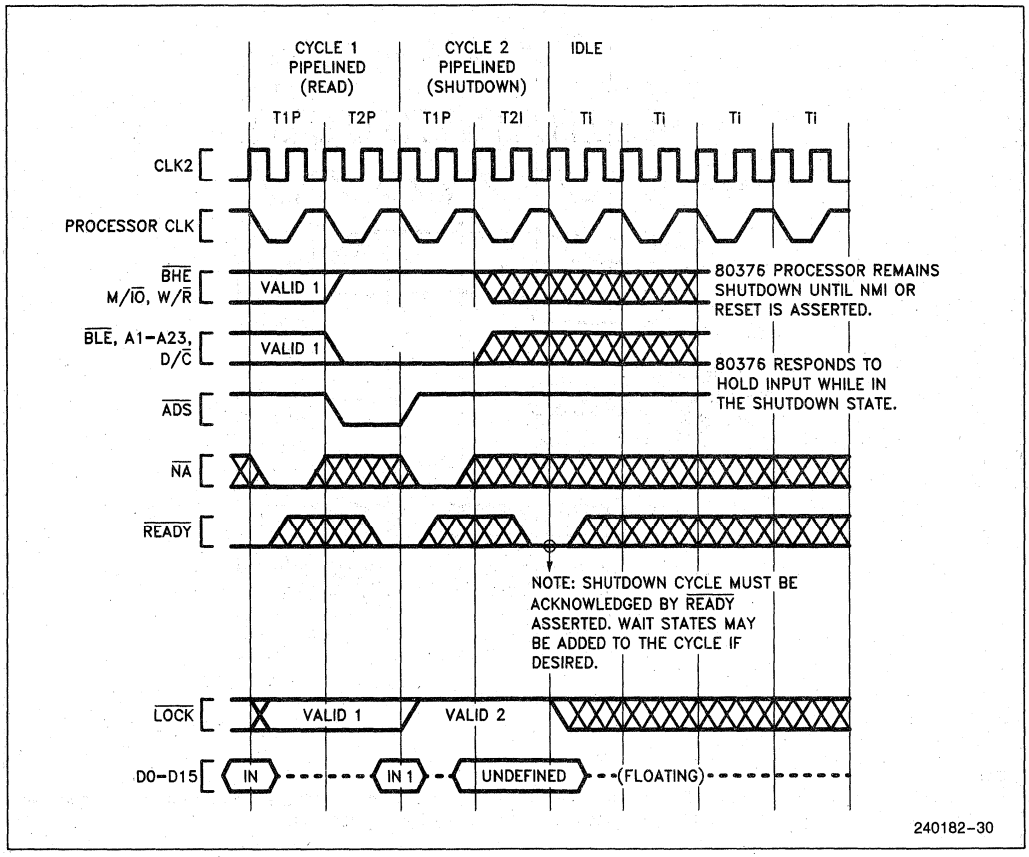
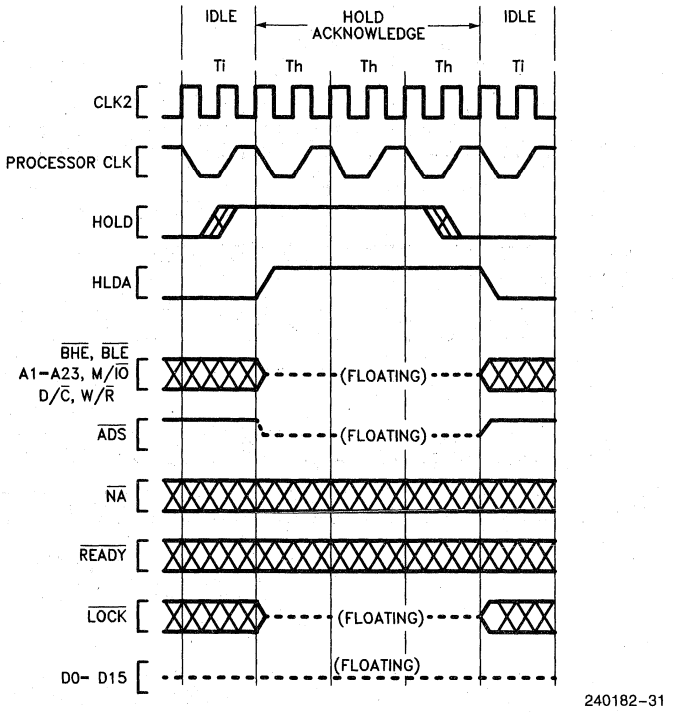


Figure 4.15. Example Shutdown Indication Cycle from Non-Pipelined Cycle

T_h may be entered from a bus idle state as in Figure 4.16 or after the acknowledgement of the current physical bus cycle if the LOCK signal is not asserted, as in Figures 4.17 and 4.18.

T_h is exited in response to the HOLD input being negated. The following state will be T_i as in Figure 4.16 if no bus request is pending. The following bus state will be T1 if a bus request is internally pending, as in Figures 4.17 and 4.18. T_h is exited in response to RESET being asserted.

If a rising edge occurs on the edge-triggered NMI input while in T_h, the event is remembered as a non-maskable interrupt 2 and is serviced when T_h is exited unless the 80376 is reset before T_h is exited.



NOTE:
For maximum design flexibility the 80376 has no internal pull-up resistors on its outputs. Your design may require an external pullup on $\overline{ADS}$ and other 80376 outputs to keep them negated during float periods.

Figure 4.16. Requesting Hold from Idle Bus

RESET DURING HOLD ACKNOWLEDGE

RESET being asserted takes priority over HOLD being asserted. If RESET is asserted while HOLD remains asserted, the 80376 drives its pins to defined states during reset, as in **Table 4.5, Pin State During Reset,** and performs internal reset activity as usual.

If HOLD remains asserted when RESET is inactive, the 80376 enters the hold acknowledge state before performing its first bus cycle, provided HOLD is still asserted when the 80376 processor would otherwise perform its first bus cycle. If HOLD remains asserted when RESET is inactive, the $\overline{BUSY}$ input is still sampled as usual to determine whether a self test is being requested.

FLOAT

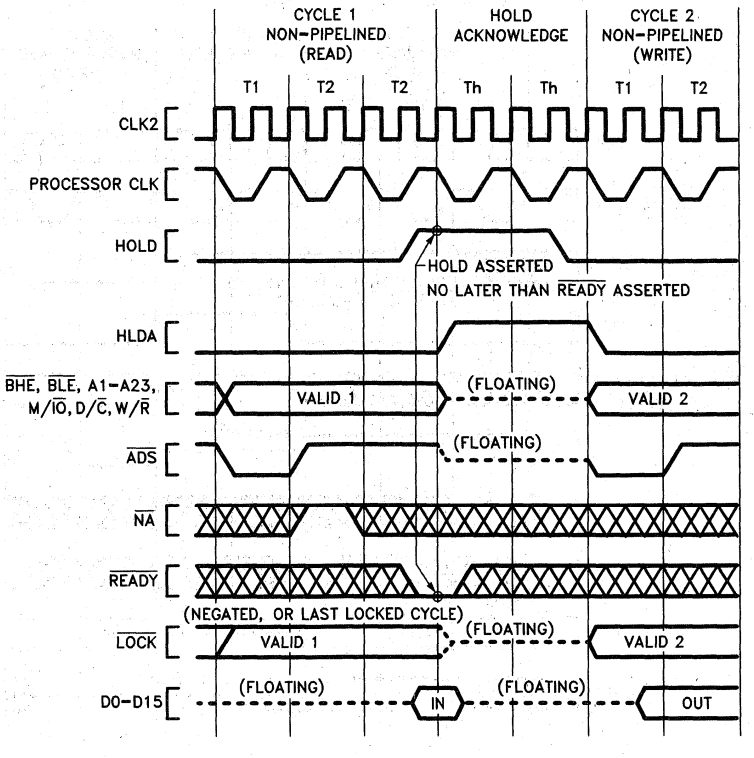
Activating the $\overline{FLT}$ input floats all 80376 bidirectional and output signals, including HLDA. Asserting $\overline{FLT}$ isolates the 80376 from the surrounding circuitry.

When an 80376 in a PQFP surface-mount package is used without a socket, it cannot be removed from the printed circuit board. The $\overline{FLT}$ input allows the 80376 to be electrically isolated to allow testing of external circuitry. This technique is known as ON-CE™ for "ON-Circuit Emulation".

ENTERING AND EXITING FLOAT

$\overline{FLT}$ is an asynchronous, active-low input. It is recognized on the rising edge of CLK2. When recognized, it aborts the current bus cycle and floats the outputs of the 80376 (Figure 4.20). $\overline{FLT}$ must be held low for a minimum of 16 CLK2 cycles. Reset should be asserted and held asserted until after $\overline{FLT}$ is deasserted. This will ensure that the 80376 will exit float in a valid state.

Asserting the $\overline{FLT}$ input unconditionally aborts the current bus cycle and forces the 80376 into the FLOAT mode. Since activating $\overline{FLT}$ unconditionally forces the 80376 into FLOAT mode, the 80376 is not



NOTE:
HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold (t_{23} and t_{24}) requirements are met. This waveform is useful for determining Hold Acknowledge latency.

Figure 4.17. Requesting Hold from Active Bus ($\overline{NA}$ Inactive)

guaranteed to enter FLOAT in a valid state. After deactivating $\overline{FLT}$, the 80376 is not guaranteed to exit FLOAT mode in a valid state. This is not a problem as the $\overline{FLT}$ pin is meant to be used only during ONCE. After exiting FLOAT, the 80376 must be reset to return it to a valid state. Reset should be asserted before $\overline{FLT}$ is deasserted. This will ensure that the 80376 will exit float in a valid state.

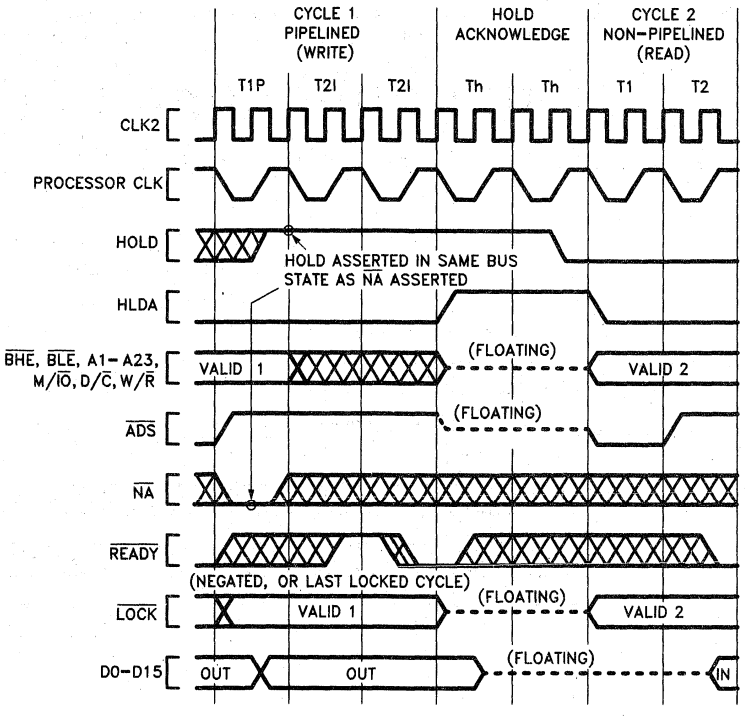
$\overline{FLT}$ has an internal pull-up resistor, and if it is not used it should be unconnected.

BUS ACTIVITY DURING AND FOLLOWING RESET

RESET is the highest priority input signal, capable of interrupting any processor activity when it is asserted. A bus cycle in progress can be aborted at any stage, or idle states or bus hold acknowledge states discontinued so that the reset state is established.

RESET should remain asserted for at least 15 CLK2 periods to ensure it is recognized throughout the 80376, and at least 80 CLK2 periods if a 80376 self-test is going to be requested at the falling edge. RESET asserted pulses less than 15 CLK2 periods may not be recognized. RESET pulses less than 80 CLK2 periods followed by a self-test may cause the self-test to report a failure when no true failure exists.

Provided the RESET falling edge meets setup and hold times t_{25} and t_{26}, the internal processor clock phase is defined at that time as illustrated by Figure 4.19 and Figure 6.7.

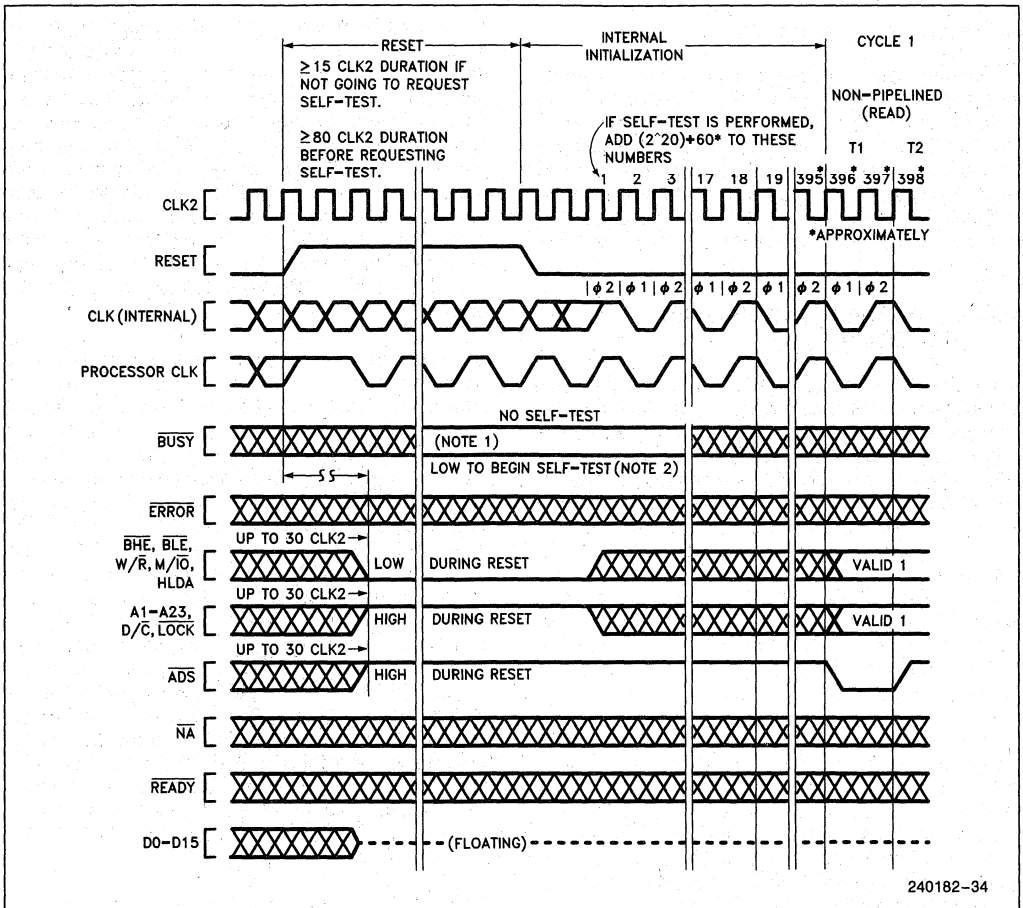


NOTE:
HOLD is a synchronous input and can be asserted at any CLK2 edge, provided setup and hold (t_{23} and t_{24}) requirements are met. This waveform is useful for determining Hold Acknowledge latency.

Figure 4.18. Requesting Hold from Idle Bus ($\overline{NA}$ Active)

An 80376 self-test may be requested at the time RESET goes inactive by having the $\overline{BUSY}$ input at a LOW level as shown in Figure 4.19. The self-test requires (2^{20} + approximately 60) CLK2 periods to complete. The self-test duration is not affected by the test results. Even if the self-test indicates a problem, the 80376 attempts to proceed with the reset sequence afterwards.

After the RESET falling edge (and after the self-test if it was requested) the 80376 performs an internal initialization sequence for approximately 350 to 450 CLK2 periods.



NOTES:
1. BUSY should be held stable for 8 CLK2 periods before and after the CLK2 period in which RESET falling edge occurs.
2. If self-test is requested, the 80376 outputs remain in their reset state as shown here.

Figure 4.19. Bus Activity from Reset until First Code Fetch

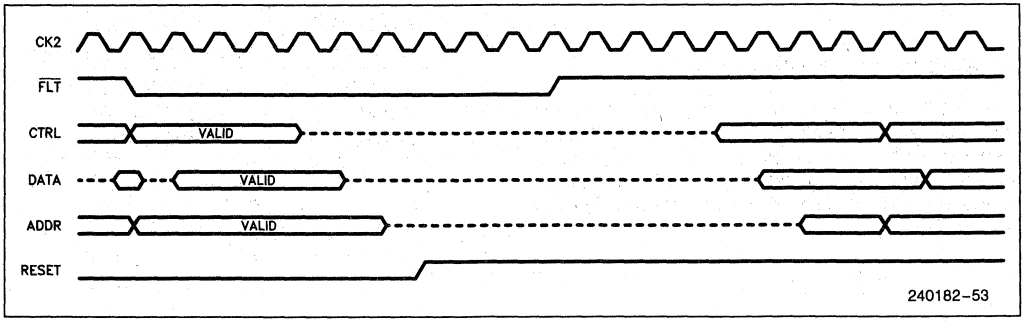


Figure 4.20. Entering and Exiting FLOAT

4.5 Self-Test Signature

Upon completion of self-test (if self-test was requested by driving $\overline{BUSY}$ LOW at the falling edge of RESET) the EAX register will contain a signature of 00000000H indicating the 80376 passed its self-test of microcode and major PLA contents with no problems detected. The passing signature in EAX, 00000000H, applies to all 80376 revision levels. Any non-zero signature indicates the 80376 unit is faulty.

4.6 Component and Revision Identifiers

To assist 80376 users, the 80376 after reset holds a component identifier and revision identifier in its DX register. The upper 8 bits of DX hold 33H as identification of the 80376 component. (The lower nibble, 03H, refers to the Intel386™ architecture. The upper nibble, 30H, refers to the third member of the Intel386 family). The lower 8 bits of DX hold an 8-bit unsigned binary number related to the component revision level. The revision identifier will, in general, chronologically track those component steppings which are intended to have certain improvements or distinction from previous steppings. The 80376 revision identifier will track that of the 80386 where possible.

The revision identifier is intended to assist 80376 users to a practical extent. However, the revision identifier value is not guaranteed to change with every stepping revision, or to follow a completely uniform numerical sequence, depending on the type or intention of revision, or manufacturing materials required to be changed. Intel has sole discretion over these characteristics of the component.

Table 4.7. Component and Revision Identifier History

80376 Stepping Name	Revision Identifier
A0	05H
B	08H

4.7 Coprocessor Interfacing

The 80376 provides an automatic interface for the Intel 80387SX numeric floating-point coprocessor. The 80387SX coprocessor uses an I/O mapped interface driven automatically by the 80376 and assisted by three dedicated signals: $\overline{BUSY}$, $\overline{ERROR}$ and PEREQ.

As the 80376 begins supporting a coprocessor instruction, it tests the $\overline{BUSY}$ and $\overline{ERROR}$ signals to determine if the coprocessor can accept its next instruction. Thus, the $\overline{BUSY}$ and $\overline{ERROR}$ inputs eliminate the need for any "preamble" bus cycles for communication between processor and coprocessor. The 80387SX can be given its command opcode immediately. The dedicated signals provide instruction synchronization, and eliminate the need of using the 80376 WAIT opcode (9BH) for 80387SX instruction synchronization (the WAIT opcode was required when the 8086 or 8088 was used with the 8087 coprocessor).

Custom coprocessors can be included in 80376 based systems by memory-mapped or I/O-mapped interfaces. Such coprocessor interfaces allow a completely custom protocol, and are not limited to a set of coprocessor protocol "primitives". Instead, memory-mapped or I/O-mapped interfaces may use all applicable 80376 instructions for high-speed coprocessor communication. The $\overline{BUSY}$ and $\overline{ERROR}$ inputs of the 80376 may also be used for the custom coprocessor interface, if such hardware assist is desired. These signals can be tested by the 80376 WAIT opcode (9BH). The WAIT instruction will wait until the $\overline{BUSY}$ input is inactive (interruptable by an NMI or enabled INTR input), but generates an exception 16 fault if the $\overline{ERROR}$ pin is active when the $\overline{BUSY}$ goes (or is) inactive. If the custom coprocessor interface is memory-mapped, protection of the addresses used for the interface can be provided with the segmentation mechanism of the 80376. If the custom interface is I/O-mapped, protection of the interface can be provided with the 80376 IOPL (I/O Privilege Level) mechanism.

The 80387SX numeric coprocessor interface is I/O mapped as shown in Table 4.8. Note that the 80387SX coprocessor interface addresses are beyond the 0H-0FFFFH range for programmed I/O. When the 80376 supports the 80387SX coprocessor, the 80376 automatically generates bus cycles to the coprocessor interface addresses.

Table 4.8 Numeric Coprocessor Port Addresses

Address in 80376 I/O Space	80387SX Coprocessor Register
8000F8H	Opcode Register
8000FCH	Operand Register
8000FEH	Operand Register

SOFTWARE TESTING FOR COPROCESSOR PRESENCE

When software is used to test coprocessor (80387SX) presence, it should use only the following coprocessor opcodes: FNINIT, FNSTCW and FNSTSW. To use other coprocessor opcodes when a coprocessor is known to be not present, first set EM = 1 in the 80376 CR0 register.

5.0 PACKAGE THERMAL SPECIFICATIONS

The Intel 80376 embedded processor is specified for operation when case temperature is within the range of 0°C–115°C for both the ceramic 88-pin PGA package and the plastic 100-pin PQFP package. The case temperature may be measured in any environment, to determine whether the 80376 is within specified operating range. The case temperature should be measured at the center of the top surface.

The ambient temperature is guaranteed as long as T_C is not violated. The ambient temperature can be calculated from the θ_{jc} and θ_{ja} from the following equations:

$$T_J = T_C + P^*\theta_{jc}$$

$$T_A = T_j - P^*\theta_{ja}$$

$$T_C = T_a + P^*[\theta_{ja} - \theta_{jc}]$$

Values for θ_{ja} and θ_{jc} are given in Table 5.1 for the 100-lead fine pitch. θ_{ja} is given at various airflows. Table 5.2 shows the maximum T_a allowable (without exceeding T_C) at various airflows. Note that T_a can be improved further by attaching "fins" or a "heat sink" to the package. P is calculated using the maximum ***cold*** I_{CC} of 305 mA and the maximum V_{CC} of 5.5V for both packages.

Table 5.1. 80376 Package Thermal Characteristics Thermal Resistances (°C/Watt) θ_{jc} and θ_{ja}

Package	θ_{jc}	θ_{ja} Versus Airflow-ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
100-Lead Fine Pitch	7.5	34.5	29.5	25.5	22.5	21.5	21.0
88-Pin PGA	2.5	29.0	22.5	17.0	14.5	12.5	12.0

Table 5.2. 80376 Maximum Allowable Ambient Temperature at Various Airflows

Package	θ_{jc}	T_A(°C) vs Airflow-ft/min (m/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
100-Lead Fine Pitch	7.5	70	78	85	90	92	93
88-Pin PGA	2.5	70	81	90	95	98	99

6.0 ELECTRICAL SPECIFICATIONS

The following sections describe recommended electrical connections for the 80376, and its electrical specifications.

6.1 Power and Grounding

The 80376 is implemented in CHMOS IV technology and has modest power requirements. However, its high clock frequency and 47 output buffers (address, data, control, and HLDA) can cause power surges as multiple output buffers drive new signal levels simultaneously. For clean on-chip power distribution at high frequency, 14 V_{CC} and 18 V_{SS} pins separately feed functional units of the 80376.

Power and ground connections must be made to all external V_{CC} and GND pins of the 80376. On the circuit board, all V_{CC} pins should be connected on a V_{CC} plane and all V_{SS} pins should be connected on a GND plane.

POWER DECOUPLING RECOMMENDATIONS

Liberal decoupling capacitors should be placed near the 80376. The 80376 driving its 24-bit address bus and 16-bit data bus at high frequencies can cause transient power surges, particularly when driving large capacitive loads. Low inductance capacitors and interconnects are recommended for best high frequency electrical performance. Inductance can be reduced by shortening circuit board traces between the 80376 and decoupling capacitors as much as possible.

RESISTOR RECOMMENDATIONS

The $\overline{\text{ERROR}}$, $\overline{\text{FLT}}$ and $\overline{\text{BUSY}}$ inputs have internal pull-up resistors of approximately 20 KΩ and the PEREQ input has an internal pull-down resistor of approximately 20 KΩ built into the 80376 to keep these signals inactive when the 80387SX is not present in the system (or temporarily removed from its socket).

In typical designs, the external pull-up resistors shown in Table 6.1 are recommended. However, a particular design may have reason to adjust the resistor values recommended here, or alter the use of pull-up resistors in other ways.

Table 6.1. Recommended Resistor Pull-Ups to V_{CC}

Pin	Signal	Pull-Up Value	Purpose
16	$\overline{ADS}$	20 KΩ ± 10%	Lightly Pull $\overline{ADS}$ Inactive during 80376 Hold Acknowledge States
26	$\overline{LOCK}$	20 KΩ ± 10%	Lightly Pull $\overline{LOCK}$ Inactive during 80376 Hold Acknowledge States

OTHER CONNECTION RECOMMENDATIONS

For reliable operation, always connect unused inputs to an appropriate signal level. N/C pins should always remain **unconnected. Connection of N/C pins to V_{CC} or V_{SS} will result in incompatibility with future steppings of the 80376.**

Particularly when not using interrupts or bus hold (as when first prototyping), prevent any chance of spurious activity by connecting these associated inputs to GND:

—INTR
—NMI
—HOLD

If not using address pipelining connect the $\overline{NA}$ pin to a pull-up resistor in the range of 20 KΩ to V_{CC}.

6.2 Absolute Maximum Ratings

Table 6.2. Maximum Ratings

Parameter	Maximum Rating
Storage Temperature	−65°C to +150°C
Case Temperature under Bias	−65°C to +120°C
Supply Voltage with Respect to V_{SS}	−0.5V to +6.5V
Voltage on Other Pins	−0.5V to (V_{CC} + 0.5)V

Table 6.2 gives a stress ratings only, and functional operation at the maximums is not guaranteed. Functional operating conditions are given in **Section 6.3, D.C. Specifications,** and **Section 6.4, A.C. Specifications.**

Extended exposure to the Maximum Ratings may affect device reliability. Furthermore, although the 80376 contains protective circuitry to resist damage from static electric discharge, always take precautions to avoid high static voltages or electric fields.

6.3 D.C. Specifications

ADVANCE INFORMATION SUBJECT TO CHANGE

Table 6.3: 80376 D.C. Characteristics

Functional Operating Range: V_{CC} = 5V ±10%; T_{CASE} = 0°C to 115°C for 88-pin PGA or 100-pin PQFP

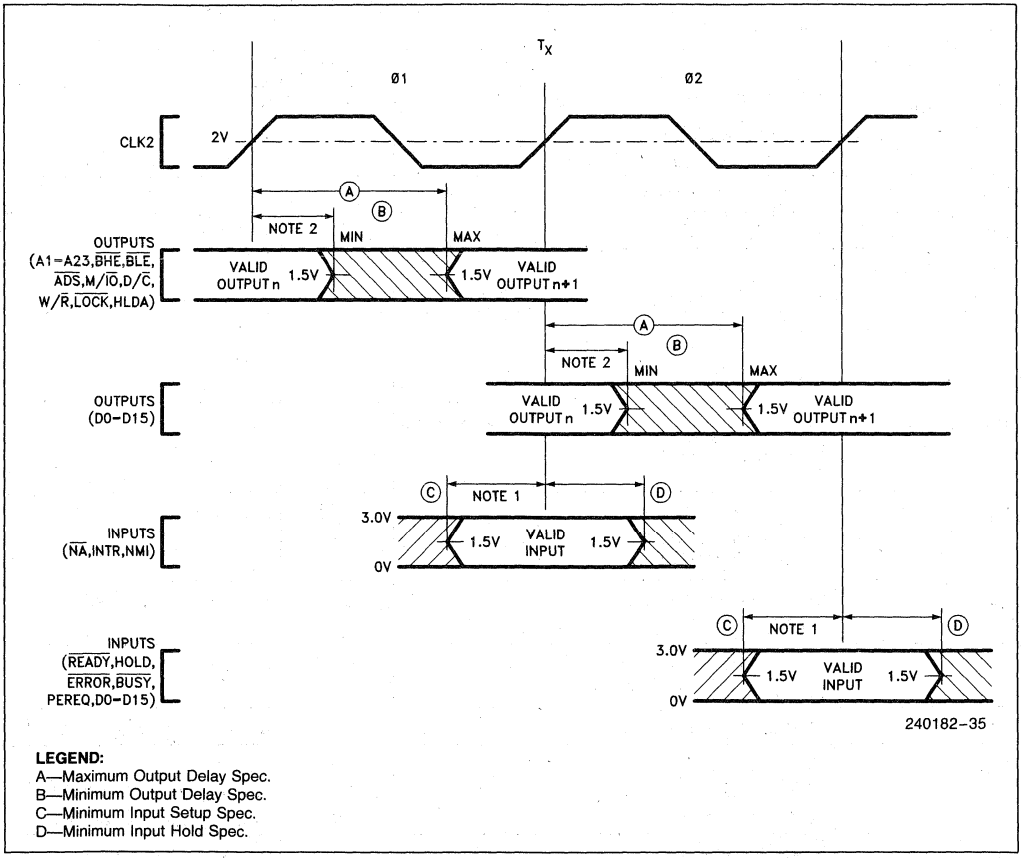
Symbol	Parameter	Min	Max	Unit
V_{IL}	Input LOW Voltage	−0.3	+0.8	V(1)
V_{IH}	Input HIGH Voltage	2.0	V_{CC} +0.3	V(1)
V_{ILC}	CLK2 Input LOW Voltage	−0.3	+0.8	V(1)
V_{IHC}	CLK2 Input HIGH Voltage	V_{CC} − 0.8	V_{CC} + 0.3	V(1)
V_{OL}	Output LOW Voltage			
I_{OL} = 4 mA:	A_{23}–A_1, D_{15}–D_0		0.45	V(1)
I_{OL} = 5 mA:	$\overline{BHE}$, $\overline{BLE}$, W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{LOCK}$, $\overline{ADS}$, HLDA		0.45	V(1)
V_{OH}	Output High Voltage			
I_{OH} = −1 mA:	A_{23}–A_1, D_{15}–D_0	2.4		V(1)
I_{OH} = −0.2 mA:	A_{23}–A_1, D_{15}–D_0	V_{CC} − 0.5		V(1)
I_{OH} = −0.9 mA:	$\overline{BHE}$, $\overline{BLE}$, W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{LOCK}$, $\overline{ADS}$, HLDA	2.4		V(1)
I_{OH} = −0.18 mA:	$\overline{BHE}$, $\overline{BLE}$, W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{LOCK}$ $\overline{ADS}$, HLDA	V_{CC} − 0.5		V(1)
I_{LI}	Input Leakage Current (For All Pins except PEREQ, $\overline{BUSY}$, $\overline{FLT}$ and $\overline{ERROR}$)		±15	μA, 0V ≤ V_{IN} ≤ V_{CC}(1)
I_{IH}	Input Leakage Current (PEREQ Pin)		200	μA, V_{IH} = 2.4V(1, 2)
I_{IL}	Input Leakage Current ($\overline{BUSY}$ and $\overline{ERROR}$ Pins)		−400	μA, V_{IL} = 0.45V(3)
I_{LO}	Output Leakage Current		±15	μA, 0.45V ≤ V_{OUT} ≤ V_{CC}(1)
I_{CC}	Supply Current CLK2 = 32 MHz CLK2 = 40 MHz		 275 305	 mA, I_{CC} typ = 175 mA(4) mA, I_{CC} typ = 200 mA(4)
C_{IN}	Input Capacitance		10	pF, F_C = 1 MHz(5)
C_{OUT}	Output or I/O Capacitance		12	pF, F_C = 1 MHz(5)
C_{CLK}	CLK2 Capacitance		20	pF, F_C = 1 MHz(5)

NOTES:
1. Tested at the minimum operating frequency of the device.
2. PEREQ input has an internal pull-down resistor.
3. $\overline{BUSY}$, $\overline{FLT}$ and $\overline{ERROR}$ inputs each have an internal pull-up resistor.
4. I_{CC} max measurement at worse case load, V_{CC} and temperature (0°C).
5. Not 100% tested.

The A.C. specifications given in Table 6.4 consist of output delays, input setup requirements and input hold requirements. All A.C. specifications are relative to the CLK2 rising edge crossing the 2.0V level.

A.C. specification measurement is defined by Figure 6.1. Inputs must be driven to the voltage levels indicated by Figure 6.1 when A.C. specifications are measured. 80376 output delays are specified with minimum and maximum limits measured as shown. The minimum 80376 delay times are hold times provided to external circuitry. 80376 input setup and hold times are specified as minimums, defining the smallest acceptable sampling window. Within the sampling window, a synchronous input signal must be stable for correct 80376 processor operation.

Outputs $\overline{NA}$, W/$\overline{R}$, D/$\overline{C}$, M/$\overline{IO}$, $\overline{LOCK}$, $\overline{BHE}$, $\overline{BLE}$, A_{23}–A_1 and HLDA only change at the beginning of phase one. D_{15}–D_0 (write cycles) only change at the beginning of phase two. The $\overline{READY}$, HOLD, $\overline{BUSY}$, $\overline{ERROR}$, PEREQ and D_{15}–D_0 (read cycles) inputs are sampled at the beginning of phase one. The $\overline{NA}$, INTR and NMI inputs are sampled at the beginning of phase two.



LEGEND:
A—Maximum Output Delay Spec.
B—Minimum Output Delay Spec.
C—Minimum Input Setup Spec.
D—Minimum Input Hold Spec.

Figure 6.1. Drive Levels and Measurement Points for A.C. Specifications

6.4 A.C. Specifications

Table 6.4. 80376 A.C. Characteristics at 16 MHz

Functional Operating Range: V_{CC} = 5V ±10%; T_{CASE} = 0°C to 115°C for 88-pin PGA or 100-pin PQFP

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Operating Frequency	4	16	MHz		Half CLK2 Freq
t_1	CLK2 Period	31	125	ns	6.3	
t_{2a}	CLK2 HIGH Time	9		ns	6.3	At 2(3)
t_{2b}	CLK2 HIGH Time	5		ns	6.3	At (V_{CC} − 0.8)V(3)
t_{3a}	CLK2 LOW Time	9		ns	6.3	At 2V(3)
t_{3b}	CLK2 LOW Time	7		ns	6.3	At 0.8V(3)
t_4	CLK2 Fall Time		8	ns	6.3	(V_{CC}−0.8)V to 0.8V(3)
t_5	CLK2 Rise Time		8	ns	6.3	0.8V to (V_{CC}−0.8)(3)
t_6	A_{23}–A_1 Valid Delay	4	36	ns	6.5	C_L = 120 pF(4)
t_7	A_{23}–A_1 Float Delay	4	40	ns	6.6	(1)
t_8	$\overline{BHE}$, $\overline{BLE}$, $\overline{LOCK}$ Valid Delay	4	36	ns	6.5	C_L = 75 pF(4)
t_9	$\overline{BHE}$, $\overline{BLE}$, $\overline{LOCK}$ Float Delay	4	40	ns	6.6	(1)
t_{10}	W/$\overline{R}$, M/$\overline{IO}$, D/$\overline{C}$, $\overline{ADS}$ Valid Delay	6	33	ns	6.5	C_L = 75 pF(4)
t_{11}	W/$\overline{R}$, M/$\overline{IO}$, D/$\overline{C}$, $\overline{ADS}$ Float Delay	6	35	ns	6.6	(1)
t_{12}	D_{15}–D_0 Write Data Valid Delay	4	40	ns	6.5	C_L = 120 pF(4)
t_{13}	D_{15}–D_0 Write Data Float Delay	4	35	ns	6.6	(1)
t_{14}	HLDA Valid Delay	4	33	ns	6.6	C_L = 75 pF(4)
t_{15}	$\overline{NA}$ Setup Time	5		ns	6.4	
t_{16}	$\overline{NA}$ Hold Time	21		ns	6.6	
t_{19}	$\overline{READY}$ Setup Time	19		ns	6.4	
t_{20}	$\overline{READY}$ Hold Time	4		ns	6.4	
t_{21}	Setup Time D_{15}–D_0 Read Data	9		ns	6.4	
t_{22}	Hold Time D_{15}–D_0 Read Data	6		ns	6.4	
t_{23}	HOLD Setup Time	26		ns	6.4	
t_{24}	HOLD Hold Time	5		ns	6.4	
t_{25}	RESET Setup Time	13		ns	6.7	
t_{26}	RESET Hold Time	4		ns	6.7	

5

Table 6.4. 80376 A.C. Characteristics at 16 MHz (Continued)
Functional Operating Range: V_{CC} = 5V ±10%; T_{CASE} = 0°C to 115°C for 88-pin PGA or 100-pin PQFP

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{27}	NMI, INTR Setup Time	16		ns	6.4	(2)
t_{28}	NMI, INTR Hold Time	16		ns	6.4	(2)
t_{29}	PEREQ, $\overline{ERROR}$, $\overline{BUSY}$, $\overline{FLT}$ Setup Time	16		ns	6.4	(2)
t_{30}	PEREQ, $\overline{ERROR}$, $\overline{BUSY}$, $\overline{FLT}$ Hold Time	5		ns	6.4	(2)

NOTES:
1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.
2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.
3. These are not tested. They are guaranteed by design characterization.
4. Tested with C_L set to 50 pF and derated to support the indicated distributed capacitive load. See Figures 6.8 through 6.10 for capacitive derating curves.
5. The 80376 does not have t_{17} or t_{18} timing specifications.

Table 6.5. 80376 A.C. Characteristics at 20 MHz
Functional Operating Range: V_{CC} = 5V ±10%; T_{CASE} = 0°C to 115°C for 88-pin PGA or 100-pin PQFP

Symbol	Parameter	Min	Max	Unit	Figure	Notes
	Operating Frequency	4	20	MHz		Half CLK2 Frequency
t_1	CLK2 Period	25	125	ns	6.3	
t_{2a}	CLK2 HIGH Time	8		ns	6.3	At 2V(3)
t_{2b}	CLK2 HIGH Time	5		ns	6.3	At (V_{CC} − 0.8)V(3)
t_{3a}	CLK2 LOW Time	8		ns	6.3	At 2V(3)
t_{3b}	CLK2 LOW Time	6		ns	6.3	At 0.8V(3)
t_4	CLK2 Fall Time		8	ns	6.3	(V_{CC}−0.8V) to 0.8V(3)
t_5	CLK2 Rise Time		8	ns	6.3	0.8V to (V_{CC}−0.8)(3)
t_6	A_{23}–A_1 Valid Delay	4	30	ns	6.5	C_L = 120 pF(4)
t_7	A_{23}–A_1 Float Delay	4		ns	6.6	(1)
t_8	$\overline{BHE}$, $\overline{BLE}$, $\overline{LOCK}$ Valid Delay	4	30	ns	6.5	C_L = 75 pF(4)
t_9	$\overline{BHE}$, $\overline{BLE}$, $\overline{LOCK}$ Float Delay	4	32	ns	6.6	(1)
t_{10a}	M/$\overline{IO}$, D/$\overline{C}$ Valid Delay	6	28	ns	6.5	C_L = 75 pF(4)
t_{10b}	W/$\overline{R}$, $\overline{ADS}$ Valid Delay	6	26	ns	6.5	C_L = 75 pF(4)
t_{11}	W/$\overline{R}$, M/$\overline{IO}$, D/$\overline{C}$, $\overline{ADS}$ Float Delay	6	30	ns	6.6	(1)
t_{12}	D_{15}–D_0 Write Data Valid Delay	4	38	ns	6.5	C_L = 120 pF
t_{13}	D_{15}–D_0 Write Data Float Delay	4	27	ns	6.6	(1)

Table 6.5. 80376 A.C. Characteristics at 20 MHz (Continued)
Functional Operating Range: V_{CC} = 5V ±10%; T_{CASE} = 0°C to 115°C for 88-pin PGA or 100-pin PQFP

Symbol	Parameter	Min	Max	Unit	Figure	Notes
t_{14}	HLDA Valid Delay	4	28	ns	6.5	C_L = 75 pF(4)
t_{15}	$\overline{NA}$ Setup Time	5		ns	6.4	
t_{16}	$\overline{NA}$ Hold Time	12		ns	6.4	
t_{19}	$\overline{READY}$ Setup Time	12		ns	6.4	
t_{20}	$\overline{READY}$ Hold Time	4		ns	6.4	
t_{21}	D_{15}–D_0 Read Data Setup Time	9		ns	6.4	
t_{22}	D_{15}–D_0 Read Data Hold Time	6		ns	6.4	
t_{23}	HOLD Setup Time	17		ns	6.4	
t_{24}	HOLD Hold Time	5		ns	6.4	
t_{25}	RESET Setup Time	12		ns	6.7	
t_{26}	RESET Hold Time	4		ns	6.7	
t_{27}	NMI, INTR Setup Time	16		ns	6.4	(2)
t_{28}	NMI, INTR Hold Time	16		ns	6.4	(2)
t_{29}	PEREQ, $\overline{ERROR}$, $\overline{BUSY}$, $\overline{FLT}$ Setup Time	14		ns	6.4	(2)
t_{30}	PEREQ, $\overline{ERROR}$, $\overline{BUSY}$, $\overline{FLT}$ Hold Time	5		ns	6.4	(2)

NOTES:
1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not 100% tested.
2. These inputs are allowed to be asynchronous to CLK2. The setup and hold specifications are given for testing purposes, to assure recognition within a specific CLK2 period.
3. These are not tested. They are guaranteed by design characterization.
4. Tested with C_L set to 50 pF and derated to support the indicated distributed capacitive load. See Figures 6.8 through 6.10 for capacitive derating curves.
5. The 80376 does not have t_{17} or t_{18} timing specifications.

5

A.C. TEST LOADS

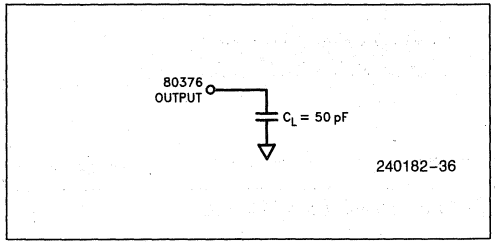


Figure 6.2. A.C. Test Loads

A.C. TIMING WAVEFORMS

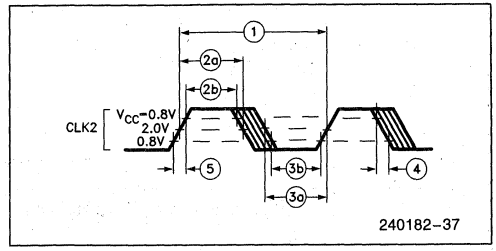


Figure 6.3. CLK2 Waveform

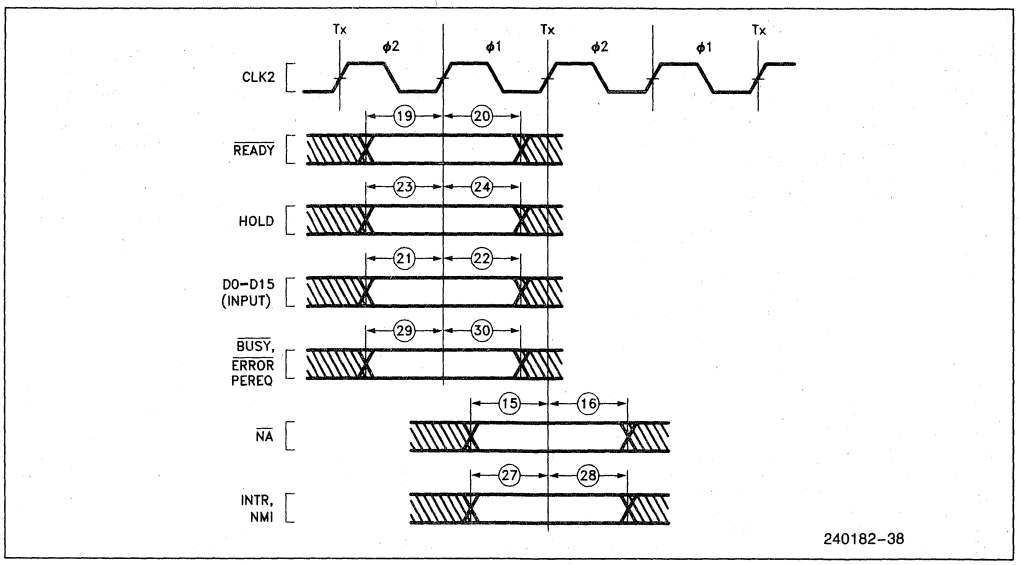


Figure 6.4. A.C. Timing Waveforms—Input Setup and Hold Timing

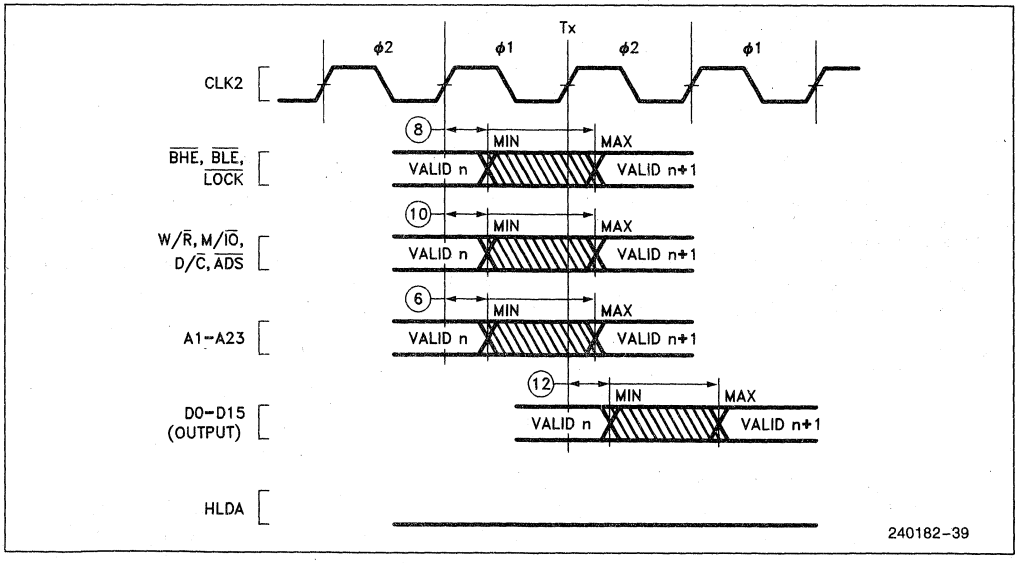


Figure 6.5. A.C. Timing Waveforms—Output Valid Delay Timing

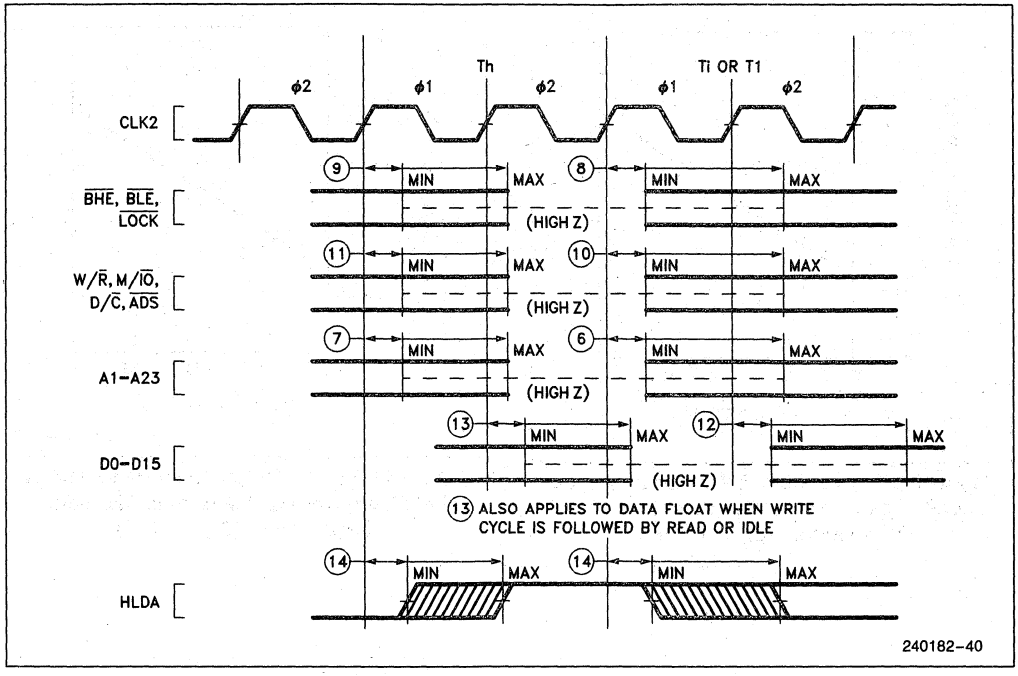
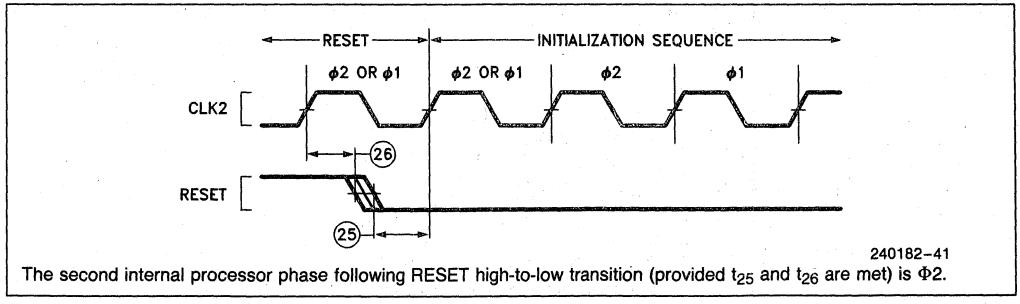


Figure 6.6. A.C. Timing Waveforms—Output Float Delay and HLDA Valid Delay Timing



The second internal processor phase following RESET high-to-low transition (provided t_{25} and t_{26} are met) is Φ2.

Figure 6.7. A.C. Timing Waveforms—RESET Setup and Hold Timing, and Internal Phase

5

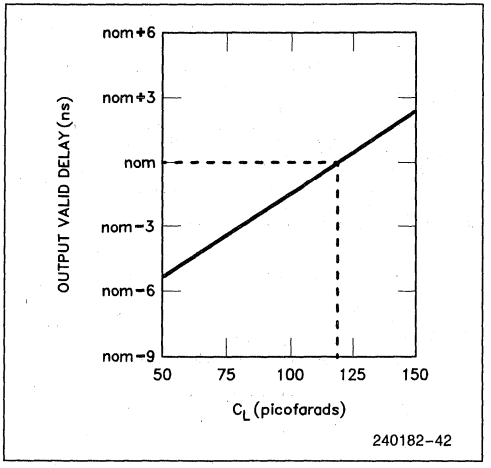


Figure 6.8. Typical Output Valid Delay versus Load Capacitance at Maximum Operating Temperature (C_L = 120 pF)

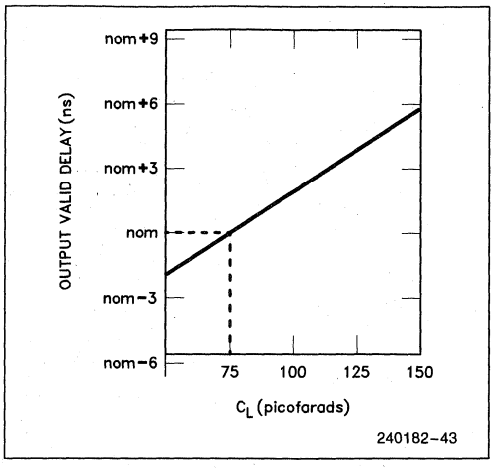


Figure 6.9. Typical Output Valid Delay versus Load Capacitance at Maximum Operating Temperature (C_L = 75 pF)

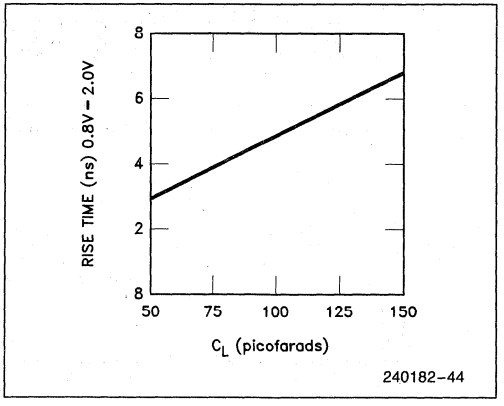


Figure 6.10. Typical Output Rise Time versus Load Capacitance at Maximum Operating Temperature

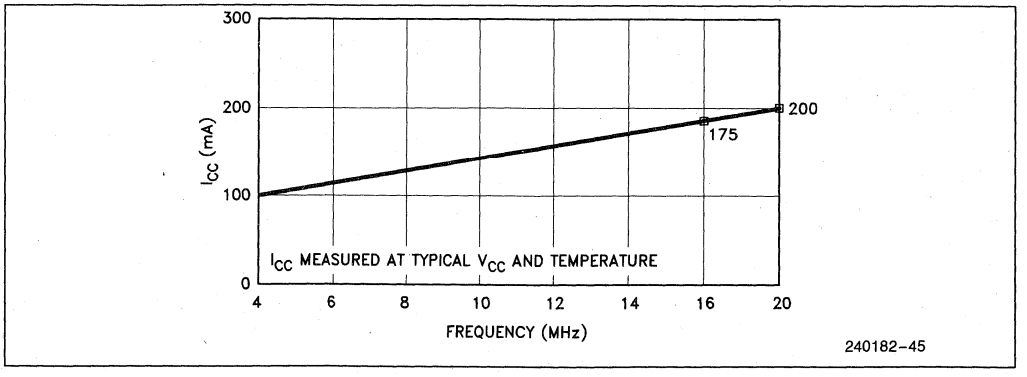


Figure 6.11. Typical I_{CC} vs Frequency

6.5 Designing for the ICE™-376 Emulator

The 376 embedded processor in-circuit emulator product is the ICE-376 emulator. Use of the emulator requires the target system to provide a socket that is compatible with the ICE-376 emulator. The 80376 offers two different probes for emulating user systems: an 88-pin PGA probe and a 100-pin fine pitch flat-pack probe. The 100-pin fine pitch flat-pack probe requires a socket, called the 100-pin PQFP, which is available from 3-M Textool (part number 2-0100-07243-000). The ICE-376 emulator probe attaches to the target system via an adapter which replaces the 80376 component in the target system. Because of the high operating frequency of 80376 systems and of the ICE-376 emulator, there is no buffering between the 80376 emulation processor in the ICE-376 emulator probe and the target system. A direct result of the non-buffered interconnect is that the ICE-376 emulator shares the address and data bus with the user's system, and the RESET signal is intercepted by the ICE emulator hardware. In order for the ICE-376 emulator to be functional in the user's system without the Optional Isolation Board (OIB) the designer must be aware of the following conditions:

1. The bus controller must only enable data transceivers onto the data bus during valid read cycles of the 80376, other local devices or other bus masters.
2. Before another bus master drives the local processor address bus, the other master must gain control of the address bus by asserting HOLD and receiving the HLDA response.
3. The emulation processor receives the RESET signal 2 or 4 CLK2 cycles later than an 80376 would, and responds to RESET later. Correct phase of the response is guaranteed.

In addition to the above considerations, the ICE-376 emulator processor module has several electrical and mechanical characteristics that should be taken into consideration when designing the 80376 system.

Capacitive Loading: ICE-376 adds up to 27 pF to each 80376 signal.

Drive Requirements: ICE-376 adds one FAST TTL load on the CLK2, control, address, and data lines. These loads are within the processor module and are driven by the 80376 emulation processor, which has standard drive and loading capability listed in Tables 6.3 and 6.4.

Power Requirements: For noise immunity and CMOS latch-up protection the ICE-376 emulator processor module is powered by the user system. The circuitry on the processor module draws up to 1.4A including the maximum 80376 I_{CC} from the user 80376 socket.

80376 Location and Orientation: The ICE-376 emulator processor module may require lateral clearance. Figure 6.12 shows the clearance requirements of the iMP adapter and Figure 6.13 shows the clearance requirements of the 88-pin PGA adapter. The

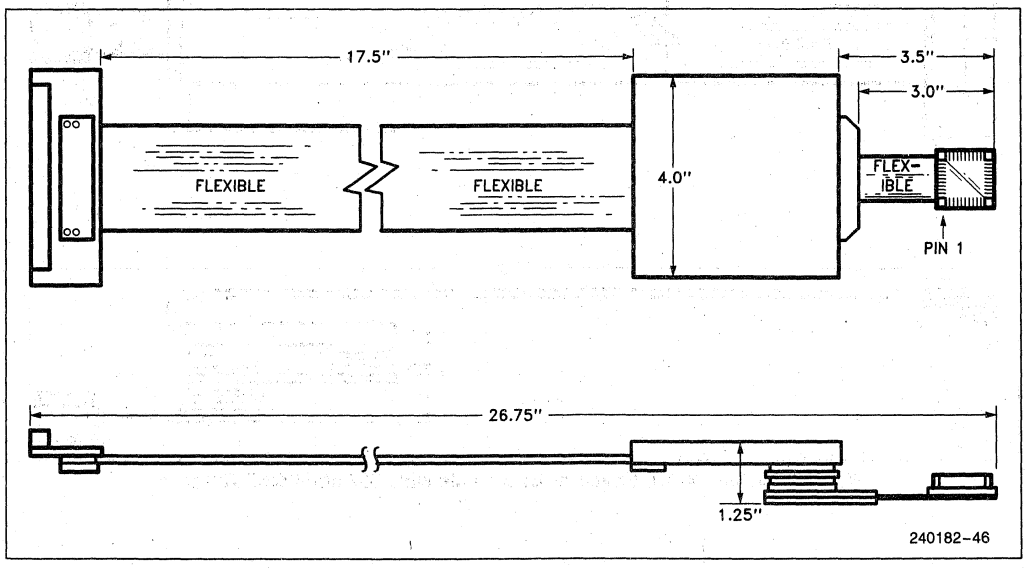


Figure 6.12. Preliminary ICE™-376 Emulator User Cable with PQFP Adapter

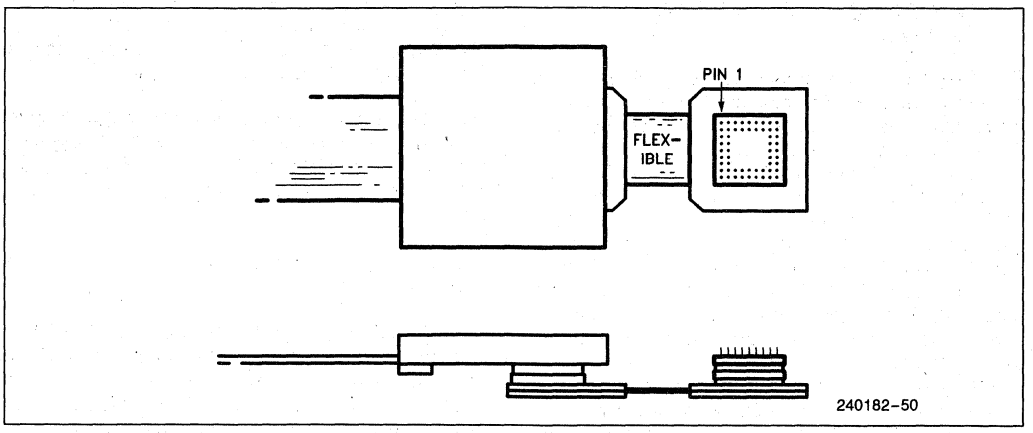


Figure 6.13. ICE™-376 Emulator User Cable with 88-Pin PGA Adapter

optional isolation board (OIB), which provides extra electrical buffering and has the same lateral clearance requirements as Figures 6.12 and 6.13, adds an additional 0.5 inches to the vertical clearance requirement. This is illustrated in Figure 6.14.

Optional Isolation Board (OIB) and the CLK2 speed reduction: Due to the unbuffered probe design, the ICE-376 emulator is susceptible to errors on the user's bus. The OIB allows the ICE-376 emulator to function in user systems with faults (shorted signals, etc.). After electrical verification the OIB may be removed. When the OIB is installed, the user system must have a maximum CLK2 frequency of 20 MHz.

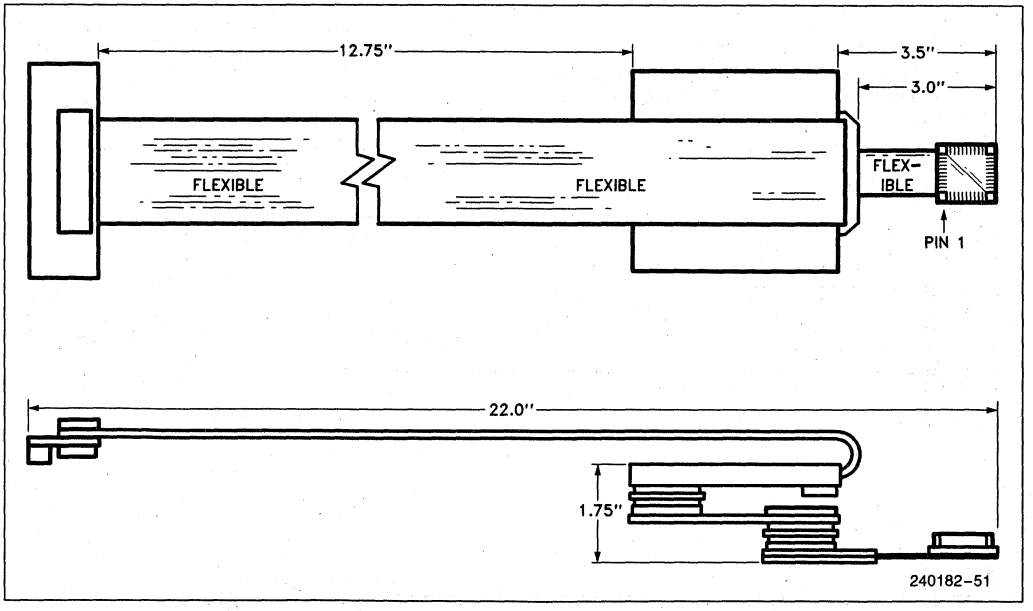


Figure 6.14. ICE™-376 Emulator User Cable with OIB and PQFP Adapter

7.0 DIFFERENCES BETWEEN THE 80376 AND THE 80386

The following are the major differences between the 80376 and the 80386.

1. The 80376 generates byte selects on $\overline{BHE}$ and $\overline{BLE}$ (like the 8086 and 80286 microprocessors) to distinguish the upper and lower bytes on its 16-bit data bus. The 80386 uses four-byte selects, $\overline{BE0}$–$\overline{BE3}$, to distinguish between the different bytes on its 32-bit bus.
2. The 80376 has no bus sizing option. The 80386 can select between either a 32-bit bus or a 16-bit bus by use of the $\overline{BS16}$ input. The 80376 has a 16-bit bus size.
3. The $\overline{NA}$ pin operation in the 80376 is identical to that of the $\overline{NA}$ pin on the 80386 with one exception: the $\overline{NA}$ pin of the 80386 cannot be activated on 16-bit bus cycles (where $\overline{BS16}$ is LOW in the 80386 case), whereas $\overline{NA}$ can be activated on any 80376 bus cycle.
4. The contents of all 80376 registers at reset are identical to the contents of the 80386 registers at reset, except the DX register. The DX register contains a component-stepping identifier at reset, i.e.

   in 80386, after reset DH = 03H indicates 80386
   DL = revision number;
   in 80376, after reset DH = 33H indicates 80376
   DL = revision number.

5. The 80386 uses A_{31} and M/$\overline{IO}$ as a select for numerics coprocessor. The 80376 uses the A_{23} and M/$\overline{IO}$ to select its numerics coprocessor.
6. The 80386 prefetch unit fetches code in four-byte units. The 80376 prefetch unit reads two bytes as one unit (like the 80286 microprocessor). In $\overline{BS16}$ mode, the 80386 takes two consecutive bus cycles to complete a prefetch request. If there is a data read or write request after the prefetch starts, the 80386 will fetch all four bytes before addressing the new request.
7. The 80376 has no paging mechanism.
8. The 80376 starts executing code in what corresponds to the 80386 protected mode. The 80386 starts execution in real mode, which is then used to enter protected mode.
9. The 80386 has a virtual-86 mode that allows the execution of a real mode 8086 program as a task in protected mode. The 80376 has no virtual-86 mode.
10. The 80386 maps a 48-bit logical address into a 32-bit physical address by segmentation and paging. The 80376 maps its 48-bit logical address into a 24-bit physical address by segmentation only.
11. The 80376 uses the 80387SX numerics coprocessor for floating point operations, while the 80386 uses the 80387 coprocessor.
12. The 80386 can execute from 16-bit code segments. The 80376 can **only** execute from 32-bit code Segments.
13. The 80376 has an input called $\overline{FLT}$ which three-states all bidirectional and output pins, including HLDA, when asserted. It is used with ON Circuit Emulation (ONCE).

8.0 INSTRUCTION SET

This section describes the 376 embedded processor instruction set. Table 8.1 lists all instructions along with instruction encoding diagrams and clock counts. Further details of the instruction encoding are then provided in the following sections, which completely describe the encoding structure and the definition of all fields occurring within 80376 instructions.

5

8.1 80376 Instruction Encoding and Clock Count Summary

To calculate elapsed time for an instruction, multiply the instruction clock count, as listed in Table 8.1 below, by the processor clock period (e.g. 50 ns for an 80376 operating at 20 MHz). The actual clock count of an 80376 program will average 10% more

than the calculated clock count due to instruction sequences which execute faster than they can be fetched from memory.

Instruction Clock Count Assumptions:

1. The instruction has been prefetched, decoded, and is ready for execution.
2. Bus cycles do not require wait states.
3. There are no local bus HOLD requests delaying processor acess to the bus.
4. No exceptions are detected during instruction execution.
5. If an effective address is calculated, it does not use two general register components. One register, scaling and displacement can be used within the clock counts showns. However, if the effective address calculation uses two general register components, add 1 clock to the clock count shown.
6. Memory reference instruction accesses byte or aligned 16-bit operands.

Instruction Clock Count Notation

— If two clock counts are given, the smaller refers to a register operand and the larger refers to a memory operand.

—n = number of times repeated.

—m = number of components in the next instruction executed, where the entire displacement (if any) counts as one component, the entire immediate data (if any) counts as one component, and all other bytes of the instruction and prefix(es) each count as one component.

Misaligned or 32-Bit Operand Accesses:

— If instructions accesses a misaligned 16-bit operand or 32-bit operand on even address add:

2* clocks for read or write.

4** clocks for read and write.

— If instructions accesses a 32-bit operand on odd address add:

4* clocks for read or write.

8** clocks for read and write.

Wait States:

Wait states add 1 clock per wait state to instruction execution for each data access.

Table 8.1. 80376 Instruction Set Clock Count Summary

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
GENERAL DATA TRANSFER				
MOV = Move:				
Register to Register/Memory	1000100w mod reg r/m	2/2*	0/1*	a
Register/Memory to Register	1000101w mod reg r/m	2/4*	0/1*	a
Immediate to Register/Memory	1100011w mod000 r/m immediate data	2/2*	0/1*	a
Immediate to Register (Short Form)	1011 w reg immediate data	2	2	
Memory to Accumulator (Short Form)	1010000w full displacement	4*	1*	a
Accumulator to Memory (Short Form)	1010001w full displacement	2*	1*	a
Register/Memory to Segment Register	10001110 mod sreg3 r/m	22/23*	0/6*	a,b,c
Segment Register to Register/Memory	10001100 mod sreg3 r/m	2/2*	0/1*	a
MOVSX = Move with Sign Extension				
Register from Register/Memory	00001111 1011111w mod reg r/m	3/6*	0/1*	a
MOVZX = Move with Zero Extension				
Register from Register/Memory	00001111 1011011w mod reg r/m	3/6*	0/1*	a
PUSH = Push:				
Register/Memory	11111111 mod110 r/m	7/9*	2/4*	a
Register (Short Form)	01010 reg	4	2	a
Segment Register (ES, CS, SS or DS)	000sreg2110	4	2	a
Segment Register (FS or GS)	00001111 10sreg3000	4	2	a
Immediate	011010s0 immediate data	4	2	a
PUSHA = Push All	01100000	34	16	a
POP = Pop				
Register/Memory	10001111 mod000 r/m	7/9*	2/4*	a
Register (Short Form)	01011 reg	6	2	a
Segment Register (ES, SS or DS)	000sreg2111	25	6	a, b, c
Segment Register (FS or GS)	00001111 10sreg3001	25	6	a, b, c
POPA = Pop All	01100001	40	16	a
XCHG = Exchange				
Register/Memory with Register	1000011w mod reg r/m	3/5**	0/2**	a, m
Register with Accumulator (Short Form)	10010 reg	3	0	
IN = Input from:				
Fixed Port	1110010w port number	6*	1*	f,k
		26*	1*	f,l
Variable Port	1110110w	7*	1*	f,k
		27*	1*	f,l
OUT = Output to:				
Fixed Port	1110011w port number	4*	1*	f,k
		24*	1*	f,l
Variable Port	1110111w	5*	1*	f,k
		26*	1*	f,l
LEA = Load EA to Register	10001101 mod reg r/m	2		

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
SEGMENT CONTROL				
LDS = Load Pointer to DS	11000101 | mod reg r/m	26*	6*	a, b, c
LES = Load Pointer to ES	11000100 | mod reg r/m	26*	6*	a, b, c
LFS = Load Pointer to FS	00001111 | 10110100 | mod reg r/m	29*	6*	a, b, c
LGS = Load Pointer to GS	00001111 | 10110101 | mod reg r/m	29*	6*	a, b, c
LSS = Load Pointer to SS	00001111 | 10110010 | mod reg r/m	26*	6*	a, b, c
FLAG CONTROL				
CLC = Clear Carry Flag	11111000	2		
CLD = Clear Direction Flag	11111100	2		
CLI = Clear Interrupt Enable Flag	11111010	8		f
CLTS = Clear Task Switched Flag	00001111 | 00000110	5		e
CMC = Complement Carry Flag	11110101	2		
LAHF = Load AH into Flag	10011111	2		
POPF = Pop Flags	10011101	7		a, g
PUSHF = Push Flags	10011100	4		a
SAHF = Store AH into Flags	10011110	3		
STC = Set Carry Flag	11111001	2		
STD = Set Direction Flag	11111101	2		
STI = Set Interrupt Enable Flag	11111011	8		f
ARITHMETIC				
ADD = Add				
Register to Register	000000dw | mod reg r/m	2		
Register to Memory	0000000w | mod reg r/m	7**	2**	a
Memory to Register	0000001w | mod reg r/m	6*	1*	a
Immediate to Register/Memory	100000sw | mod000 r/m | immediate data	2/7**	0/2**	a
Immediate to Accumulator (Short Form)	0000010w | immediate data	2		
ADC = Add with Carry				
Register to Register	000100dw | mod reg r/m	2		
Register to Memory	0001000w | mod reg r/m	7**	2**	a
Memory to Register	0001001w | mod reg r/m	6*	1*	a
Immediate to Register/Memory	100000sw | mod010 r/m | immediate data	2/7**	0/2**	a
Immediate to Accumulator (Short Form)	0001010w | immediate data	2		
INC = Increment				
Register/Memory	1111111w | mod000 r/m	2/6**	0/2**	a
Register (Short Form)	01000 reg	2		
SUB = Subtract				
Register from Register	001010dw | mod reg r/m	2		

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number Of Data Cycles	Notes
ARITHMETIC (Continued)				
Register from Memory	0010100w | mod reg r/m	7**	2**	a
Memory from Register	0010101w | mod reg r/m	6*	1	a
Immediate from Register/Memory	100000sw | mod101 r/m | immediate data	2/7**	0/1**	a
Immediate from Accumulator (Short Form)	0010110w | immediate data	2		
SBB = Subtract with Borrow				
Register from Register	000110dw | mod reg r/m	2		
Register from Memory	0001100w | mod reg r/m	7**	2**	a
Memory from Register	0001101w | mod reg r/m	6*	1*	a
Immediate from Register/Memory	100000sw | mod011 r/m | immediate data	2/7**	0/2**	a
Immediate from Accumulator (Short Form)	0001110w | immediate data	2		
DEC = Decrement				
Register/Memory	1111111w | reg001 r/m	2/6**	0/2**	a
Register (Short Form)	01001 reg	2		
CMP = Compare				
Register with Register	001110dw | mod reg r/m	2		
Memory with Register	0011100w | mod reg r/m	5*	1*	a
Register with Memory	0011101w | mod reg r/m	6**	2**	a
Immediate with Register/Memory	100000sw | mod111 r/m | immediate data	2/5*	0/1*	a
Immediate with Accumulator (Short Form)	0011110w | immediate data	2		
NEG = Change Sign	1111011w | mod011 r/m	2/6*	0/2*	a
AAA = ASCII Adjust for Add	00110111	4		
AAS = ASCII Adjust for Subtract	00111111	4		
DAA = Decimal Adjust for Add	00100111	4		
DAS = Decimal Adjust for Subtract	00101111	4		
MUL = Multiply (Unsigned)				
Accumulator with Register/Memory	1111011w | mod100 r/m			
Multiplier—Byte		12–17/15–20	0/1	a,n
—Word		12–25/15–28*	0/1*	a,n
—Doubleword		12–41/17–46*	0/2*	a,n
IMUL = Integer Multiply (Signed)				
Accumulator with Register/Memory	1111011w | mod101 r/m			
Multiplier—Byte		12–17/15–20	0/1	a,n
—Word		12–25/15–28*	0/1*	a,n
—Doubleword		12–41/17–46*	0/2*	a,n
Register with Register/Memory	00001111 | 10101111 | mod reg r/m			
Multiplier—Byte		12–17/15–20	0/1	a,n
—Word		12–25/15–28*	0/1*	a,n
—Doubleword		12–41/17–46*	0/2*	a,n
Register/Memory with Immediate to Register	011010s1 | mod reg r/m | immediate data			
—Word		13–26/14–27*	0/1*	a,n
—Doubleword		13–42/16–45*	0/2*	a,n

5

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number Of Data Cycles	Notes
ARITHMETIC (Continued)				
DIV = Divide (Unsigned)				
Accumulator by Register/Memory	1 1 1 1 0 1 1 w | mod 1 1 0 r/m			
Divisor—Byte		14/17	0/1	a, o
—Word		22/25*	0/1*	a, o
—Doubleword		38/43*	0/2*	a, o
IDIV = Integer Divide (Signed)				
Accumulator by Register/Memory	1 1 1 1 0 1 1 w | mod 1 1 1 r/m			
Divisor—Byte		19/22	0/1	a, o
—Word		27/30*	0/1	a, o
—Doubleword		43/48*	0/2*	a, o
AAD = ASCII Adjust for Divide	1 1 0 1 0 1 0 1 | 0 0 0 0 1 0 1 0	19		
AAM = ASCII Adjust for Multiply	1 1 0 1 0 1 0 0 | 0 0 0 0 1 0 1 0	17		
CBW = Convert Byte to Word	1 0 0 1 1 0 0 0	3		
CWD = Convert Word to Double Word	1 0 0 1 1 0 0 1	2		
LOGIC				
Shift Rotate Instructions Not Through Carry (**ROL, ROR, SAL, SAR, SHL,** and **SHR**)				
Register/Memory by 1	1 1 0 1 0 0 0 w | mod TTT r/m	3/7**	0/2**	a
Register/Memory by CL	1 1 0 1 0 0 1 w | mod TTT r/m	3/7**	0/2**	a
Register/Memory by Immediate Count	1 1 0 0 0 0 0 w | mod TTT r/m | immed 8-bit data	3/7**	0/2**	a
Through Carry (**RCL** and **RCR**)				
Register/Memory by 1	1 1 0 1 0 0 0 w | mod TTT r/m	9/10**	0/2**	a
Register/Memory by CL	1 1 0 1 0 0 1 w | mod TTT r/m	9/10**	10/2**	a
Register/Memory by Immediate Count	1 1 0 0 0 0 0 w | mod TTT r/m | immed 8-bit data	9/10**	0/2**	a
	TTT Instruction 0 0 0 ROL 0 0 1 ROR 0 1 0 RCL 0 1 1 RCR 1 0 0 SHL/SAL 1 0 1 SHR 1 1 1 SAR			
SHLD = Shift Left Double				
Register/Memory by Immediate	0 0 0 0 1 1 1 1 | 1 0 1 0 0 1 0 0 | mod reg r/m | immed 8-bit data	3/7**	0/2**	
Register/Memory by CL	0 0 0 0 1 1 1 1 | 1 0 1 0 0 1 0 1 | mod reg r/m	3/7**	0/2**	
SHRD = Shift Right Double				
Register/Memory by Immediate	0 0 0 0 1 1 1 1 | 1 0 1 0 1 1 0 0 | mod reg r/m | immed 8-bit data	3/7**	0/2**	
Register/Memory by CL	0 0 0 0 1 1 1 1 | 1 0 1 0 1 1 0 1 | mod reg r/m	3/7**	0/2**	
AND = And				
Register to Register	0 0 1 0 0 0 d w | mod reg r/m	2		

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
LOGIC (Continued)				
Register to Memory	0010000w mod reg r/m	7**	2**	a
Memory to Register	0010001w mod reg r/m	6*	1*	a
Immediate to Register/Memory	1000000w mod 100 r/m immediate data	2/7**	0/2**	a
Immediate to Accumulator (Short Form)	0010010w immediate data	2		
TEST = And Function to Flags, No Result				
Register/Memory and Register	1000010w mod reg r/m	2/5*	0/1*	a
Immediate Data and Register/Memory	1111011w mod 000 r/m immediate data	2/5*	0/1*	a
Immediate Data and Accumulator (Short Form)	1010100w immediate data	2		
OR = Or				
Register to Register	000010dw mod reg r/m	2		
Register to Memory	0000100w mod reg r/m	7**	2**	a
Memory to Register	0000101w mod reg r/m	6*	1*	a
Immediate to Register/Memory	1000000w mod 001 r/m immediate data	2/7**	0/2**	a
Immediate to Accumulator (Short Form)	0000110w immediate data	2		
XOR = Exclusive Or				
Register to Register	001100dw mod reg r/m	2		
Register to Memory	0011000w mod reg r/m	7**	2**	a
Memory to Register	0011001w mod reg r/m	6*	1*	a
Immediate to Register/Memory	1000000w mod 110 r/m immediate data	2/7**	0/2**	a
Immediate to Accumulator (Short Form)	0011010w immediate data	2		
NOT = Invert Register/Memory	1111011w mod 010 r/m	2/6**	0/2**	a
STRING MANIPULATION				
CMPS = Compare Byte Word	1010011w	10*	2*	a
INS = Input Byte/Word from DX Port	0110110w	9** 29**	1** 1**	a,f,k a,f,l
LODS = Load Byte/Word to AL/AX/EAX	1010110w	5*	1*	a
MOVS = Move Byte Word	1010010w	7**	2**	a
OUTS = Output Byte/Word to DX Port	0110111w	8** 28**	1** 1**	a,f,k a,f,l
SCAS = Scan Byte Word	1010111w	7*	1*	a
STOS = Store Byte/Word from AL/AX/EX	1010101w	4*	1*	a
XLAT = Translate String	11010111	5*	1*	a
REPEATED STRING MANIPULATION Repeated by Count in CX or ECX				
REPE CMPS = Compare String (Find Non-Match)	11110011 1010011w	5 + 9n**	2n**	a

5

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
REPEATED STRING MANIPULATION (Continued)				
REPNE CMPS = Compare String				
(Find Match)	11110010 1010011w	5 + 9n**	2n**	a
REP INS = Input String	11110011 0110110w	7 + 6n*	1n*	a,f,k
		27 + 6n*	1n*	a,f,l
REP LODS = Load String	11110011 1010110w	5 + 6n*	1n*	a
REP MOVS = Move String	11110011 1010010w	7 + 4n**	2n**	a
REP OUTS = Output String	11110011 0110111w	6 + 5n*	1n*	a,f,k
		26 + 5n*	1n*	a,f,l
REPE SCAS = Scan String				
(Find Non-AL/AX/EAX)	11110011 1010111w	5 + 8n*	1n*	a
REPNE SCAS = Scan String				
(Find AL/AX/EAX)	11110010 1010111w	5 + 8n*	1n*	a
REP STOS = Store String	11110011 1010101w	5 + 5n*	1n*	a
BIT MANIPULATION				
BSF = Scan Bit Forward	00001111 10111100 mod reg r/m	10 + 3n**	2n**	a
BSR = Scan Bit Reverse	00001111 10111101 mod reg r/m	10 + 3n**	2n**	a
BT = Test Bit				
Register/Memory, Immediate	00001111 10111010 mod 100 r/m immed 8-bit data	3/6*	0/1*	a
Register/Memory, Register	00001111 10100011 mod reg r/m	3/12*	0/1*	a
BTC = Test Bit and Complement				
Register/Memory, Immediate	00001111 10111010 mod 111 r/m immed 8-bit data	6/8*	0/2*	a
Register/Memory, Register	00001111 10111011 mod reg r/m	6/13*	0/2*	a
BTR = Test Bit and Reset				
Register/Memory, Immediate	00001111 10111010 mod 110 r/m immed 8-bit data	6/8*	0/2*	a
Register/Memory, Register	00001111 10110011 mod reg r/m	6/13*	0/2*	a
BTS = Test Bit and Set				
Register/Memory, Immediate	00001111 10111010 mod 101 r/m immed 8-bit data	6/8*	0/2*	a
Register/Memory, Register	00001111 10101011 mod reg r/m	6/13*	0/2*	a
CONTROL TRANSFER				
CALL = Call				
Direct within Segment	11101000 full displacement	9 + m*	2	j
Register/Memory				
Indirect within Segment	11111111 mod 010 r/m	9 + m/12 + m	2/3	a, j
Direct Intersegment	10011010 unsigned full offset, selector	42 + m	9	c, d, j

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
CONTROL TRANSFER (Continued)				
(Direct Intersegment)				
Via Call Gate to Same Privilege Level		64 + m	13	a,c,d,j
Via Call Gate to Different Privilege Level, (No Parameters)		98 + m	13	a,c,d,j
Via Call Gate to Different Privilege Level, (x Parameters)		106 + 8x + m	13 + 4x	a,c,d,j
From 386 Task to 386 TSS		392	124	a,c,d,j
Indirect Intersegment	11111111 | mod011 r/m	46 + m	10	a,c,d,j
Via Call Gate to Same Privilege Level		68 + m	14	a,c,d,j
Via Call Gate to Different Privilege Level, (No Parameters)		102 + m	14	a,c,d,j
Via Call Gate to Different Privilege Level, (x Parameters)		110 + 8x + m	14 + 4x	a,c,d,j
From 386 Task to 386 TSS		399	130	a,c,d,j
JMP = Unconditional Jump				
Short	11101011 | 8-bit displacement	7 + m		j
Direct within Segment	11101001 | full displacement	7 + m		j
Register/Memory Indirect within Segment	11111111 | mod100 r/m	9 + m/14 + m	2/4	a,j
Direct Intersegment	11101010 | unsigned full offset, selector	37 + m	5	c,d,j
Via Call Gate to Same Privilege Level		53 + m	9	a,c,d,j
From 386 Task to 386 TSS		395	124	a,c,d,j
Indirect Intersegment	11111111 | mod101 r/m	37 + m	9	a,c,d,j
Via Call Gate to Same Privilege Level		59 + m	13	a,c,d,j
From 386 Task to 386 TSS		401	124	a,c,d,j

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
CONTROL TRANSFER (Continued)				
RET = Return from CALL:				
Within Segment	11000011	12 + m	2	a,j,p
Within Segment Adding Immediate to SP	11000010 | 16-bit displ	12 + m	2	a,j,p
Intersegment	11001011	36 + m	4	a,c,d,j,p
Intersegment Adding Immediate to SP	11001010 | 16-bit displ	36 + m	4	a,c,d,j,p
to Different Privilege Level				
Intersegment		80	4	c,d,j,p
Intersegment Adding Immediate to SP		80	4	c,d,j,p
CONDITIONAL JUMPS				
NOTE: Times Are Jump "Taken or Not Taken"				
JO = Jump on Overflow				
8-Bit Displacement	01110000 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10000000 | full displacement	7 + m or 3		j
JNO = Jump on Not Overflow				
8-Bit Displacement	01110001 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10000001 | full displacement	7 + m or 3		j
JB/JNAE = Jump on Below/Not Above or Equal				
8-Bit Displacement	01110010 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10000010 | full displacement	7 + m or 3		j
JNB/JAE = Jump on Not Below/Above or Equal				
8-Bit Displacement	01110011 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10000011 | full displacement	7 + m or 3		j
JE/JZ = Jump on Equal/Zero				
8-Bit Displacement	01110100 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10000100 | full displacement	7 + m or 3		j
JNE/JNZ = Jump on Not Equal/Not Zero				
8-Bit Displacement	01110101 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10000101 | full displacement	7 + m or 3		j
JBE/JNA = Jump on Below or Equal/Not Above				
8-Bit Displacement	01110110 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10000110 | full displacement	7 + m or 3		j
JNBE/JA = Jump on Not Below or Equal/Above				
8-Bit Displacement	01110111 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10000111 | full displacement	7 + m or 3		j
JS = Jump on Sign				
8-Bit Displacement	01111000 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10001000 | full displacement	7 + m or 3		j

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
CONDITIONAL JUMPS (Continued)				
JNS = Jump on Not Sign				
8-Bit Displacement	01111001 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10001001 | full displacement	7 + m or 3		j
JP/JPE = Jump on Parity/Parity Even				
8-Bit Displacement	01111010 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10001010 | full displacement	7 + m or 3		j
JNP/JPO = Jump on Not Parity/Parity Odd				
8-Bit Displacement	01111011 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10001011 | full displacement	7 + m or 3		j
JL/JNGE = Jump on Less/Not Greater or Equal				
8-Bit Displacement	01111100 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10001100 | full displacement	7 + m or 3		j
JNL/JGE = Jump on Not Less/Greater or Equal				
8-Bit Displacement	01111101 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10001101 | full displacement	7 + m or 3		j
JLE/JNG = Jump on Less or Equal/Not Greater				
8-Bit Displacement	01111110 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10001110 | full displacement	7 + m or 3		j
JNLE/JG = Jump on Not Less or Equal/Greater				
8-Bit Displacement	01111111 | 8-bit displ	7 + m or 3		j
Full Displacement	00001111 | 10001111 | full displacement	7 + m or 3		j
JECXZ = Jump on ECX Zero	11100011 | 8-bit displ	9 + m or 5		j
(Address Size Prefix Differentiates JCXZ from JECXZ)				
LOOP = Loop ECX Times	11100010 | 8-bit displ	11 + m		j
LOOPZ/LOOPE = Loop with Zero/Equal	11100001 | 8-bit displ	11 + m		j
LOOPNZ/LOOPNE = Loop While Not Zero	11100000 | 8-bit displ	11 + m		j
CONDITIONAL BYTE SET				
NOTE: Times Are Register/Memory				
SETO = Set Byte on Overflow				
To Register/Memory	00001111 | 10010000 | mod 000 r/m	4/5*	0/1*	a
SETNO = Set Byte on Not Overflow				
To Register/Memory	00001111 | 10010001 | mod 000 r/m	4/5*	0/1*	a
SETB/SETNAE = Set Byte on Below/Not Above or Equal				
To Register/Memory	00001111 | 10010010 | mod 000 r/m	4/5*	0/1*	a

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format			Clock Counts	Number of Data Cycles	Notes
CONDITIONAL BYTE SET (Continued)						
SETNB = Set Byte on Not Below/Above or Equal						
To Register/Memory	00001111	10010011	mod000 r/m	4/5*	0/1*	a
SETE/SETZ = Set Byte on Equal/Zero						
To Register/Memory	00001111	10010100	mod000 r/m	4/5*	0/1*	a
SETNE/SETNZ = Set Byte on Not Equal/Not Zero						
To Register/Memory	00001111	10010101	mod000 r/m	4/5*	0/1*	a
SETBE/SETNA = Set Byte on Below or Equal/Not Above						
To Register/Memory	00001111	10010110	mod000 r/m	4/5*	0/1*	a
SETNBE/SETA = Set Byte on Not Below or Equal/Above						
To Register/Memory	00001111	10010111	mod000 r/m	4/5*	0/1*	a
SETS = Set Byte on Sign						
To Register/Memory	00001111	10011000	mod000 r/m	4/5*	0/1*	a
SETNS = Set Byte on Not Sign						
To Register/Memory	00001111	10011001	mod000 r/m	4/5*	0/1*	a
SETP/SETPE = Set Byte on Parity/Parity Even						
To Register/Memory	00001111	10011010	mod000 r/m	4/5*	0/1*	a
SETNP/SETPO = Set Byte on Not Parity/Parity Odd						
To Register/Memory	00001111	10011011	mod000 r/m	4/5*	0/1*	a
SETL/SETNGE = Set Byte on Less/Not Greater or Equal						
To Register/Memory	00001111	10011100	mod000 r/m	4/5*	0/1*	a
SETNL/SETGE = Set Byte on Not Less/Greater or Equal						
To Register/Memory	00001111	01111101	mod000 r/m	4/5*	0/1*	a
SETLE/SETNG = Set Byte on Less or Equal/Not Greater						
To Register/Memory	00001111	10011110	mod000 r/m	4/5*	0/1*	a
SETNLE/SETG = Set Byte on Not Less or Equal/Greater						
To Register/Memory	00001111	10011111	mod000 r/m	4/5*	0/1*	a
ENTER = Enter Procedure	11001000	16-bit displacement, 8-bit level				
L = 0				10		a
L = 1				14	1	a
L > 1				17 + 8(n − 1)	4(n − 1)	a
LEAVE = Leave Procedure	11001001			6		a

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
INTERRUPT INSTRUCTIONS				
INT = Interrupt:				
Type Specified	11001101 | type			
Via Interrupt or Trap Gate to Same Privilege Level		71	14	c,d,j,p
Via Interrupt or Trap Gate to Different Privilege Level		111	14	c,d,j,p
From 386 Task to 386 TSS via Task Gate		467	140	c,d,j,p
Type 3	11001100			
Via Interrupt or Trap Gate to Same Privilege Level		71	14	c,d,j,p
Via Interrupt or Trap Gate to Different Privilege Level		111	14	c,d,j,p
From 386 Task to 386 TSS via Task Gate		308	136	c,d,j,p
INTO = Interrupt 4 if Overflow Flag Set	11001110			
If OF = 1:		3		
If OF = 0				
Via Interrupt or Trap Gate to Same Privilege Level		71	14	c,d,j,p
Via Interrupt or Trap Gate to Different Privilege Level		111	14	c,d,j,p
From 386 Task to 386 TSS via Task Gate		413	138	c,d,j,p

5

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number Of Data Cycles	Notes
INTERRUPT INSTRUCTIONS (Continued)				
Bound = Out of Range Interrupt 5 if Detect Value	01100010 | mod reg r/m			
if in Range		10	0	a,c,d,j,o,p
if Out of Range:				
Via Interrupt or Trap Gate to Same Privilege Level		71	14	c,d,j,p
Via Interrupt or Trap Gate to Different Privilege Level		111	14	c,d,j,p
From 386 Task to 386 TSS via Task Gate		398	138	c,d,j,p
INTERRUPT RETURN				
IRET = Interrupt Return	11001111			
To the Same Privilege Level (within Task)		42	5	a,c,d,j,p
To Different Privilege Level (within Task)		86	5	a,c,d,j,p
From 386 Task to 386 TSS		328	138	c,d,j,p
PROCESSOR CONTROL				
HLT = HALT	11110100	5		b
MOV = Move to and from Control/Debug/Test Registers				
CR0 from register	00001111 | 00100010 | 11 eee reg	10		b
Register from CR0	00001111 | 00100000 | 11 eee reg	6		b
DR0–3 from Register	00001111 | 00100011 | 11 eee reg	22		b
DR6–7 from Register	00001111 | 00100011 | 11 eee reg	16		b
Register from DR6–7	00001111 | 00100001 | 11 eee reg	14		b
Register from DR0–3	00001111 | 00100001 | 11 eee reg	22		b
NOP = No Operation	10010000	3		
WAIT = Wait until $\overline{\text{BUSY}}$ Pin is Negated	10011011	6		

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
PROCESSOR EXTENSION INSTRUCTIONS				
Processor Extension Escape	11011TTT | mod LLL r/m TTT and LLL bits are opcode information for coprocessor.	See 80387SX Data Sheet		a
PREFIX BYTES				
Address Size Prefix	01100111	0		
LOCK = Bus Lock Prefix	11110000	0		f
Operand Size Prefix	01100110	0		
Segment Override Prefix				
CS:	00101110	0		
DS:	00111110	0		
ES:	00100110	0		
FS:	01100100	0		
GS:	01100101	0		
SS:	00110110	0		
PROTECTION CONTROL				
ARPL = Adjust Requested Privilege Level				
From Register/Memory	01100011 | mod reg r/m	20/21**	2**	a
LAR = Load Access Rights				
From Register/Memory	00001111 | 00000010 | mod reg r/m	17/18*	1*	a,c,i,p
LGDT = Load Global Descriptor				
Table Register	00001111 | 00000001 | mod 010 r/m	13**	3*	a,e
LIDT = Load Interrupt Descriptor				
Table Register	00001111 | 00000001 | mod 011 r/m	13**	3*	a,e
LLDT = Load Local Descriptor				
Table Register to Register/Memory	00001111 | 00000000 | mod 010 r/m	24/28*	5*	a,c,e,p
LMSW = Load Machine Status Word				
From Register/Memory	00001111 | 00000001 | mod 110 r/m	10/13*	1*	a,e
LSL = Load Segment Limit				
From Register/Memory	00001111 | 00000011 | mod reg r/m			
Byte-Granular Limit		24/27*	2*	a,c,i,p
Page-Granular Limit		29/32*	2*	a,c,i,p
LTR = Load Task Register				
From Register/Memory	00001111 | 00000000 | mod 001 r/m	27/31*	4*	a,c,e,p
SGDT = Store Global Descriptor				
Table Register	00001111 | 00000001 | mod 000 r/m	11*	3*	a
SIDT = Store Interrupt Descriptor				
Table Register	00001111 | 00000001 | mod 001 r/m	11*	3*	a
SLDT = Store Local Descriptor Table Register				
To Register/Memory	00001111 | 00000000 | mod 000 r/m	2/2*	4*	a

5

Table 8.1. 80376 Instruction Set Clock Count Summary (Continued)

Instruction	Format	Clock Counts	Number of Data Cycles	Notes
PROTECTION CONTROL (Continued)				
SMSW = Store Machine Status Word	00001111 | 00000001 | mod 1 0 0 r/m	2/2*	1*	a, c
STR = Store Task Register				
To Register/Memory	00001111 | 00000000 | mod 0 0 1 r/m	2/2*	1*	a
VERR = Verify Read Accesss				
Register/Memory	00001111 | 00000000 | mod 1 0 0 r/m	10/11**	2**	a,c,i,p
VERW = Verify Write Accesss	00001111 | 00000000 | mod 1 0 1 r/m	15/16**	2**	a,c,i,p

NOTES:

a. Exception 13 fault (general violation) will occur if the memory operand in CS, DS, ES, FS or GS cannot be used due to either a segment limit violation or access rights violation. If a stack limit is violated, and exception 12 (stack segment limit violation or not present) occurs.

b. For segment load operations, the CPL, RPL and DPL must agree with the privilege rules to avoid an exception 13 fault (general protection violation). The segments's descriptor must indicate "present" or exception 11 (CS, DS, ES, FS, GS not present). If the SS register is loaded and a stack segment not present is detected, an exception 12 (stack segment limit violation or not present occurs).

c. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert $\overline{LOCK}$ to maintain descriptor integrity in multiprocessor systems.

d. JMP, CALL, INT, RET and IRET instructions referring to another code segment will cause an exception 13 (general protection violation) if an applicable privilege rule is volated.

e. An exception 13 fault occurs if CPL is greater than 0.

f. An exception 13 fault occurs if CPL is greater than IOPL.

g. The IF bit of the flag register is not updated if CPL is greater than IOPL. The IOPL field of the flag register is updated only if CPL = 0.

h. Any violation of privelege rules as applied to the selector operand does not cause a protection exception; rather, the zero flag is cleared.

i. If the coprocessor's memory operand violates a segment limit or segment access rights, an exception 13 fault (general protection exception) will occur before the ESC instruction is executed. An exception 12 fault (stack segment limit violation or no present) will occur if the stack limit is violated by the operand's starting address.

j. The destination of a JMP, CALL, INT, RET or IRET must be in the defined limit of a code segment or an exception 13 fault (general protection violation) will occur.

k. If CPL $\leq$ IOPL

l. If CPL $>$ IOPL

m. $\overline{LOCK}$ is automatically asserted, regardless of the presence or absence of the $\overline{LOCK}$ prefix.

n. The 80376 uses an early-out multiply algorithm. The actual number of clocks depends on the position of the most significant bit in the operand (multiplier). Clock counts given are minimum to maximum. To calculate actual clocks use the following formula:

Actual Clock = if m $<>$ 0 then max ($[\log_2 |m|]$, 3) + 9 clocks:
if m = 0 then 12 clocks (where m is the multiplier)

o. An exception may occur, depending on the value of the operand.

p. $\overline{LOCK}$ is asserted during descriptor table accesses.

8.2 INSTRUCTION ENCODING

Overview

All instruction encodings are subsets of the general instruction format shown in Figure 8.1. Instructions consist of one or two primary opcode bytes, possibly an address specifier consisting of the "mod r/m" byte and "scaled index" byte, a displacement if required, and an immediate data field if required.

Within the primary opcode or opcodes, smaller encoding fields may be defined. These fields vary according to the class of operation. The fields define such information as direction of the operation, size of the displacements, register encoding, or sign extension.

Almost all instructions referring to an operand in memory have an addressing mode byte following the primary opcode byte(s). This byte, the mod r/m byte, specifies the address mode to be used. Certain encodings of the mod r/m byte indicate a second addressing byte, the scale-index-base byte, follows the mod r/m byte to fully specify the addressing mode.

Addressing modes can include a displacement immediately following the mod r/m byte, or scaled index byte. If a displacement is present, the possible sizes are 8, 16 or 32 bits.

If the instruction specifies an immediate operand, the immediate operand follows any displacement bytes. The immediate operand, if specified, is always the last field of the instruction.

Figure 8.1 illustrates several of the fields that can appear in an instruction, such as the mod field and the r/m field, but the Figure does not show all fields. Several smaller fields also appear in certain instructions, sometimes within the opcode bytes themselves. Table 8.2 is a complete list of all fields appearing in the 80376 instruction set. Further ahead, following Table 8.2, are detailed tables for each field.

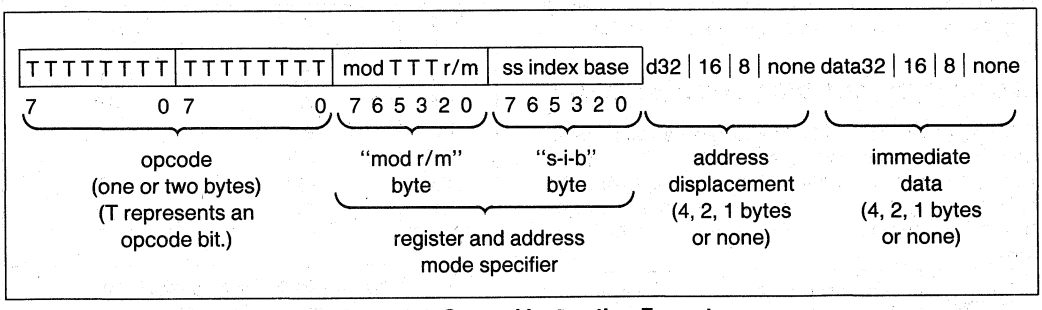


Figure 8.1. General Instruction Format

Table 8.2. Fields within 80376 Instructions

Field Name	Description	Number of Bits
w	Specifies if Data is Byte or Full Size (Full Size is either 16 or 32 Bits	1
d	Specifies Direction of Data Operation	1
s	Specifies if an Immediate Data Field Must be Sign-Extended	1
reg	General Register Specifier	3
mod r/m	Address Mode Specifier (Effective Address can be a General Register)	2 for mod; 3 for r/m
ss	Scale Factor for Scaled Index Address Mode	2
index	General Register to be used as Index Register	3
base	General Register to be used as Base Register	3
sreg2	Segment Register Specifier for CS, SS, DS, ES	2
sreg3	Segment Register Specifier for CS, SS, DS, ES, FS, GS	3
tttn	For Conditional Instructions, Specifies a Condition Asserted or a Condition Negated	4

Note: Table 8.1 shows encoding of individual instructions.

16-Bit Extensions of the Instruction Set

Two prefixes, the operand size prefix (66H) and the effective address size prefix (67H), allow overriding individually the default selection of operand size and effective address size. These prefixes may precede any opcode bytes and affect only the instruction they precede. If necessary, one or both of the prefixes may be placed before the opcode bytes. The presence of the operand size prefix (66H) and the effective address prefix will allow 16-bit data operation and 16-bit effective address calculations.

For instructions with more than one prefix, the order of prefixes is unimportant.

Unless specified otherwise, instructions with 8-bit and 16-bit operands do not affect the contents of the high-order bits of the extended registers.

Encoding of Instruction Fields

Within the instruction are several fields indicating register selection, addressing mode and so on.

ENCODING OF OPERAND LENGTH (w) FIELD

For any given instruction performing a data operation, the instruction will execute as a 32-bit operation. Within the constraints of the operation size, the w field encodes the operand size as either one byte or the full operation size, as shown in the table below.

w Field	Operand Size with 66H Prefix	Normal Operand Size
0	8 Bits	8 Bits
1	16 Bits	32 Bits

ENCODING OF THE GENERAL REGISTER (reg) FIELD

The general register is specified by the reg field, which may appear in the primary opcode bytes, or as the reg field of the "mod r/m" byte, or as the r/m field of the "mod r/m" byte.

Encoding of reg Field When w Field is not Present in Instruction

reg Field	Register Selected with 66H Prefix	Register Selected During 32-Bit Data Operations
000	AX	EAX
001	CX	ECX
010	DX	EDX
011	BX	EBX
100	SP	ESP
101	BP	EBP
110	SI	ESI
111	DI	EDI

Encoding of reg Field When w Field is Present in Instruction

Register Specified by reg Field with 66H Prefix		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	AX
001	CL	CX
010	DL	DX
011	BL	BX
100	AH	SP
101	CH	BP
110	DH	SI
111	BH	DI

Register Specified by reg Field without 66H Prefix		
reg	Function of w Field	
	(when w = 0)	(when w = 1)
000	AL	EAX
001	CL	ECX
010	DL	EDX
011	BL	EBX
100	AH	ESP
101	CH	EBP
110	DH	ESI
111	BH	EDI

ENCODING OF THE SEGMENT REGISTER (sreg) FIELD

The sreg field in certain instructions is a 2-bit field allowing one of the CS, DS, ES or SS segment registers to be specified. The sreg field in other instructions is a 3-bit field, allowing the FS and GS segment registers to be specified also.

2-Bit sreg2 Field

2-Bit sreg2 Field	Segment Register Selected
00	ES
01	CS
10	SS
11	DS

3-Bit sreg3 Field

3-Bit sreg3 Field	Segment Register Selected
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS
110	do not use
111	do not use

ENCODING OF ADDRESS MODE

Except for special instructions, such as PUSH or POP, where the addressing mode is pre-determined, the addressing mode for the current instruction is specified by addressing bytes following the primary opcode. The primary addressing byte is the "mod r/m" byte, and a second byte of addressing information, the "s-i-b" (scale-index-base) byte, can be specified.

The s-i-b byte (scale-index-base byte) is specified when using 32-bit addressing mode and the "mod r/m" byte has r/m = 100 and mod = 00, 01 or 10. When the sib byte is present, the 32-bit addressing mode is a function of the mod, ss, index, and base fields.

The primary addressing byte, the "mod r/m" byte, also contains three bits (shown as TTT in Figure 8.1) sometimes used as an extension of the primary opcode. The three bits, however, may also be used as a register field (reg).

When calculating an effective address, either 16-bit addressing or 32-bit addressing is used. 16-bit addressing uses 16-bit address components to calculate the effective address while 32-bit addressing uses 32-bit address components to calculate the effective address. When 16-bit addressing is used, the "mod r/m" byte is interpreted as a 16-bit addressing mode specifier. When 32-bit addressing is used, the "mod r/m" byte is interpreted as a 32-bit addressing mode specifier.

Tables on the following three pages define all encodings of all 16-bit addressing modes and 32-bit addressing modes.

Encoding of Normal Address Mode with "mod r/m" byte (no "s-i-b" byte present):

mod r/m	Effective Address
00 000	DS:[EAX]
00 001	DS:[ECX]
00 010	DS:[EDX]
00 011	DS:[EBX]
00 100	s-i-b is present
00 101	DS:d32
00 110	DS:[ESI]
00 111	DS:[EDI]
01 000	DS:[EAX + d8]
01 001	DS:[ECX + d8]
01 010	DS:[EDX + d8]
01 011	DS:[EBX + d8]
01 100	s-i-b is present
01 101	SS:[EBP + d8]
01 110	DS:[ESI + d8]
01 111	DS:[EDI + d8]

mod r/m	Effective Address
10 000	DS:[EAX + d32]
10 001	DS:[ECX + d32]
10 010	DS:[EDX + d32]
10 011	DS:[EBX + d32]
10 100	s-i-b is present
10 101	SS:[EBP + d32]
10 110	DS:[ESI + d32]
10 111	DS:[EDI + d32]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Register Specified by reg or r/m during Normal Data Operations:		
mod r/m	function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	EAX
11 001	CL	ECX
11 010	DL	EDX
11 011	BL	EBX
11 100	AH	ESP
11 101	CH	EBP
11 110	DH	ESI
11 111	BH	EDI

Register Specified by reg or r/m during 16-Bit Data Operations: (66H Prefix)		
mod r/m	function of w field	
	(when w = 0)	(when w = 1)
11 000	AL	AX
11 001	CL	CX
11 010	DL	DX
11 011	BL	BX
11 100	AH	SP
11 101	CH	BP
11 110	DH	SI
11 111	BH	DI

Encoding of 16-bit Address Mode with "mod r/m" Byte Using 67H Prefix

mod r/m	Effective Address
00 000	DS:[BX + SI]
00 001	DS:[BX + DI]
00 010	SS:[BP + SI]
00 011	SS:[BP + DI]
00 100	DS:[SI]
00 101	DS:[DI]
00 110	DS:d16
00 111	DS:[BX]
01 000	DS:[BX + SI + d8]
01 001	DS:[BX + DI + d8]
01 010	SS:[BP + SI + d8]
01 011	SS:[BP + DI + d8]
01 100	DS:[SI + d8]
01 101	DS:[DI + d8]
01 110	SS:[BP + d8]
01 111	DS:[BX + d8]

mod r/m	Effective Address
10 000	DS:[BX + SI + d16]
10 001	DS:[BX + DI + d16]
10 010	SS:[BP + SI + d16]
10 011	SS:[BP + DI + d16]
10 100	DS:[SI + d16]
10 101	DS:[DI + d16]
10 110	SS:[BP + d16]
10 111	DS:[BX + d16]
11 000	register—see below
11 001	register—see below
11 010	register—see below
11 011	register—see below
11 100	register—see below
11 101	register—see below
11 110	register—see below
11 111	register—see below

Encoding of 32-bit Address Mode ("mod r/m" byte and "s-i-b" byte present):

mod base	Effective Address
00 000	DS:[EAX + (scaled index)]
00 001	DS:[ECX + (scaled index)]
00 010	DS:[EDX + (scaled index)]
00 011	DS:[EBX + (scaled index)]
00 100	SS:[ESP + (scaled index)]
00 101	DS:[d32 + (scaled index)]
00 110	DS:[ESI + (scaled index)]
00 111	DS:[EDI + (scaled index)]
01 000	DS:[EAX + (scaled index) + d8]
01 001	DS:[ECX + (scaled index) + d8]
01 010	DS:[EDX + (scaled index) + d8]
01 011	DS:[EBX + (scaled index) + d8]
01 100	SS:[ESP + (scaled index) + d8]
01 101	SS:[EBP + (scaled index) + d8]
01 110	DS:[ESI + (scaled index) + d8]
01 111	DS:[EDI + (scaled index) + d8]
10 000	DS:[EAX + (scaled index) + d32]
10 001	DS:[ECX + (scaled index) + d32]
10 010	DS:[EDX + (scaled index) + d32]
10 011	DS:[EBX + (scaled index) + d32]
10 100	SS:[ESP + (scaled index) + d32]
10 101	SS:[EBP + (scaled index) + d32]
10 110	DS:[ESI + (scaled index) + d32]
10 111	DS:[EDI + (scaled index) + d32]

NOTE:
Mod field in "mod r/m" byte; ss, index, base fields in "s-i-b" byte.

ss	Scale Factor
00	x1
01	x2
10	x4
11	x8

index	Index Register
000	EAX
001	ECX
010	EDX
011	EBX
100	no index reg**
101	EBP
110	ESI
111	EDI

****IMPORTANT NOTE:**
When index field is 100, indicating "no index register," then ss field MUST equal 00. If index is 100 and ss does not equal 00, the effective address is undefined.

ENCODING OF OPERATION DIRECTION (d) FIELD

In many two-operand instructions the d field is present to indicate which operand is considered the source and which is the destination.

d	Direction of Operation
0	Register/Memory <- - Register "reg" Field Indicates Source Operand; "mod r/m" or "mod ss index base" Indicates Destination Operand
1	Register <- - Register/Memory "reg" Field Indicates Destination Operand; "mod r/m" or "mod ss index base" Indicates Source Operand

ENCODING OF SIGN-EXTEND (s) FIELD

The s field occurs primarily to instructions with immediate data fields. The s field has an effect only if the size of the immediate data is 8 bits and is being placed in a 16-bit or 32-bit destination.

s	Effect on Immediate Data8	Effect on Immediate Data 16|32
0	None	None
1	Sign-Extend Data8 to Fill 16-Bit or 32-Bit Destination	None

ENCODING OF CONDITIONAL TEST (tttn) FIELD

For the conditional instructions (conditional jumps and set on condition), tttn is encoded with n indicating to use the condition (n = 0) or its negation (n = 1), and tt giving the condition to test.

Mnemonic	Condition	tttn
O	Overflow	0000
NO	No Overflow	0001
B/NAE	Below/Not Above or Equal	0010
NB/AE	Not Below/Above or Equal	0011
E/Z	Equal/Zero	0100
NE/NZ	Not Equal/Not Zero	0101
BE/NA	Below or Equal/Not Above	0110
NBE/A	Not Below or Equal/Above	0111
S	Sign	1000
NS	Not Sign	1001
P/PE	Parity/Parity Even	1010
NP/PO	Not Parity/Parity Odd	1011
L/NGE	Less Than/Not Greater or Equal	1100
NL/GE	Not Less Than/Greater or Equal	1101
LE/NG	Less Than or Equal/Greater Than	1110
NLE/G	Not Less or Equal/Greater Than	1111

ENCODING OF CONTROL OR DEBUG REGISTER (eee) FIELD

For the loading and storing of the Control and Debug registers.

When Interpreted as Control Register Field

eee Code	Reg Name
000	CR0
010	Reserved
011	Reserved
Do not use any other encoding	

When Interpreted as Debug Register Field

eee Code	Reg Name
000	DR0
001	DR1
010	DR2
011	DR3
110	DR6
111	DR7
Do not use any other encoding	

9.0 REVISION HISTORY

The sections significantly revised since version -003 are:

Section 1.0 Added $\overline{FLT}$ pin.

Section 4.4 Added description of FLOAT operation and ONCE Mode. Figure 4.20 is new.

Section 4.6 Added revision identifier information for change to CHMOS IV manufacturing process.

Section 5.0 Both packages now specified for 0°C–115°C case temperature operation. Thermal resistance values changed.

Section 6.3 I_{CC} Max. specifications changed from 400 mA (cold) and 360 mA (hot) to 275 mA (cold, 16 MHz) and 305 mA (cold, 20 MHz).

Section 6.4 HLDA Valid Delay, t_{14}, min. changed from 6 ns to 4 ns. Added 20 MHz A.C. specifications in Table 6.5. Replaced Capacitive Derating Curves in Figures 6.8–6.10 to reflect new manufacturing process. Replaced I_{CC} vs. Frequency data (Figure 6.11) to reflect new specifications.

The sections significantly revised since version -002 are:

Section 1.0 Modified table 1.1. to list pins in alphabetical order.

The sections significantly revised since version -001 are:

Section 2.0 Figure 2.0 was updated to show the 16-bit registers SI, DI, BP and SP.

Section 2.1 Figure 2.2 was updated to show the correct bit polarity for bit 4 in the CR0 register.

Section 2.1 Tables 2.1 and 2.2 were updated to include additional information on the EFLAGs and CR0 registers.

Section 2.3 Figure 2.3 was updated to more accurately reflect the addressing mechanism of the 80376.

Section 2.6 In the subsection **Maskable Interrupt** a paragraph was added to describe the effect of interrupt gates on the IF EFLAGs bit.

Section 2.8 Table 2.7 was updated to reflect the correct power up condition of the CR0 register.

Section 2.10 Figure 2.6 was updated to show the correct bit positions of the BT, BS and BD bits in the DR6 register.

Section 3.0 Figure 3.1 was updated to clearly show the address calculation process.

Section 3.2 The subsection **DESCRIPTORS** was elaborated upon to clearly define the relationship between the linear address space and physical address space of the 80376.

Section 3.2 Figures 3.3 and 3.4 were updated to show the AVL bit field.

Section 3.3 The last sentence in the first paragraph of subsection **PROTECTION AND I/O PERMISSION BIT MAP** was deleted. This was an incorrect statement.

Section 4.1 In the Subsection **ADDRESS BUS ($\overline{BHE}$, $\overline{BLE}$, A_{23}–A_1** last sentence in the first paragraph was updated to reflect the numerics operand addresses as 8000FCH and 8000FEH. Because the 80376 sometimes does a double word I/O access a second access to 8000FEH can be seen.

Section 4.1 The Subsection **Hold Lantencies** was updated to describe how 32-bit and unaligned accesses are internally locked but do not assert the $\overline{LOCK}$ signal.

Section 4.2 Table 4.6 was updated to show the correct active data bits during a $\overline{BLE}$ assertion.

9.0 REVISION HISTORY (Continued)

Section 4.4 This section was updated to correctly reflect the pipelining of the address and status of the 80376 as opposed to "Address Pipelining" which occurs on processors such as the 80286.

Section 4.6 Table 4.7 was updated to show the correct Revision number, 05H.

Section 4.7 Table 4.8 was updated to show the numerics operand register 8000FEH. This address is seen when the 80376 does a DWORD operation to the port address 8000FCH.

Section 5.0 In the first paragraph the case temperatures were updated to reflect the 0°C–115°C for the ceramic package and 0°C–110°C for the plastic package.

Section 6.2 Table 6.2 was updated to reflect the Case Temperature under Bias specification of −65°C–120°C.

Section 6.4 Figure 6.8 vertical axis was updated to reflect "Output Valid Delay (ns)".

Section 6.4 Figure 6.11 was updated to show typical I_{CC} vs Frequency for the 80376.

Section 8.1 The clock counts and opcodes for various instructions were updated to their correct values.

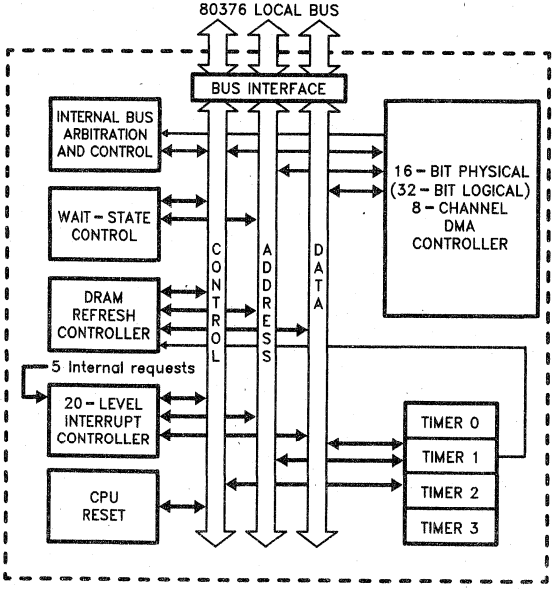
Section 8.2 The section **INSTRUCTION ENCODING** was appended to the data sheet.

82370
INTEGRATED SYSTEM PERIPHERAL

- **High Performance 32-Bit DMA Controller for 16-Bit Bus**
  - **— 16 MBytes/Sec Maximum Data Transfer Rate at 16 MHz**
  - **— 8 Independently Programmable Channels**
- **20-Source Interrupt Controller**
  - **— Individually Programmable Interrupt Vectors**
  - **— 15 External, 5 Internal Interrupts**
  - **— 82C59A Superset**
- **Four 16-Bit Programmable Interval Timers**
  - **— 82C54 Compatible**
- **Software Compatible to 82380**
- **Programmable Wait State Generator**
  - **— 0 to 15 Wait States Pipelined**
  - **— 1 to 16 Wait States Non-Pipelined**
- **DRAM Refresh Controller**
- **80376 Shutdown Detect and Reset Control**
  - **— Software/Hardware Reset**
- **High Speed CHMOS III Technology**
- **100-Pin Plastic Quad Flat-Pack Package and 132-Pin Pin Grid Array Package**
  (See Packaging Handbook Order #231369)
- **Optimized for Use with the 80376 Microprocessor**
  - **— Resides on Local Bus for Maximum Bus Bandwidth**

The 82370 is a multi-function support peripheral that integrates system functions necessary in an 80376 environment. It has eight channels of high performance 32-bit DMA (32-bit internal, 16-bit external) with the most efficient transfer rates possible on the 80376 bus. System support peripherals integrated into the 82370 provide Interrupt Control, Timers, Wait State generation, DRAM Refresh Control, and System Reset logic.

The 82370's DMA Controller can transfer data between devices of different data path widths using a single channel. Each DMA channel operates independently in any of several modes. Each channel has a temporary data storage register for handling non-aligned data without the need for external alignment logic.



290164-1

Internal Block Diagram

December 1990
Order Number: 290164-004

Table of Contents

CONTENTS

PAGE

Pin Descriptions 5-1318

1.0 Functional Overview 5-1323
1.1 82370 Architecture 5-1323
1.1.1 DMA (Direct Memory Access) Controller 5-1325
1.1.2 Programmable Interval Timers 5-1326
1.1.3 Interrupt Controller 5-1327
1.1.4 Wait State Generator 5-1328
1.1.5 DRAM (Dynamic RAM) Refresh Controller 5-1328
1.1.6 CPU Reset Function 5-1328
1.1.7 Register Map Relocation 5-1329
1.2 Host Interface 5-1329

2.0 80376 Host Interface 5-1329
2.1 Master and Slave Modes 5-1330
2.2 80376 Interface Signals 5-1330
2.2.1 Clock (CLK2) 5-1331
2.2.2 Data Bus (D_0–D_{15}) 5-1331
2.2.3 Address Bus (A_{23}–A_1) 5-1331
2.2.4 Byte Enable (BHE#, BLE#) 5-1331
2.2.5 Bus Cycle Definition Signals (D/C#, W/R#, M/IO#) 5-1332
2.2.6 Address Status (ADS#) 5-1332
2.2.7 Transfer Acknowledge (READY#) 5-1332
2.2.8 Next Address Request (NA#) 5-1332
2.2.9 Reset (RESET, CPURST) 5-1333
2.2.10 Interrupt Out (INT) 5-1334
2.3 82370 Bus Timing 5-1334
2.3.1 Address Pipelining 5-1334
2.3.2 Master Mode Bus Timing 5-1334
2.3.3 Slave Mode Bus Timing 5-1335

3.0 DMA Controller 5-1337
3.1 Functional Description 5-1337
3.2 Interface Signals 5-1339
3.2.1 DREQn and EDACK (0–2) 5-1340
3.2.2 HOLD and HLDA 5-1340
3.2.3 EOP# 5-1341
3.3 Modes of Operation 5-1341
3.3.1 Target/Requester Definition 5-1341
3.3.2 Buffer Transfer Processes 5-1342

5

CONTENTS

PAGE

3.0 DMA Controller (Continued)
3.3.3 Data Transfer Modes 5-1342
3.3.4 Channel Priority Arbitration 5-1346
3.3.5 Combining Priority Modes 5-1348
3.3.6 Bus Operation 5-1348
3.3.6.1 Fly-By Transfers 5-1348
3.3.6.2 Two-Cycle Transfers 5-1349
3.3.6.3 Data Path Width and Data Transfer Rate Considerations 5-1349
3.3.6.4 Read, Write, and Verify Cycles 5-1350
3.4 Bus Arbitration and Handshaking 5-1350
3.4.1 Synchronous and Asynchronous Sampling of DREQn and EOP# 5-1352
3.4.2 Arbitration of Cascaded Master Requests 5-1354
3.4.3 Arbitration of Refresh Requests 5-1355
3.5 DMA Controller Register Overview 5-1356
3.5.1 Control/Status Registers 5-1356
3.5.2 Channel Registers 5-1357
3.5.3 Temporary Registers 5-1358
3.6 DMA Controller Programming 5-1358
3.6.1 Buffer Processes 5-1359
3.6.1.1 Single Buffer Process 5-1359
3.6.1.2 Buffer Auto-Initialize Process 5-1359
3.6.1.3 Buffer Chaining Process 5-1360
3.6.2 Data Transfer Modes 5-1360
3.6.3 Cascaded Bus Masters 5-1360
3.6.4 Software Commands 5-1360
3.7 Register Definitions 5-1360
3.8 8237A Compatibility 5-1367

4.0 Programmable Interrupt Controller (PIC) 5-1367
4.1 Functional Description 5-1367
4.1.1 Internal Block Diagram 5-1367
4.1.2 Interrupt Controller Banks 5-1368
4.2 Interface Signals 5-1370
4.2.1 Interrupt Inputs 5-1370
4.2.2 Interrupt Output (INT) 5-1370
4.3 Bus Functional Description 5-1370
4.4 Modes of Operation 5-1371
4.4.1 End-of-Interrupt 5-1372
4.4.2 Interrupt Priorities 5-1372
4.4.2.1 Fully Nested Mode 5-1372
4.4.2.2 Automatic Rotation—Equal Priority Devices 5-1373
4.4.2.3 Specific Rotation—Specific Priority 5-1374
4.4.2.4 Interrupt Priority Mode Summary 5-1374

CONTENTS

PAGE

4.0 Programmable Interrupt Controller (Continued)
4.4.3 Interrupt Masking 5-1374
4.4.4 Edge or Level Interrupt Triggering 5-1375
4.4.5 Interrupt Cascading 5-1375
4.4.5.1 Special Fully Nested Mode 5-1376
4.4.6 Reading Interrupt Status 5-1376
4.4.6.1 Poll Command 5-1376
4.4.6.2 Reading Interrupt Registers 5-1376
4.5 Register Set Overview 5-1376
4.5.1 Initialization Command Words (ICW) 5-1378
4.5.2 Operation Control Words (OCW) 5-1378
4.5.3 Poll/Interrupt Request/In-Service Status Register 5-1379
4.5.4 Interrupt Mask Register (IMR) 5-1379
4.5.5 Vector Register (VR) 5-1379
4.6 Programming 5-1379
4.6.1 Initialization (ICW) 5-1379
4.6.2 Vector Registers (VR) 5-1380
4.6.3 Operation Control Words (OCW) 5-1381
4.6.3.1 Read Status and Poll Commands (OCW3) 5-1381
4.7 Register Bit Definition 5-1381
4.8 Register Operational Summary 5-1385

5.0 Programmable Interval Timer 5-1385
5.1 Functional Description 5-1385
5.1.1 Internal Architecture 5-1386
5.2 Interface Signals 5-1388
5.2.1 CLKIN 5-1388
5.2.2 TOUT1, TOUT2#, TOUT3# 5-1388
5.2.3 GATE 5-1388
5.3 Modes of Operation 5-1388
5.3.1 Mode 0—Interrupt on Terminal Count 5-1388
5.3.2 Mode 1—GATE Retriggerable One-Shot 5-1389
5.3.3 Mode 2—Rate Generator 5-1390
5.3.4 Mode 3—Square Wave Generator 5-1391
5.3.5 Mode 4—Initial Count Triggered Strobe 5-1393
5.3.6 Mode 5—GATE Retriggerable Strobe 5-1394
5.3.7 Operation Common to All Modes 5-1395
5.3.7.1 GATE 5-1395
5.3.7.2 Counter 5-1396
5.4 Register Set Overview 5-1396
5.4.1 Counter 0, 1, 2, 3 Registers 5-1396
5.4.2 Control Word Register I & II 5-1396

5

CONTENTS

PAGE

5.0 Programmable Interval Timer (Continued)
5.5 Programming 5-1397
5.5.1 Initialization 5-1397
5.5.2 Read Operation 5-1397
5.6 Register Bit Definitions 5-1399

6.0 Wait State Generator 5-1401
6.1 Functional Description 5-1401
6.2 Interface Signals 5-1401
6.2.1 READY# 5-1401
6.2.2 READYO# 5-1401
6.2.3 WSC (0–1) 5-1401
6.3 Bus Function 5-1402
6.3.1 Wait States in Non-Pipelined Cycle 5-1402
6.3.2 Wait States in Pipelined Cycles 5-1403
6.3.3 Extending and Early Terminating Bus Cycle 5-1403
6.4 Register Set Overview 5-1405
6.5 Programming 5-1405
6.6 Register Bit Definition 5-1405
6.7 Application Issues 5-1406
6.7.1 External 'READY' Control Logic 5-1406

7.0 DRAM Refresh Controller 5-1407
7.1 Functional Description 5-1407
7.2 Interface Signals 5-1407
7.2.1 TOUT1/REF# 5-1407
7.3 Bus Function 5-1407
7.3.1 Arbitration 5-1407
7.4 Modes of Operation 5-1408
7.4.1 Word Size and Refresh Address Counter 5-1408
7.5 Register Set Overview 5-1408
7.6 Programming 5-1409
7.7 Register Bit Definition 5-1409

8.0 Relocation Register, Address Decode and Chip-Select (CHPSEL#) 5-1409
8.1 Relocation Register 5-1409
8.1.1 I/O-Mapped 82370 5-1410
8.1.2 Memory-Mapped 82370 5-1410
8.2 Address Decoding 5-1410
8.3 Chip-Select (CHPSEL#) 5-1410

CONTENTS

PAGE

9.0 CPU Reset and Shutdown Detect 5-1411
9.1 Hardware Reset 5-1411
9.2 Software Reset 5-1411
9.3 Shutdown Detect 5-1411

10.0 Internal Control and Diagnostic Ports 5-1412
10.1 Internal Control Port 5-1412
10.2 Diagnostic Ports 5-1412

11.0 Internal Reserved I/O Ports 5-1412

12.0 Package Thermal Specifications 5-1413

13.0 Electrical Specifications 5-1414

Appendix A—Ports Listed by Address 5-1422

Appendix B—Ports Listed by Function 5-1426

Appendix C—System Notes 5-1430

Pin Descriptions

The 82370 provides all of the signals necessary to interface an 80376 host processor. It has a separate 24-bit address and 16-bit data bus. It also has a set of control signals to support operation as a bus master or a bus slave. Several special function signals exist on the 82370 for interfacing the system support peripherals to their respective system counterparts. Following are the definitions of the individual pins of the 82370. These brief descriptions are provided as a reference. Each signal is further defined within the sections which describe the associated 82370 function.

Symbol	Type	Name and Function
A_1–A_{23}	I/O	**ADDRESS BUS:** Outputs physical memory or port I/O addresses. See **Address Bus** (2.2.3) for additional information.
BHE# BLE#	I/O	**BYTE ENABLES:** Indicate which data bytes of the data bus take part in a bus cycle. See **Byte Enable** (2.2.4) for additional information.
D_0–D_{15}	I/O	**DATA BUS:** This is the 16-bit data bus. These pins are active outputs during interrupt acknowledges, during Slave accesses, and when the 82370 is in the Master Mode.
CLK2	I	**PROCESSOR CLOCK:** This pin must be connected to the processor's clock, CLK2. The 82370 monitors the phase of this clock in order to remain synchronized with the CPU. This clock drives all of the internal synchronous circuitry.
D/C#	I/O	**DATA/CONTROL:** D/C# is used to distinguish between CPU control cycles and DMA or CPU data access cycles. It is active as an output only in the Master Mode.
W/R#	I/O	**WRITE/READ:** W/R# is used to distinguish between write and read cycles. It is active as an output only in the Master Mode.
M/IO#	I/O	**MEMORY/IO:** M/IO# is used to distinguish between memory and IO accesses. It is active as an output only in the Master Mode.
ADS#	I/O	**ADDRESS STATUS:** This signal indicates presence of a valid address on the address bus. It is active as output only in the Master Mode. ADS# is active during the first T-state where addresses and control signals are valid.
NA#	I	**NEXT ADDRESS:** Asserted by a peripheral or memory to begin a pipelined address cycle. This pin is monitored only while the 82370 is in the Master Mode. In the Slave Mode, pipelining is determined by the current and past status of the ADS# and READY# signals.
HOLD	O	**HOLD REQUEST:** This is an active-high signal to the Bus Master to request control of the system bus. When control is granted, the Bus Master activates the hold acknowledge signal (HLDA).
HLDA	I	**HOLD ACKNOWLEDGE:** This input signal tells the DMA controller that the Bus Master has relinquished control of the system bus to the DMA controller.

Pin Descriptions (Continued)

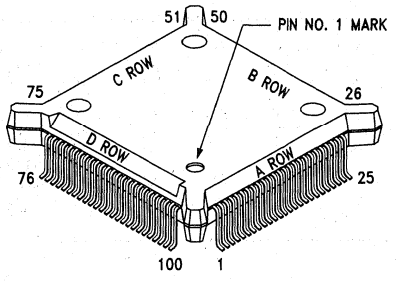
Symbol	Type	Name and Function
DREQ (0–3, 5–7)	I	**DMA REQUEST:** The DMA Request inputs monitor requests from peripherals requiring DMA service. Each of the eight DMA channels has one DREQ input. These active-high inputs are internally synchronized and prioritized. Upon request, channel 0 has the highest priority and channel 7 the lowest.
DREQ4/IRQ9#	I	**DMA/INTERRUPT REQUEST:** This is the DMA request input for channel 4. It is also connected to the interrupt controller via interrupt request 9. This internal connection is available for DMA channel 4 only. The interrupt input is active low and can be programmed as either edge or level triggered. Either function can be masked by the appropriate mask register. Priorities of the DMA channel and the interrupt request are not related but follow the rules of the individual controllers. Note that this pin has a weak internal pull-up. This causes the interrupt request to be inactive, but the DMA request will be active if there is no external connection made. Most applications will require that either one or the other of these functions be used, but not both. For this reason, it is advised that DMA channel 4 be used for transfers where a software request is more appropriate (such as memory-to-memory transfers). In such an application, DREQ4 can be masked by software, freeing IRQ9# for other purposes.
EOP#	I/O	**END OF PROCESS:** As an output, this signal indicates that the current Requester access is the last access of the currently operating DMA channel. It is activated when Terminal Count is reached. As an input, it signals the DMA channel to terminate the current buffer and proceed to the next buffer, if one is available. This signal may be programmed as an asynchronous or synchronous input. EOP# must be connected to a pull-up resistor. This will prevent erroneous external requests for termination of a DMA process.
EDACK (0–2)	O	**ENCODED DMA ACKNOWLEDGE:** These signals contain the encoded acknowledgment of a request for DMA service by a peripheral. The binary code formed by the three signals indicates which channel is active. Channel 4 does not have a DMA acknowledge. The inactive state is indicated by the code 100. During a Requester access, EDACK presents the code for the active DMA channel. During a Target access, EDACK presents the inactive code 100.
IRQ (11–23)#	I	**INTERRUPT REQUEST:** These are active low interrupt request inputs. The inputs can be programmed to be edge or level sensitive. Interrupt priorities are programmable as either fixed or rotating. These inputs have weak internal pull-up resistors. Unused interrupt request inputs should be tied inactive externally.
INT	O	**INTERRUPT OUT:** INT signals that an interrupt request is pending.
CLKIN	I	**TIMER CLOCK INPUT:** This is the clock input signal to all of the 82370's programmable timers. It is independent of the system clock input (CLK2).
TOUT1/REF#	O	**TIMER 1 OUTPUT/REFRESH:** This pin is software programmable as either the direct output of Timer 1, or as the indicator of a refresh cycle in progress. As REF#, this signal is active during the memory read cycle which occurs during refresh.

Pin Descriptions (Continued)

Symbol	Type	Name and Function
TOUT2#/IRQ3#	I/O	**TIMER 2 OUTPUT/INTERRUPT REQUEST:** This is the inverted output of Timer 2. It is also connected directly to interrupt request 3. External hardware can use IRQ3# if Timer 2 is programmed as OUT = 0 (TOUT2# = 1).
TOUT3#	O	**TIMER 3 OUTPUT:** This is the inverted output of Timer 3.
READY#	I	**READY INPUT:** This active-low input indicates to the 82370 that the current bus cycle is complete. READY is sampled by the 82370 both while it is in the Master Mode, and while it is in the Slave Mode.
WSC (0-1)	I	**WAIT STATE CONTROL:** WSC0 and WSC1 are inputs used by the Wait-State Generator to determine the number of wait states required by the currently accessed memory or I/O. The binary code on these pins, combined with the M/IO# signal, selects an internal register in which a wait-state count is stored. The combination WSC = 11 disables the wait-state generator.
READYO#	O	**READY OUTPUT:** This is the synchronized output of the wait-state generator. It is also valid during CPU accesses to the 82370 in the Slave Mode when the 82370 requires wait states. READYO# should feed directly the processor's READY# input.
RESET	I	**RESET:** This synchronous input serves to initialize the state of the 82370 and provides basis for the CPURST output. RESET must be held active for at least 15 CLK2 cycles in order to guarantee the state of the 82370. After Reset, the 82370 is in the Slave Mode with all outputs except timers and interrupts in their inactive states. The state of the timers and interrupt controller must be initialized through software. This input must be active for the entire time required by the host processor to guarantee proper reset.
CHPSEL#	O	**CHIP SELECT:** This pin is driven active whenever the 82370 is addressed in a slave bus read or write cycle. It is also active during interrupt acknowledge cycles when the 82370 is driving the Data Bus. It can be used to control the local bus transceivers to prevent contention with the system bus.
CPURST	O	**CPU RESET:** CPURST provides a synchronized reset signal for the CPU. It is activated in the event of a software reset command, a processor shut-down detect, or a hardware reset via the RESET pin. The 82370 holds CPURST active for 62 clocks in response to either a software reset command or a shut-down detection. Otherwise CPURST reflects the RESET input.
V_{CC}		**POWER:** +5V input power.
V_{SS}		Ground Reference.

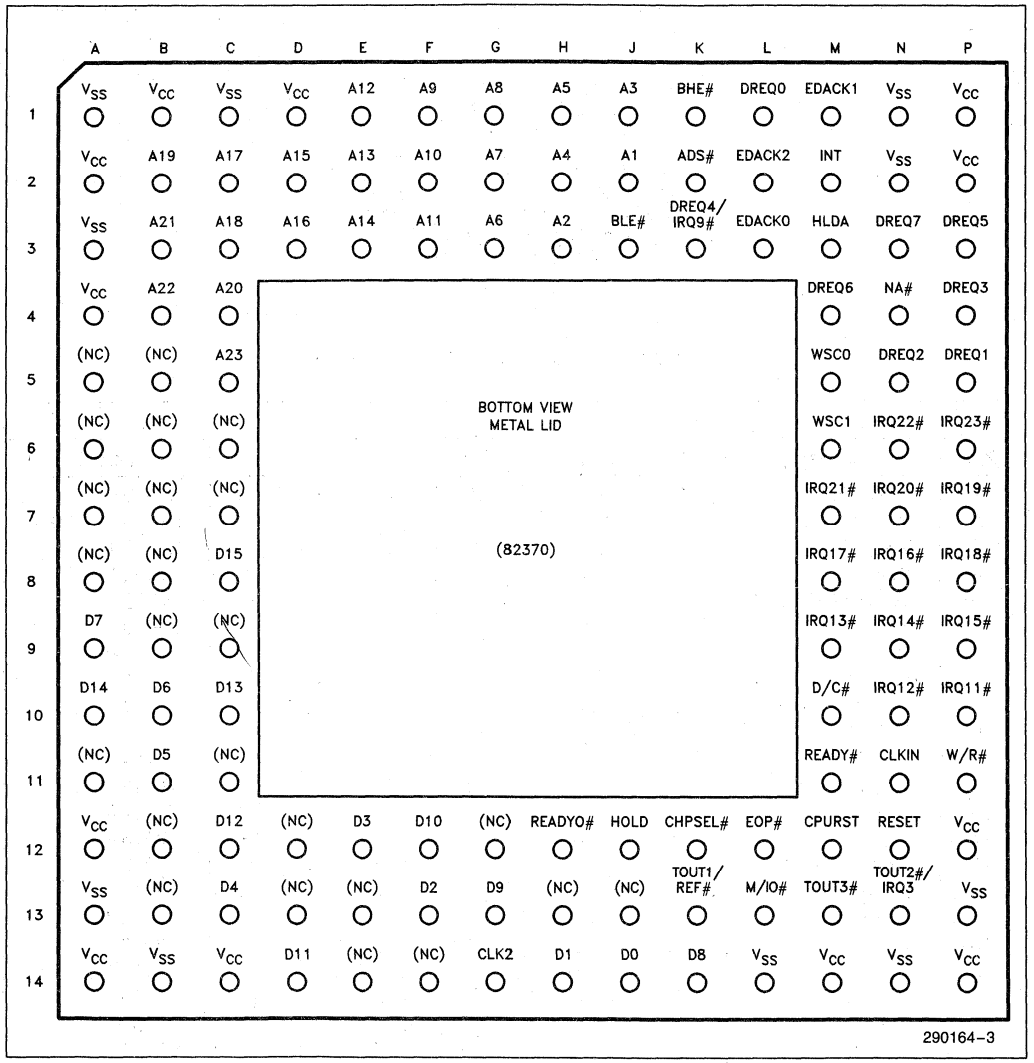
Table 1. Wait-State Select Inputs

Port Address	Wait-State Registers D7 D4	Wait-State Registers D3 D0	Select Inputs WSC1	Select Inputs WSC0
72H	MEMORY 0	I/O 0	0	0
73H	MEMORY 1	I/O 1	0	1
74H	MEMORY 2	I/O 2	1	0
	DISABLED		1	1
M/IO#	1	0		



100 Pin Quad Flat-Pack Pin Out (Top View)

A Row		B Row		C Row		D Row	
Pin	**Label**	**Pin**	**Label**	**Pin**	**Label**	**Pin**	**Label**
1	CPURST	26	V_{CC}	51	A_{11}	76	DREQ5
2	INT	27	D_{11}	52	A_{10}	77	DREQ4/IRQ9#
3	V_{CC}	28	D_4	53	A_9	78	DREQ3
4	V_{SS}	29	D_{12}	54	A_8	79	DREQ2
5	TOUT2#/IRQ3#	30	D_5	55	A_7	80	DREQ1
6	TOUT3#	31	D_{13}	56	A_6	81	DREQ0
7	D/C#	32	D_6	57	A_5	82	IRQ23#
8	V_{CC}	33	V_{SS}	58	V_{CC}	83	IRQ22#
9	W/R#	34	D_{14}	59	A_4	84	IRQ21#
10	M/IO#	35	D_7	60	A_3	85	IRQ20#
11	HOLD	36	D_{15}	61	A_2	86	IRQ19#
12	TOUT1/REF#	37	A_{23}	62	A_1	87	IRQ18#
13	CLK2	38	A_{22}	63	V_{SS}	88	IRQ17#
14	V_{SS}	39	A_{21}	64	BLE#	89	IRQ16#
15	READYO#	40	A_{20}	65	BHE#	90	IRQ15#
16	EOP#	41	A_{19}	66	V_{SS}	91	IRQ14#
17	CHPSEL#	42	A_{18}	67	ADS#	92	IRQ13#
18	V_{CC}	43	V_{CC}	68	V_{CC}	93	IRQ12#
19	D_0	44	A_{17}	69	EDACK2	94	IRQ11#
20	D_8	45	A_{16}	70	EDACK1	95	CLKIN
21	D_1	46	A_{15}	71	EDACK0	96	WSC0
22	D_9	47	A_{14}	72	HLDA	97	WSC1
23	D_2	48	V_{SS}	73	DREQ7	98	RESET
24	D_{10}	49	A_{13}	74	DREQ6	99	READY#
25	D_3	50	A_{12}	75	NA#	100	V_{SS}



82370 PGA Pinout

Pin	Label	Pin	Label	Pin	Label	Pin	Label
G14	CLK2	D14	D_{11}	L1	DREQ0	A2	V_{CC}
N12	RESET	F12	D_{10}	P6	IRQ23#	P2	V_{CC}
M12	CPURST	G13	D_9	N6	IRQ22#	A4	V_{CC}
C5	A_{23}	K14	D_8	M7	IRQ21#	A12	V_{CC}
B4	A_{22}	A9	D_7	N7	IRQ20#	P12	V_{CC}
B3	A_{21}	B10	D_6	P7	IRQ19#	A14	V_{CC}
C4	A_{20}	B11	D_5	P8	IRQ18#	C14	V_{CC}
B2	A_{19}	C13	D_4	M8	IRQ17#	M14	V_{CC}
C3	A_{18}	E12	D_3	N8	IRQ16#	P14	V_{CC}
C2	A_{17}	F13	D_2	P9	IRQ15#	A5	NC
D3	A_{16}	H14	D_1	N9	IRQ14#	B5	NC
D2	A_{15}	J14	D_0	M9	IRQ13#	A6	NC
E3	A_{14}	P11	W/R#	N10	IRQ12#	B6	NC
E2	A_{13}	L13	M/IO#	P10	IRQ11#	C6	NC
E1	A_{12}	K2	ADS#	M5	WSC0	A7	NC
F3	A_{11}	M10	D/C#	M6	WSC1	B7	NC
F2	A_{10}	N4	NA#	M13	TOUT3#	C7	NC
F1	A_9	M11	READY#	N13	TOUT2#/IRQ3#	A8	NC
G1	A_8	H12	READYO#	K13	TOUT1/REF#	B8	NC
G2	A_7	J12	HOLD	N11	CLKIN	B9	NC
G3	A_6	M3	HLDA	A1	V_{SS}	C9	NC
H1	A_5	M2	INT	C1	V_{SS}	A11	NC
H2	A_4	L12	EOP#	N1	V_{SS}	B12	NC
J1	A_3	L2	EDACK2	N2	V_{SS}	C11	NC
H3	A_2	M1	EDACK1	A3	V_{SS}	D12	NC
J2	A_1	L3	EDACK0	A13	V_{SS}	G12	NC
J3	BLE#	N3	DREQ7	P13	V_{SS}	B13	NC
K1	BHE#	M4	DREQ6	B14	V_{SS}	D13	NC
K12	CHPSEL#	P3	DREQ5	L14	V_{SS}	E13	NC
C8	D_{15}	K3	DREQ4/IRQ9#	N14	V_{SS}	H13	NC
A10	D_{14}	P4	DREQ3	B1	V_{CC}	J13	NC
C10	D_{13}	N5	DREQ2	D1	V_{CC}	E14	NC
C12	D_{12}	P5	DREQ1	P1	V_{CC}	F14	NC

5

1.0 FUNCTIONAL OVERVIEW

The 82370 contains several independent functional modules. The following is a brief discussion of the components and features of the 82370. Each module has a corresponding detailed section later in this data sheet. Those sections should be referred to for design and programming information.

1.1 82370 Architecture

The 82370 is comprised of several computer system functions that are normally found in separate LSI and VLSI components. These include: a high-performance, eight-channel, 32-bit Direct Memory Access Controller; a 20-level Programmable Interrupt Controller which is a superset of the 82C59A; four 16-bit Programmable Interval Timers which are functionally equivalent to the 82C54 timers; a DRAM Refresh Controller; a Programmable Wait State Generator; and system reset logic. The interface to the 82370 is optimized for high-performance operation with the 80376 microprocessor.

The 82370 operates directly on the 80376 bus. In the Slave Mode, it monitors the state of the processor at all times and acts or idles according to the commands of the host. It monitors the address pipeline status and generates the programmed number of wait states for the device being accessed. The 82370 also has logic to the reset of the 80376 via hardware or software reset requests and processor shutdown status.

After a system reset, the 82370 is in the Slave Mode. It appears to the system as an I/O device. It becomes a bus master when it is performing DMA transfers.

To maintain compatibility with existing software, the registers within the 82370 are accessed as bytes. If the internal logic of the 82370 requires a delay before another access by the processor, wait states are automatically inserted into the access cycle. This allows the programmer to write initialization routines, etc. without regard to hardware recovery times.

Figure 1-1 shows the basic architectural components of the 82370. The following sections briefly discuss the architecture and function of each of the distinct sections of the 82370.

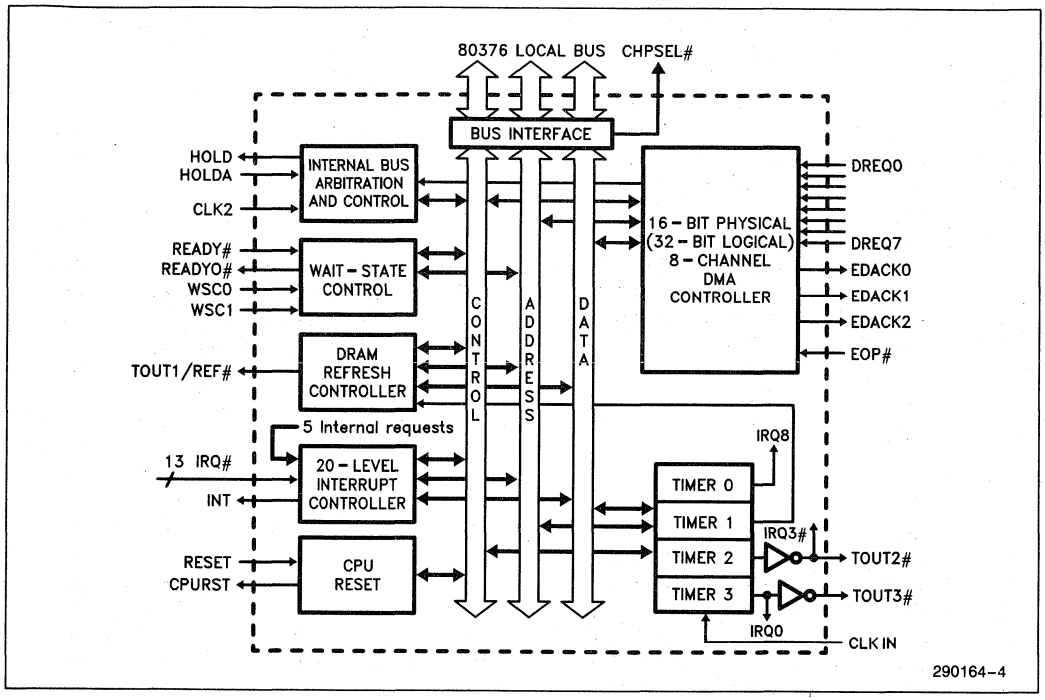


Figure 1-1. Architecture of the 82370

1.1.1 DMA CONTROLLER

The 82370 contains a high-performance, 8-channel DMA Controller. It provides a 32-bit internal data path. Through its 16-bit external physical data bus, it is capable of transferring data in any combination of bytes, words and double-words. The addresses of both source and destination can be independently incremented, decremented or held constant, and cover the entire 16-bit physical address space of the 80376. It can disassemble and assemble non-aligned data via a 32-bit internal temporary data storage register. Data transferred between devices of different data path widths can also be assembled and disassembled using the internal temporary data storage register. The DMA Controller can also transfer aligned data between I/O and memory on the fly, allowing data transfer rates up to 16 megabytes per second for an 82370 operating at 16 MHz. Figure 1-2 illustrates the functional components of the DMA Controller.

There are twenty-four general status and command registers in the 82370 DMA Controller. Through these registers any of the channels may be programmed into any of the possible modes. The operating modes of any one channel are independent of the operation of the other channels.

Each channel has three programmable registers which determine the location and amount of data to be transferred:

- Byte Count Register — Number of bytes to transfer. (24-bits)
- Requester Register — Byte Address of memory or peripheral which is requesting DMA service. (24-bits)
- Target Register — Byte Address of peripheral or memory which will be accessed. (24-bits)

There are also port addresses which, when accessed, cause the 82370 to perform specific functions. The actual data written doesn't matter, the act of writing to the specific address causes the command to be executed. The commands which operate in this mode are: Master Clear, Clear Terminal Count Interrupt Request, Clear Mask Register, and Clear Byte Pointer Flip-Flop.

DMA transfers can be done between all combinations of memory and I/O; memory-to-memory, memory-to-I/O, I/O-to-memory, and I/O-to-I/O. DMA service can be requested through software and/or hardware. Hardware DMA acknowledge signals are available for all channels (except channel 4) through an encoded 3-bit DMA acknowledge bus (EDACK0–2).

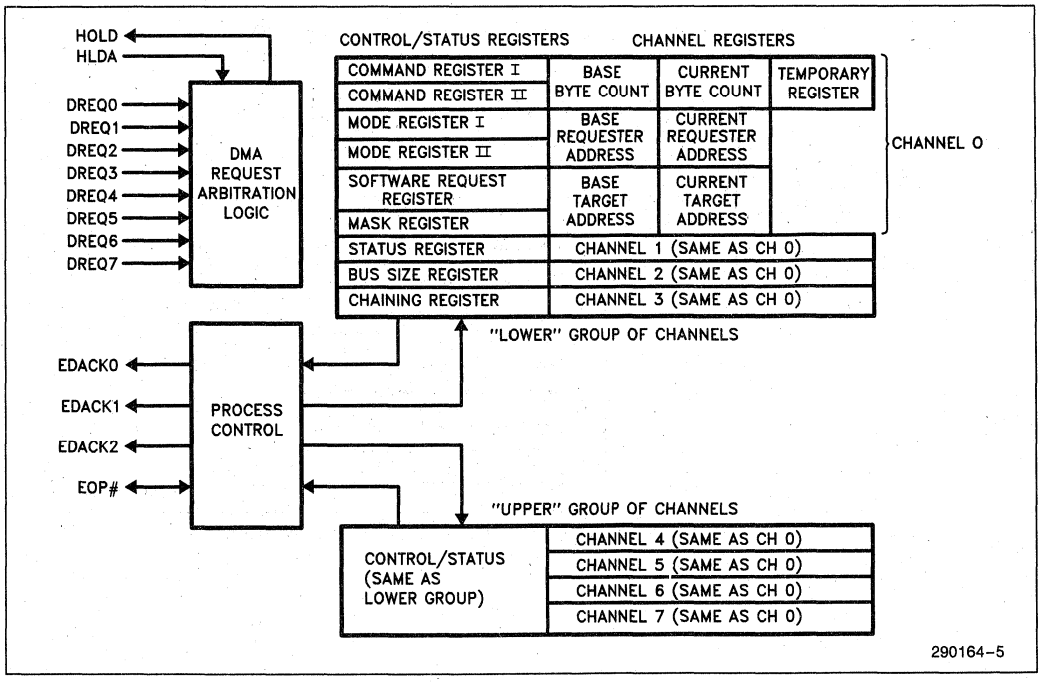


Figure 1-2. 82370 DMA Controller

The 82370 DMA Controller transfers blocks of data (buffers) in three modes: Single Buffer, Buffer Auto-Initialize, and Buffer Chaining. In the Single Buffer Process, the 82370 DMA Controller is programmed to transfer one particular block of data. Successive transfers then require reprogramming of the DMA channel. Single Buffer transfers are useful in systems where it is known at the time the transfer begins what quantity of data is to be transferred, and there is a contiguous block of data area available.

The Buffer Auto-Initialize Process allows the same data area to be used for successive DMA transfers without having to reprogram the channel.

The Buffer Chaining Process allows a program to specify a list of buffer transfers to be executed. The 82370 DMA Controller, through interrupt routines, is reprogrammed from the list. The channel is reprogrammed for a new buffer before the current buffer transfer is complete. This pipelining of the channel programming process allows the system to allocate non-contiguous blocks of data storage space, and transfer all of the data with one DMA process. The buffers that make up the chain do not have to be in contiguous locations.

Channel priority can be fixed or rotating. Fixed priority allows the programmer to define the priority of DMA channels based on hardware or other fixed parameters. Rotating priority is used to provide peripherals access to the bus on a shared basis.

With fixed priority, the programmer can set any channel to have the current lowest priority. This allows the user to reset or manually rotate the priority schedule without reprogramming the command registers.

1.1.2 PROGRAMMABLE INTERVAL TIMERS

Four 16-bit programmable interval timers reside within the 82370. These timers are identical in function to the timers in the 82C54 Programmable Interval Timer. All four of the timers share a common clock input which can be independent of the system clock. The timers are capable of operating in six different modes. In all of the modes, the current count can be latched and read by the 80376 at any time, making these very versatile event timers. Figure 1-3 shows the functional components of the Programmable Interval Timers.

The outputs of the timers are directed to key system functions, making system design simpler. Timer 0 is routed directly to an interrupt input and is not available externally. This timer would typically be used to generate time-keeping interrupts.

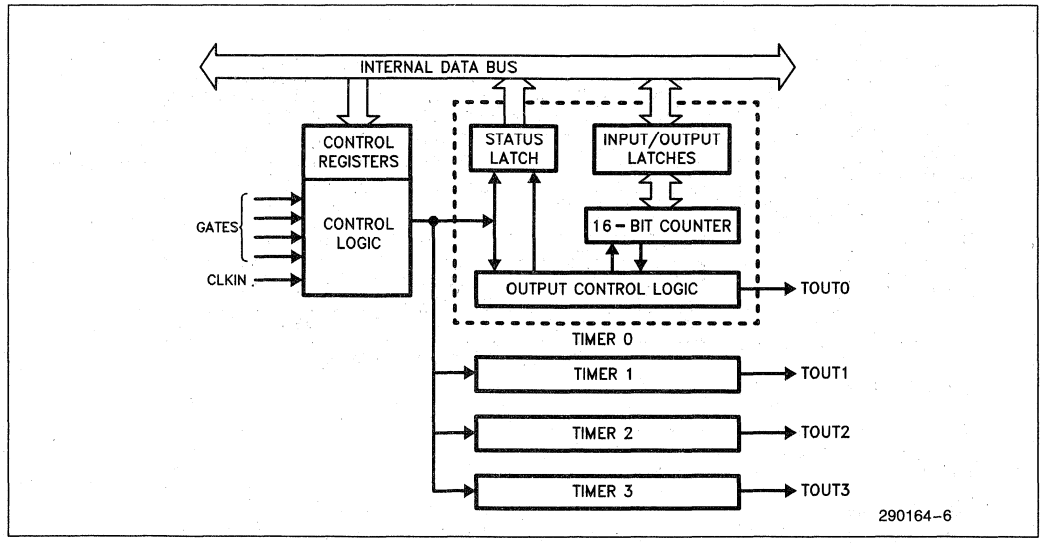


Figure 1-3. Programmable Interval Timers—Block Diagram

Timers 1 and 2 have outputs which are available for general timer/counter purposes as well as special functions. Timer 1 is routed to the refresh control logic to provide refresh timing. Timer 2 is connected to an interrupt request input to provide other timer functions. Timer 3 is a general purpose timer/counter whose output is available to external hardware. It is also connected internally to the interrupt request which defaults to the highest priority (IRQ0).

1.1.3 INTERRUPT CONTROLLER

The 82370 has the equivalent of three enhanced 82C59A Programmable Interrupt Controllers. These controllers can all be operated in the Master Mode, but the priority is always as if they were cascaded. There are 15 interrupt request inputs provided for the user, all of which can be inputs from external slave interrupt controllers. Cascading 82C59As to these request inputs allows a possible total of 120 external interrupt requests. Figure 1-4 is a block diagram of the 82370 Interrupt Controller.

Each of the interrupt request inputs can be individually programmed with its own interrupt vector, allowing more flexibility in interrupt vector mapping than was available with the 82C59A. An interrupt is provided to alert the system that an attempt is being made to program the vectors in the method of the 82C59A. This provides compatibility of existing software that used the 82C59A or 8259A with new designs using the 82370.

In the event of an unrequested or otherwise erroneous interrupt acknowledge cycle, the 82370 Interrupt Controller issues a default vector. This vector, programmed by the system software, will alert the system of unsolicited interrupts of the 80376.

The functions of the 82370 Interrupt Controller are identical to the 82C59A, except in regards to programming the interrupt vectors as mentioned above. Interrupt request inputs are programmable as either edge or level triggered and are software maskable. Priority can be either fixed or rotating and interrupt requests can be nested.

Enhancements are added to the 82370 for cascading external interrupt controllers. Master to Slave handshaking takes place on the data bus, instead of dedicated cascade lines.

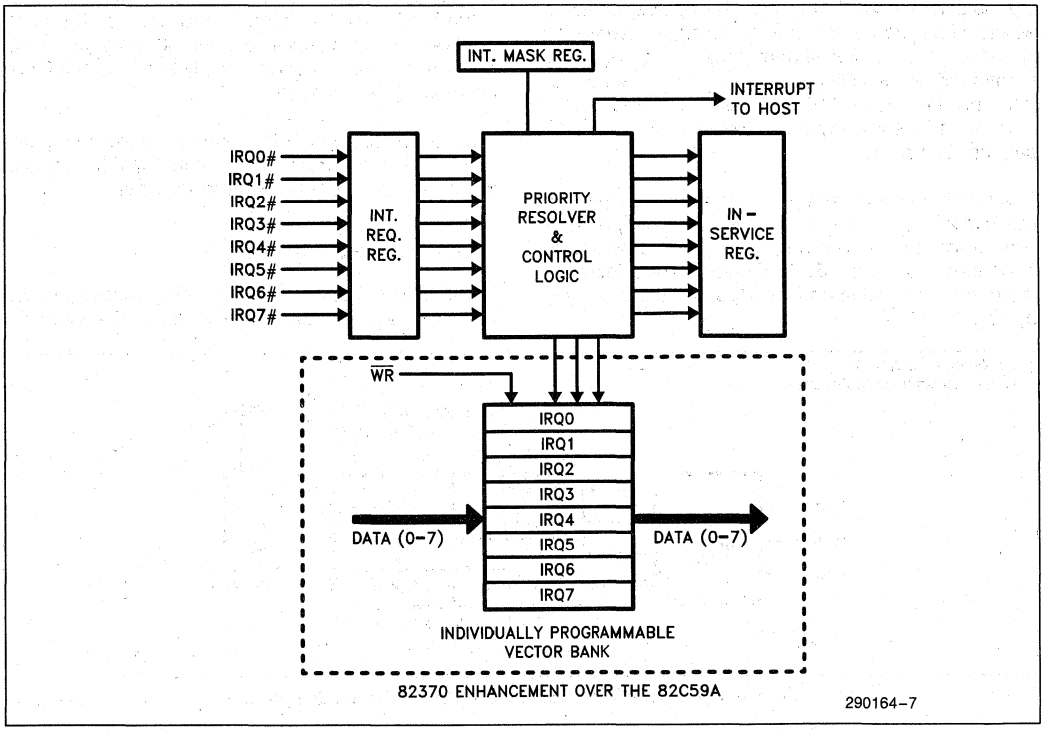


Figure 1-4. 82370 Interrupt Controller—Block Diagram

1.1.4 WAIT STATE GENERATOR

The Wait State Generator is a programmable READY generation circuit for the 80376 bus. A peripheral requiring wait states can request the Wait State Generator to hold the processor's READY input inactive for a predetermined number of bus states. Six different wait state counts can be programmed into the Wait State Generator by software; three for memory accesses and three for I/O accesses. A block diagram of the 82370 Wait State Generator is shown in Figure 1-5.

The peripheral being accessed selects the required wait state count by placing a code on a 2-bit wait state select bus. This code along with the M/IO# signal from the bus master is used to select one of six internal 4-bit wait state registers which has been programmed with the desired number of wait states. From zero to fifteen wait states can be programmed into the wait state registers. The Wait State generator tracks the state of the processor or current bus master at all times, regardless of which device is the current bus master and regardless of whether or not the wait state generator is currently active.

The 82370 Wait State Generator is disabled by making the select inputs both high. This allows hardware which is intelligent enough to generate its own ready signal to be accessed without penalty. As previously mentioned, deselecting the Wait State Generator does not disable its ability to determine the proper number of wait states due to pipeline status in subsequent bus cycles.

The number of wait states inserted into a pipelined bus cycle is the value in the selected wait state register. If the bus master is operating in the non-pipelined mode, the Wait State Generator will increase the number of wait states inserted into the bus cycle by one.

On reset, the Wait State Generator's registers are loaded with the value FFH, giving the maximum number of wait states for any access in which the wait state select inputs are active.

1.1.5 DRAM REFRESH CONTROLLER

The 82370 DRAM Refresh Controller consists of a 24-bit refresh address counter and bus arbitration logic. The output of Timer 1 is used to periodically request a refresh cycle. When the controller receives the request, it requests access to the system bus through the HOLD signal. When bus control is acknowledged by the processor or current bus master, the refresh controller executes a memory read operation at the address currently in the Refresh Address Register. At the same time, it activates a refresh signal (REF#) that the memory uses to force a refresh instead of a normal read. Control of the bus is transferred to the processor at the completion of this cycle. Typically a refresh cycle will take six clock cycles to execute on an 80376 bus.

The 82370 DRAM Refresh Controller has the highest priority when requesting bus access and will interrupt any active DMA process. This allows large blocks of data to be moved by the DMA controller without affecting the refresh function. Also the DMA controller is not required to completely relinquish the bus, the refresh controller simply steals a bus cycle between DMA accesses.

The amount by which the refresh address is incremented is programmable to allow for different bus widths and memory bank arrangements.

1.1.6 CPU RESET FUNCTION

The 82370 contains a special reset function which can respond to hardware reset signals as well as a

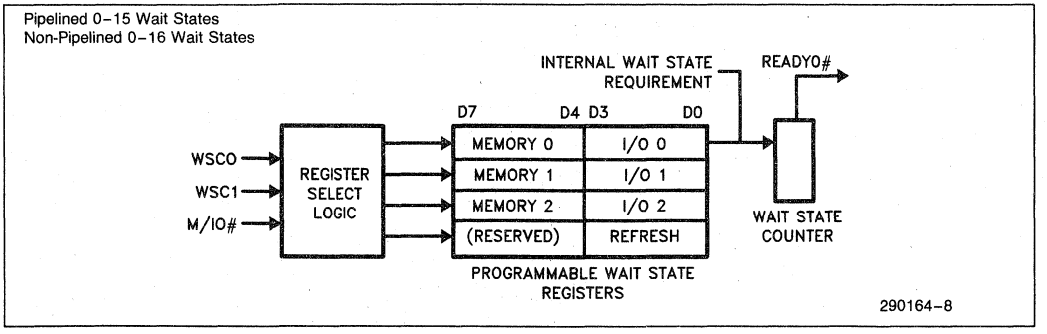


Figure 1-5. 82370 Wait State Generator—Block Diagram

software reset command. The circuit will hold the 80376's RESET line active while an external hardware reset signal is present at its RESET input. It can also reset the 80376 processor as the result of a software command. The software reset command causes the 82370 to hold the processor's RESET line active for a minimum of 62 clock cycles. The 80376 requires that its RESET line be held active for a minimum of 80 clock cycles to re-initialize. For a more detailed explanation and solution, see Appendix D (System Notes).

The 82370 can be programmed to sense the shutdown detect code on the status lines from the 80376. If the Shutdown Detect function is enabled, the 82370 will automatically reset the processor. A diagnostic register is available which can be used to determine the cause of reset.

1.1.7 REGISTER MAP RELOCATION

After a hardware reset, the internal registers of the 82370 are located in I/O space beginning at port address 0000H. The map of the 82370's registers is relocatable via a software command. The default mapping places the 82370 between I/O addresses 0000H and 00DBH. The relocation register allows this map to be moved to any even 256-byte boundary in the processor's 16-bit I/O address space or any even 64 kbyte boundary in the 24-bit memory address space.

1.2 Host Interface

The 82370 is designed to operate efficiently on the local bus of an 80376 microprocessor. The control signals of the 82370 are identical in function to those of the 80376. As a slave, the 82370 operates with all of the features available on the 80376 bus. When the 82370 is in the Master Mode, it looks identical to an 80376 to the connected devices.

The 82370 monitors the bus at all times, and determines whether the current bus cycle is a pipelined or non-pipelined access. All of the status signals of the processor are monitored.

The control, status, and data registers within the 82370 are located at fixed addresses relative to each other, but the group can be relocated to either memory or I/O space and to different locations within those spaces.

As a Slave device, the 82370 monitors the control/status lines of the CPU. The 82370 will generate all of the wait states it needs whenever it is accessed. This allows the programmer the freedom of accessing 82370 registers without having to insert NOPs in the program to wait for slower 82370 internal registers.

The 82370 can determine if a current bus cycle is a pipelined or a non-pipelined cycle. It does this by monitoring the ADS#, NA# and READY# signals and thereby keeping track of the current state of the 80376.

As a bus master, the 82370 looks like an 80376 to the rest of the system. This enables the designer greater flexibility in systems which include the 82370. The designer does not have to alter the interfaces of any peripherals designed to operate with the 80376 to accommodate the 82370. The 82370 will access any peripherals on the bus in the same manner as the 80376, including recognizing pipelined bus cycles.

The 82370 is accessed as an 8-bit peripheral. The 80376 places the data of all 8-bit accesses either on D(0–7) or D(8–15). The 82370 will only accept data on these lines when in the Slave Mode. When in the Master Mode, the 82370 is a full 16-bit machine, sending and receiving data in the same manner as the 80376.

2.0 80376 HOST INTERFACE

The 82370 contains a set of interface signals to operate efficiently with the 80376 host processor. These signals were designed so that minimal hardware is needed to connect the 82370 to the 80376. Figure 2-1 depicts a typical system configuration with the 80376 processor. As shown in the diagram, the 82370 is designed to interface directly with the 80376 bus.

Since the 82370 resides on the opposite side of the data bus transceivers with respect to the rest of the system peripherals, it is important to note that the transceivers should be controlled so that contention between the data bus transceivers and the 82370 will not occur. In order to ease the implementation of this, the 82370 activates the CHPSEL# signal which indicates that the 82370 has been addressed and may output data. This signal should be included in the direction and enable control logic of the transceiver. When any of the 82370 internal registers are read, the data bus transceivers should be disabled so that only the 82370 will drive the local bus.

This section describes the basic bus functions of the 82370 to show how this device interacts with the 80376 processor. Other signals which are not directly related to the host interface will be discussed in their associated functional block description.

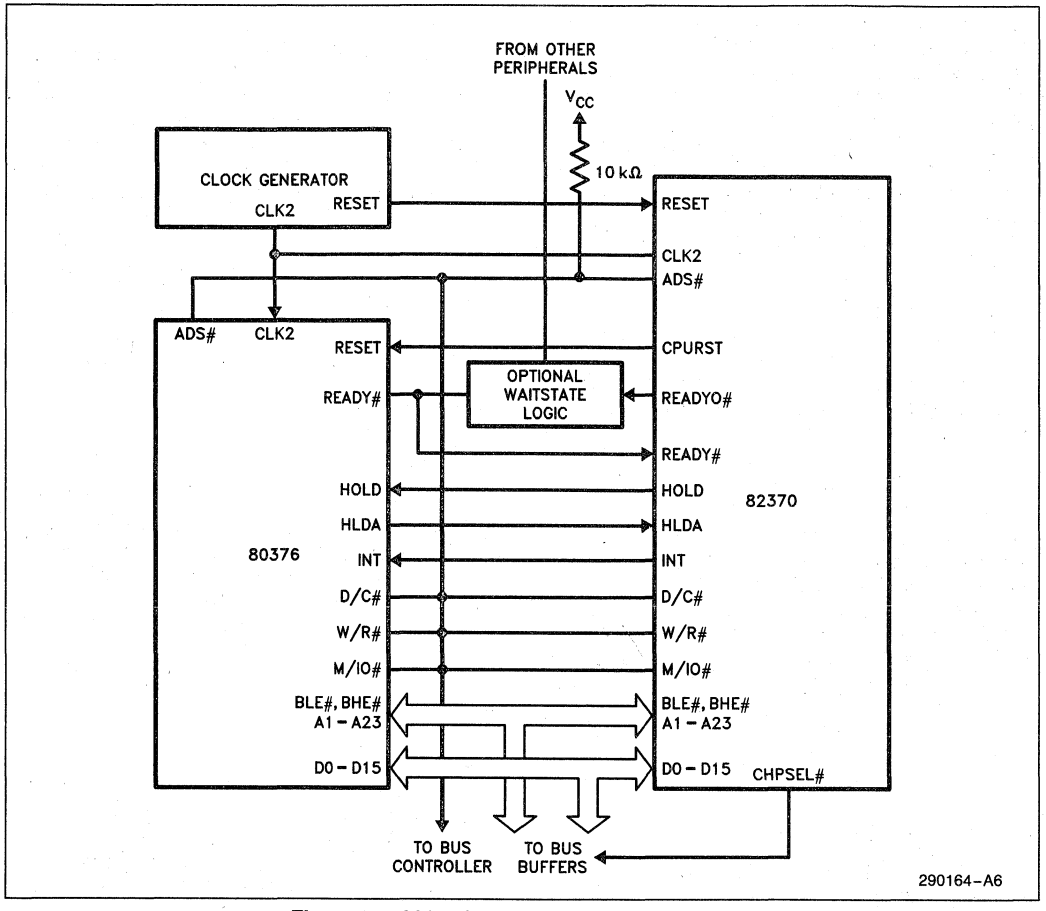


Figure 2-1. 80376/82370 System Configuration

2.1 Master and Slave Modes

At any time, the 82370 acts as either a Slave device or a Master device in the system. Upon reset, the 82370 will be in the Slave Mode. In this mode, the 80376 processor can read/write into the 82370 internal registers. Initialization information may be programmed into the 82370 during Slave Mode.

When DMA service (including DRAM Refresh Cycles generated by the 82370) is requested, the 82370 will request and subsequently get control of the 80376 local bus. This is done through the HOLD and HLDA (Hold Acknowledge) signals. When the 80376 processor responds by asserting the HLDA signal, the 82370 will switch into Master Mode and perform DMA transfers. In this mode, the 82370 is the bus master of the system. It can read/write data from/to memory and peripheral devices. The 82370 will return to the Slave Mode upon completion of DMA transfers, or when HLDA is negated.

2.2 80376 Interface Signals

As mentioned in the Architecture section, the Bus Interface module of the 82370 (see Figure 1-1) contains signals that are directly connected to the 80376 host processor. This module has separate

16-bit Data and 24-bit Address busses. Also, it has additional control signals to support different bus operations on the system. By residing on the 80376 local bus, the 82370 shares the same address, data and control lines with the processor. The following subsections discuss the signals which interface to the 80376 host processor.

2.2.1 CLOCK (CLK2)

The CLK2 input provides fundamental timing for the 82370. It is divided by two internally to generate the 82370 internal clock. Therefore, CLK2 should be driven with twice the 80376's frequency. In order to maintain synchronization with the 80376 host processor, the 82370 and the 80376 should share a common clock source.

The internal clock consists of two phases: PHI1 and PHI2. Each CLK2 period is a phase of the internal clock. PHI2 is usually used to sample input and set up internal signals and PHI1 is for latching internal data. Figure 2-2 illustrates the relationship of CLK2 and the 82370 internal clock signals. The CPURST signal generated by the 82370 guarantees that the 80376 will wake up in phase with PHI1.

2.2.2 DATA BUS (D_0–D_{15})

This 16-bit three-state bidirectional bus provides a general purpose data path between the 82370 and the system. These pins are tied directly to the corresponding Data Bus pins of the 80376 local bus. The Data Bus is also used for interrupt vectors generated by the 82370 in the Interrupt Acknowledge cycle.

During Slave I/O operations, the 82370 expects a single byte to be written or read. When the 80376 host processor writes into the 82370, either D_0–D_7 or D_8–D_{15} will be latched into the 82370, depending upon whether Byte Enable bit BLE# is 0 or 1 (see Table 2-1). When the 80376 host processor reads from the 82370, the single byte data will be duplicated twice on the Data Bus; i.e. on D_0–D_7 and D_8–D_{15}.

During Master Mode, the 82370 can transfer 16-, and 8-bit data between memory (or I/O devices) and I/O devices (or memory) via the Data Bus.

2.2.3 ADDRESS BUS (A_{23}–A_1)

These three-state bidirectional signals are connected directly to the 80376 Address Bus. In the Slave Mode, they are used as input signals so that the processor can address the 82370 internal ports/registers. In the Master Mode, they are used as output signals by the 82370 to address memory and peripheral devices. The Address Bus is capable of addressing 16 Mbytes of physical memory space (000000H to FFFFFFH), and 64 Kbytes of I/O addresses.

2.2.4 BYTE ENABLE (BHE#, BLE#)

The Byte Enable pins BHE# and BLE# select the specific byte(s) in the word addressed by A_1–A_{23}. During Master Mode operation, it is used as an output by the 82370 to address memory and I/O locations. The definition of BHE# and BLE# is further illustrated in Table 2-1.

NOTE:
The 82370 will activate BHE# when output in Master Mode. For a more detailed explanation and its solutions, see Appendix D (System Notes).

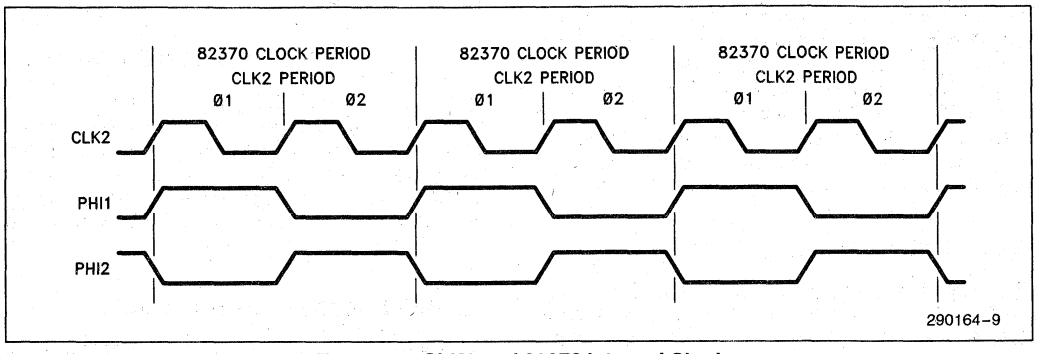


Figure 2-2. CLK2 and 82370 Internal Clock

As an output (Master Mode):

Table 2-1. Byte Enable Signals

BHE#	BLE#	Byte to be Accessed Relative to A_{23}–A_1	Logical Byte Presented on Data Bus During WRITE Only* D_{15}–D_8	D_7–D_0
0	0	0, 1	B	A
0	1	1	A	A
1	0	0	U	A
1	1	(Not Used)		

U = Undefined
A = Logical D_0–D_7
B = Logical D_8–D_{15}

***NOTE:**
Actual number of bytes accessed depends upon the programmed data path width.

Table 2-2. Bus Cycle Definition

M/IO#	D/C#	W/R#	As INPUTS	As OUTPUTS
0	0	0	Interrupt Acknowledge	NOT GENERATED
0	0	1	UNDEFINED	NOT GENERATED
0	1	0	I/O Read	I/O Read
0	1	1	I/O Write	I/O Write
1	0	0	UNDEFINED	NOT GENERATED
1	0	1	HALT if A_1 = 1 SHUTDOWN if A_1 = 0	NOT GENERATED
1	1	0	Memory Read	Memory Read
1	1	1	Memory Write	Memory Write

2.2.5 BUS CYCLE DEFINITION SIGNALS (D/C#, W/R#, M/IO#)

These three-state bidirectional signals define the type of bus cycle being performed. W/R# distinguishes between write and read cycles. D/C# distinguishes between processor data and control cycles. M/IO# distinguishes between memory and I/O cycles.

During Slave Mode, these signals are driven by the 80376 host processor; during Master Mode, they are driven by the 82370. In either mode, these signals will be valid when the Address Status (ADS#) is driven LOW. Exact bus cycle definitions are given in Table 2-2. Note that some combinations are recognized as inputs, but not generated as outputs. In the Master Mode, D/C# is always HIGH.

2.2.6 ADDRESS STATUS (ADS#)

This signal indicates that a valid address (A_1–A_{23}, BHE#, BLE#) and bus cycle definition (W/R#, D/C#, M/IO#) is being driven on the bus. In the Master Mode, it is driven by the 82370 as an output. In the Slave Mode, this signal is monitored as an input by the 82370. By the current and past status of ADS# and the READY# input, the 82370 is able to determine, during Slave Mode, if the next bus cycle is a pipelined address cycle. ADS# is asserted during T1 and T2P bus states (see Bus State Definition).

NOTE:
ADS# must be qualified with the rising edge of CLK2.

2.2.7 TRANSFER ACKNOWLEDGE (READY#)

This input indicates that the current bus cycle is complete. In the Master Mode, assertion of this signal indicates the end of a DMA bus cycle. In the Slave Mode, the 82370 monitors this input and ADS# to detect a pipelined address cycle. This signal should be tied directly to the READY# input of the 80376 host processor.

2.2.8 NEXT ADDRESS REQUEST (NA#)

This input is used to indicate to the 82370 in the Master Mode that the system is requesting address

pipelining. When driven LOW by either memory or peripheral devices during Master Mode, it indicates that the system is prepared to accept a new address and bus cycle definition signals from the 82370 before the end of the current bus cycle. If this input is active when sampled by the 82370, the next address is driven onto the bus, provided a bus request is already pending internally.

This input pin is monitored only in the Master Mode. In the Slave Mode, the 82370 uses the ADS# and READY# signals to determine address pipelining cycles, and NA# will be ignored.

2.2.9 RESET (RESET, CPURST)

RESET

This synchronous input suspends any operation in progress and places the 82370 in a known initial state. Upon reset, the 82370 will be in the Slave Mode waiting to be initialized by the 80376 host processor. The 82370 is reset by asserting RESET for 15 or more CLK2 periods. When RESET is asserted, all other input pins are ignored, and all other bus pins are driven to an idle bus state as shown in Table 2-3. The 82370 will determine the phase of its internal clock following RESET going inactive.

RESET is level-sensitive and must be synchronous to the CLK2 signal. The RESET setup and hold time requirements are shown in Figure 2-3.

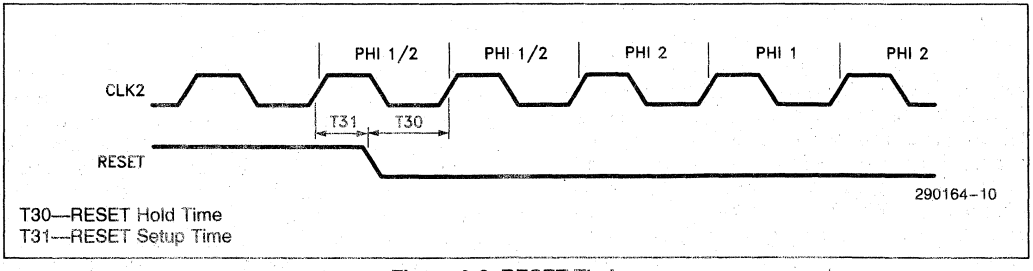
Table 2-3. Output Signals Following RESET

Signal	Level
A_1–A_{23}, D_0–D_{15}, BHE#, BLE#	Float
D/C#, W/R#, M/IO#, ADS#	Float
READYO#	'1'
EOP#	'1' (Weak Pull-UP)
EDACK2–EDACK0	'100'
HOLD	'0'
INT	UNDEFINED*
TOUT1/REF#, TOUT2#/IRQ3#, TOUT3#	UNDEFINED*
CPURST	'0'
CHPSEL#	'1'

***NOTE:**
The Interrupt Controller and Programmable Interval Timer are initialized by software commands.

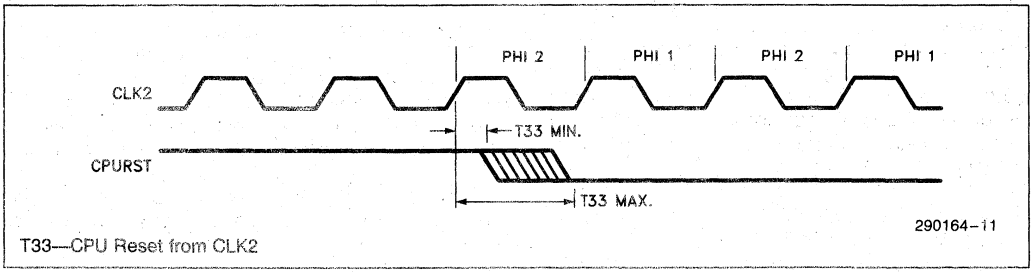
CPURST

This output signal is used to reset the 80376 host processor. It will go active (HIGH) whenever one of the following events occurs: a) 82370's RESET input is active; b) a software RESET command is issued to the 82370; or c) when the 82370 detects a processor Shutdown cycle and when this detection feature is enabled (see CPU Reset and Shutdown Detect). When activated, CPURST will be held active for 62 clocks. The timing of CPURST is such that the 80376 processor will be in synchronization with the 82370. This timing is shown in Figure 2-4.



T30—RESET Hold Time
T31—RESET Setup Time

Figure 2-3. RESET Timing



T33—CPU Reset from CLK2

Figure 2-4. CPURST Timing

2.2.10 INTERRUPT OUT (INT)

This output pin is used to signal the 80376 host processor that one or more interrupt requests (either internal or external) are pending. The processor is expected to respond with an Interrupt Acknowledge cycle. This signal should be connected directly to the Maskable Interrupt Request (INTR) input of the 80376 host processor.

2.3 82370 Bus Timing

The 82370 internally divides the CLK2 signal by two to generate its internal clock. Figure 2-2 showed the relationship of CLK2 and the internal clock which consists of two phases: PHI1 and PHI2. Each CLK2 period is a phase of the internal clock.

In the 82370, whether it is in the Master or Slave Mode, the shortest time unit of bus activity is a bus state. A bus state, which is also referred as a 'T-state', is defined as one 82370 PHI2 clock period (i.e. two CLK2 periods). Recall in Table 2-2 various types of bus cycles in the 82370 are defined by the M/IO#, D/C# and W/R# signals. Each of these bus cycles is composed of two or more bus states. The length of a bus cycle depends on when the READY# input is asserted (i.e. driven LOW).

2.3.1 ADDRESS PIPELINING

The 82370 supports Address Pipelining as an option in both the Master and Slave Mode. This feature typically allows a memory or peripheral device to operate with one less wait state than would otherwise be required. This is possible because during a pipelined cycle, the address and bus cycle definition of the next cycle will be generated by the bus master while waiting for the end of the current cycle to be acknowledged. The pipelined bus is especially well suited for an interleaved memory environment. For 16 MHz interleaved memory designs with 100 ns access time DRAMs, zero wait state memory accesses can be achieved when pipelined addressing is selected.

In the Master Mode, the 82370 is capable of initiating, on a cycle-by-cycle basis, either a pipelined or non-pipelined access depending upon the state of the NA# input. If a pipelined cycle is requested (indicated by NA# being driven LOW), the 82370 will drive the address and bus cycle definition of the next cycle as soon as there is an internal bus request pending.

In the Slave Mode, the 82370 is constantly monitoring the ADS# and READY# signals on the processor local bus to determine if the current bus cycle is a pipelined cycle. If a pipelined cycle is detected, the 82370 will request one less wait state from the processor if the Wait State Generator feature is selected. On the other hand, during an 82370 internal register access in a pipelined cycle, it will make use of the advance address and bus cycle information. In all cases, Address Pipelining will result in a savings of one wait state.

2.3.2 MASTER MODE BUS TIMING

When the 82370 is in the Master Mode, it will be in one of six bus states. Figure 2-5 shows the complete bus state diagram of the Master Mode, including pipelined address states. As seen in the figure, the 82370 state diagram is very similar to that of the 80376. The major difference is that in the 82370, there is no Hold state. Also, in the 82370, the conditions for some state transitions depend upon whether it is the end of a DMA process.

NOTE:
The term 'end of a DMA process' is loosely defined here. It depends on the DMA modes of operation as well as the state of the EOP# and DREQ inputs. This is expained in detail in section 3—DMA Controller.

The 82370 will enter the idle state, Ti, upon RESET and whenever the internal address is not available at the end of a DMA cycle or at the end of a DMA process. When address pipelining is not used (NA# is not asserted), a new bus cycle always begins with state T1. During T1, address and bus cycle definition signals will be driven on the bus. T1 is always followed by T2.

If a bus cycle is not acknowledged (with READY#) during T2 and NA# is negated, T2 will be repeated. When the end of the bus cycle is acknowledged during T2, the following state will be T1 of the next bus cycle (if the internal address latch is loaded and if this is not the end of the DMA process). Otherwise, the Ti state will be entered. Therefore, if the memory or peripheral accessed is fast enough to respond within the first T2, the fastest non-pipelined cycle will take one T1 and one T2 state.

Use of the address pipelining feature allows the 82370 to enter three additional bus states: T1P, T2P and T2i. T1P is the first bus state of a pipelined bus cycle. T2P follows T1P (or T2) if NA# is asserted when sampled. The 82370 will drive the bus with the address and bus cycle definition signals of the next cycle during T2P. From the state diagram, it can be seen that after an idle state Ti, the first bus cycle must begin with T1, and is therefore a non-pipelined bus cycle. The next bus cycle can be pipelined if

NA# is asserted and the previous bus cycle ended in a T2P state. Once the 82370 is in a pipelined cycle and provided that NA# is asserted in subsequent cycles, the 82370 will be switching between T1P and T2P states. If the end of the current bus cycle is not acknowledged by the READY# input, the 82370 will extend the cycle by adding T2P states. The fastest pipelined cycle will consist of one T1P and one T2P state.

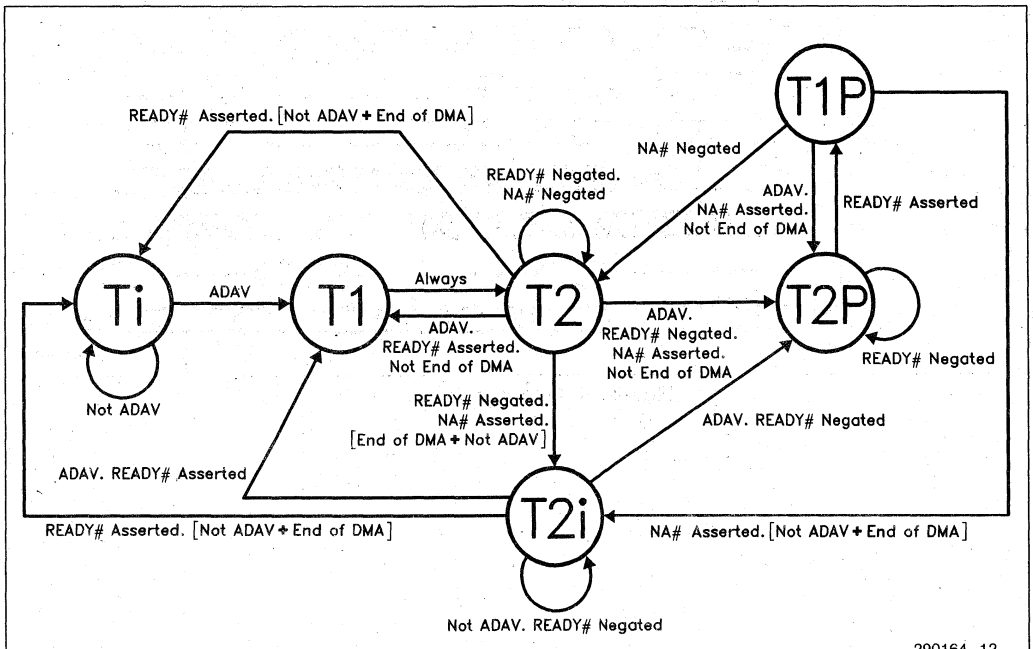
The 82370 will enter state T2i when NA# is asserted and when one of the following two conditions occurs. The first condition is when the 82370 is in state T2. T2i will be entered if READY# is not asserted and there is no next address available. This situation is similar to a wait state. The 82370 will stay in T2i for as long as this condition exists. The second condition which will cause the 82370 to enter T2i is when the 82370 is in state T1P. Before going to state T2P, the 82370 needs to wait in state T2i until the next address is available. Also, in both cases, if the DMA process is complete, the 82370 will enter the T2i state in order to finish the current DMA cycle.

Figure 2-6 is a timing diagram showing non-pipelined bus accesses in the Master Mode. Figure 2-7 shows the timing of pipelined accesses in the Master Mode.

2.3.3 SLAVE MODE BUS TIMING

Figure 2-8 shows the Slave Mode bus timing in both pipelined and non-pipelined cycles when the 82370 is being accessed. Recall that during Slave Mode, the 82370 will constantly monitor the ADS# and READY# signals to determine if the next cycle is pipelined. In Figure 2-8, the first cycle is non-pipelined and the second cycle is pipelined. In the pipelined cycle, the 82370 will start decoding the address and bus cycle signals one bus state earlier than in a non-pipelined cycle.

The READY# input signal is sampled by the 80376 host processor to determine the completion of a bus cycle. This occurs during the end of every T2, T2i and T2P state. Normally, the output of the 82370 Wait State Generator, READYO#, is directly connected to the READY# input of the 80376 host processor and the 82370. In such case, READYO# and READY# will be identical (see Wait State Generator).



NOTE:
ADAV—Internal Address Available

Figure 2-5. Master Mode State Diagram

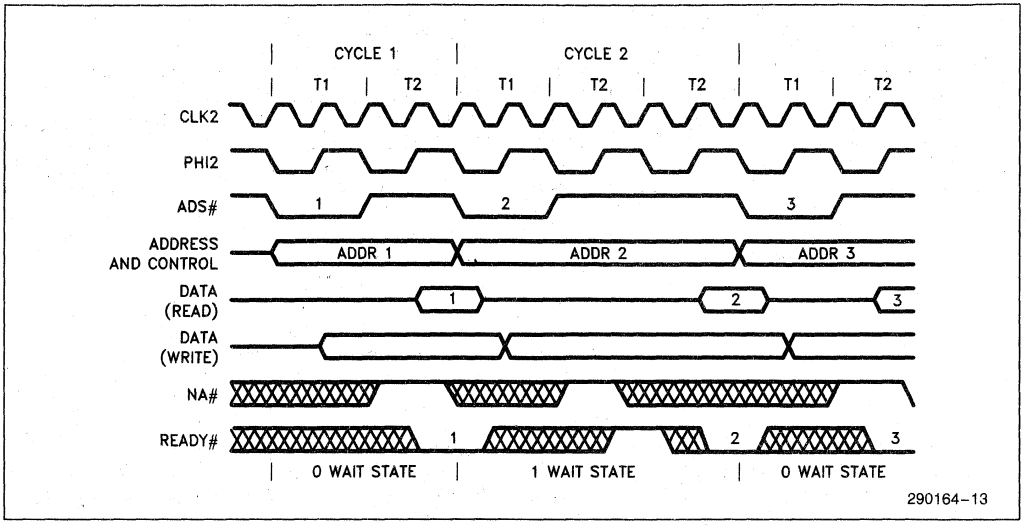


Figure 2-6. Non-Pipelined Bus Cycles

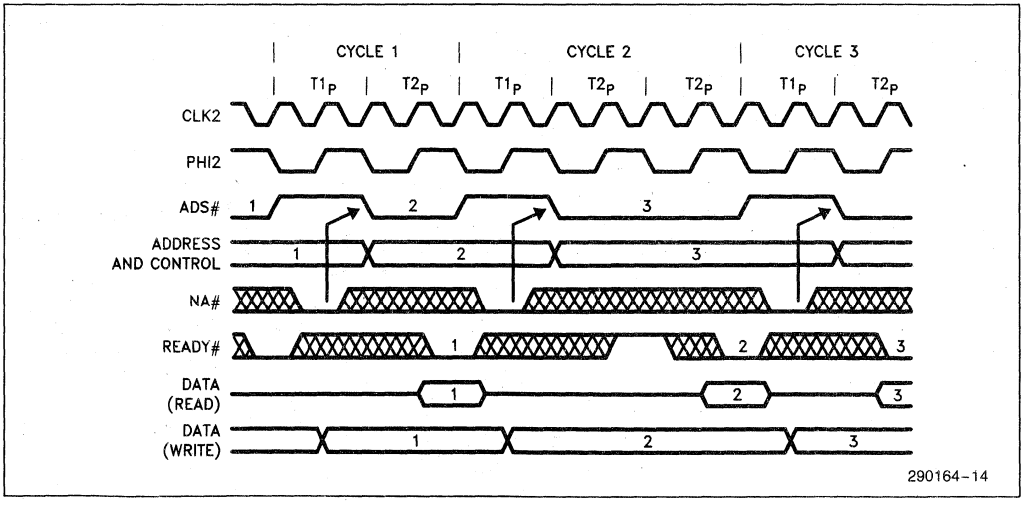
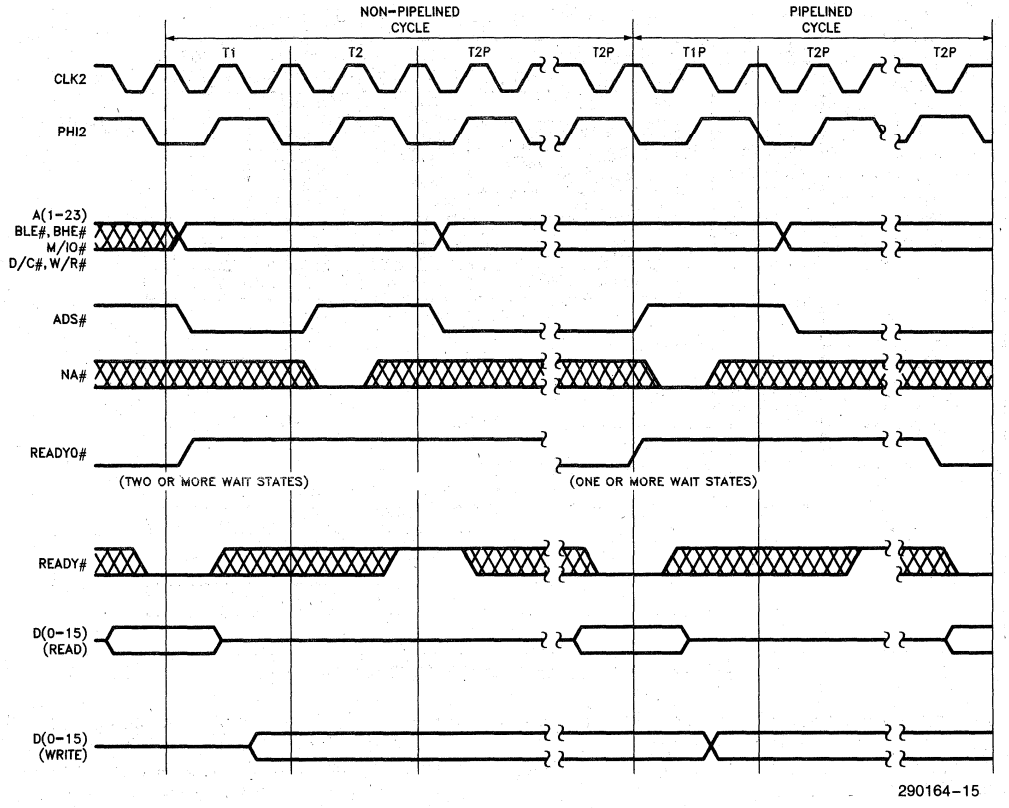


Figure 2-7. Pipelined Bus Cycles



NOTE:
NA# is shown here only for timing reference. It is not sampled by the 82370 during Slave Mode.
When the 82370 registers are accessed, it will take one or more wait states in pipelined and two or more wait states in non-pipelined cycle to complete the internal access.

Figure 2-8. Slave Read/Write Timing

3.0 DMA CONTROLLER

The 82370 DMA Controller is capable of transferring data between any combination of memory and/or I/O, with any combination of data path widths. The 82370 DMA Controller can be programmed to accommodate 8- or 16-bit devices. With its 16-bit external data path, it can transfer data in units of byte or a word. Bus bandwidth is optimized through the use of an internal temporary register which can disassemble or assemble data to or from either an aligned or non-aligned destination or source. Figure 3-1 is a block diagram of the 82370 DMA Controller.

The 82370 has eight channels of DMA. Each channel operates independently of the others. Within the operation of the individual channels, there are many different modes of data transfer available. Many of the operating modes can be intermixed to provide a very versatile DMA controller.

3.1 Functional Description

In describing the operation of the 82370's DMA Controller, close attention to terminology is required. Be-

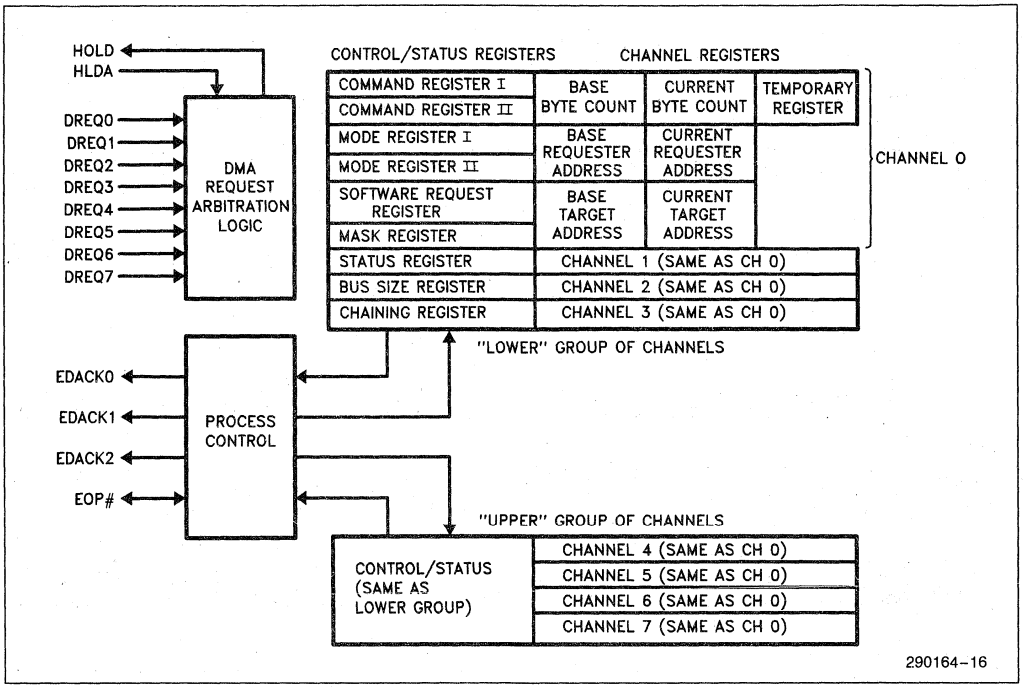


Figure 3-1. 82370 DMA Controller Block Diagram

fore entering the discussion of the function of the 82370 DMA Controller, the following explanations of some of the terminology used herein may be of benefit. First, a few terms for clarification:

DMA PROCESS—A DMA process is the execution of a programmed DMA task from beginning to end. Each DMA process requires intitial programming by the host 80376 microprocessor.

BUFFER—A contiguous block of data.

BUFFER TRANSFER—The action required by the DMA to transfer an entire buffer.

DATA TRANSFER—The DMA action in which a group of bytes or words are moved between devices by the DMA Controller. A data transfer operation may involve movement of one or many bytes.

BUS CYCLE—Access by the DMA to a single byte or word.

Each DMA channel consists of three major components. These components are identified by the contents of programmable registers which define the memory or I/O devices being serviced by the DMA. They are the Target, the Requester, and the Byte Count. They will be defined generically here and in greater detail in the DMA register definition section.

The Requester is the device which requires service by the 82370 DMA Controller, and makes the request for service. All of the control signals which the DMA monitors or generates for specific channels are logically related to the Requester. Only the Requester is considered capable of initiating or terminating a DMA process.

The Target is the device with which the Requester wishes to communicate. As far as the DMA process is concerned, the Target is a slave which is incapable of control over the process.

The direction of data transfer can be either from Requester to Target or from Target to Requester; i.e. each can be either a source or a destination.

The Requester and Target may each be either I/O or memory. Each has an address associated with it that can be incremented, decremented, or held constant. The addresses are stored in the Requester

Address Registers and Target Address Registers, respectively. These registers have two parts: one which contains the current address being used in the DMA process (Current Address Register), and one which holds the programmed base address (Base Address Register). The contents of the Base Registers are never changed by the 82370 DMA Controller. The Current Registers are incremented or decremented according to the progress of the DMA process.

The Byte Count is the component of the DMA process which dictates the amount of data which must be transferred. Current and Base Byte Count Registers are provided. The Current Byte Count Register is decremented once for each byte transferred by the DMA process. When the register is decremented past zero, the Byte Count is considered 'expired' and the process is terminated or restarted, depending on the mode of operation of the channel. The point at which the Byte Count expires is called 'Terminal Count' and several status signals are dependent on this event.

Each channel of the 82370 DMA Controller also contains a 32-bit Temporary Register for use in assembling and disassembling non-aligned data. The operation of this register is transparent to the user, although the contents of it may affect the timing of some DMA handshake sequences. Since there is data storage available for each channel, the DMA Controller can be interrupted without loss of data.

To avoid unexpected results, care should be taken in programming the byte count correctly when assembing and disassembling non-aligned data. For example:

Words to Bytes:
Transferring two words to bytes, but setting the byte count to three, will result in three bytes transferred and the final byte flushed.

Bytes to Words:
Transferring six bytes to three words, but setting the byte count to five, will result in the sixth byte transferred being undefined.

The 82370 DMA Controller is a slave on the bus until a request for DMA service is received via either a software request command or a hardware request signal. The host processor may access any of the control/status or channel registers at any time the 82370 is a bus slave. Figure 3-2 shows the flow of operations that the DMA Controller performs.

At the time a DMA service request is received, the DMA Controller issues a bus hold request to the host processor. The 82370 becomes the bus master when the host relinquishes the bus by asserting a hold acknowledge signal. The channel to be serviced will be the one with the highest priority at the time the DMA Controller becomes the bus master. The DMA Controller will remain in control of the bus until the hold acknowledge signal is removed, or until the current DMA transfer is complete.

While the 82370 DMA Controller has control of the bus, it will perform the required data transfer(s). The type of transfer, source and destination addresses, and amount of data to transfer are programmed in the control registers of the DMA channel which received the request for service.

At completion of the DMA process, the 82370 will remove the bus hold request. At this time the 82370 becomes a slave again, and the host returns to being a master. If there are other DMA channels with requests pending, the controller will again assert the hold request signal and restart the bus arbitration and switching process.

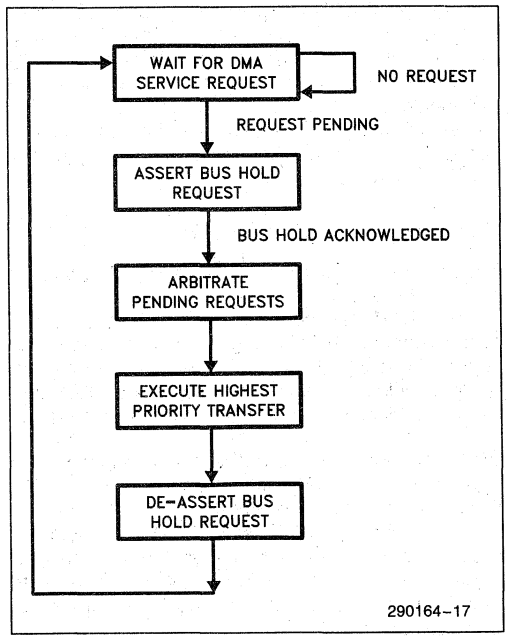


Figure 3-2. Flow of DMA Controller Operation

3.2 Interface Signals

There are fourteen control signals dedicated to the DMA process. They include eight DMA Channel Requests (DREQn), three Encoded DMA Acknowledge signals (EDACKn), Processor Hold and Hold Ac-

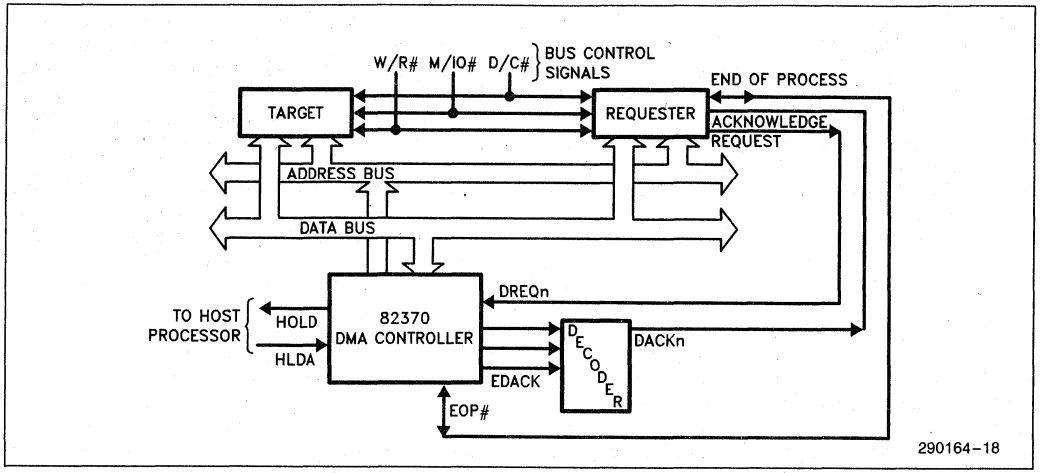


Figure 3-3. Requester, Target and DMA Controller Interconnection

knowledge (HOLD, HLDA), and End-of-Process (EOP#). The DREQn inputs and EDACK (0–2) outputs are handshake signals to the devices requiring DMA service. The HOLD output and HLDA input are handshake signals to the host processor. Figure 3-3 shows these signals and how they interconnect between the 82370 DMA Controller, and the Requester and Target devices.

3.2.1 DREQn and EDACK (0–2)

These signals are the handshake signals between the peripheral and the 82370. When the peripheral requires DMA service, it asserts the DREQn signal of the channel which is programmed to perform the service. The 82370 arbitrates the DREQn against other pending requests and begins the DMA process after finishing other higher priority processes.

When the DMA service for the requested channel is in progress, the EDACK (0–2) signals represent the DMA channel which is accessing the Requester. The 3-bit code on the EDACK (0–2) lines indicates the number of the channel presently being serviced. Table 3-2 shows the encoding of these signals. Note that Channel 4 does not have a corresponding hardware acknowledge.

The DMA acknowledge (EDACK) signals indicate the active channel only during DMA accesses to the Requester. During accesses to the Target, EDACK (0–2) has the idle code (100). EDACK (0–2) can thus be used to select a Requester device during a transfer.

DREQn can be programmed as either an Asynchronous or Synchronous input. See section 3.4.1 for details on synchronous versus asynchronous operation of these pins.

Table 3-2. EDACK Encoding During a DMA Transfer

EDACK2	EDACK1	EDACK0	Active Channel
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	Target Access
1	0	1	5
1	1	0	6
1	1	1	7

The EDACKn signals are always active. They either indicate 'no acknowledge' or they indicate a bus access to the requester. The acknowledge code is either 100, for an idle DMA or during a DMA access to the Target, or 'n' during a Requester access, where n is the binary value representing the channel. A simple 3-line to 8-line decoder can be used to provide discrete acknowledge signals for the peripherals.

3.2.2 HOLD AND HLDA

The Hold Request (HOLD) and Hold Acknowledge (HLDA) signals are the handshake signals between the DMA Controller and the host processor. HOLD is an output from the 82370 and HLDA is an input. HOLD is asserted by the DMA Controller when there is a pending DMA request, thus requesting the processor to give up control of the bus so the DMA process can take place. The 80376 responds by asserting HLDA when it is ready to relinquish control of the bus.

The 82370 will begin operations on the bus one clock cycle after the HLDA signal goes active. For this reason, other devices on the bus should be in the slave mode when HLDA is active.

HOLD and HLDA should not be used to gate or select peripherals requesting DMA service. This is because of the use of DMA-like operations by the DRAM Refresh Controller. The Refresh Controller is arbitrated with the DMA Controller for control of the bus, and refresh cycles have the highest priority. A refresh cycle will take place between DMA cycles without relinquishing bus control. See section 3.4.3 for a more detailed discussion of the interaction between the DMA Controller and the DRAM Refresh Controller.

3.2.3 EOP#

EOP# is a bi-directional signal used to indicate the end of a DMA process. The 82370 activates this as an output during the T2 states of the last Requester bus cycle for which a channel is programmed to execute. The Requester should respond by either withdrawing its DMA request, or interrupting the host processor to indicate that the channel needs to be programmed with a new buffer. As an input, this signal is used to tell the DMA Controller that the peripheral being serviced does not require any more data to be transferred. This indicates that the current buffer is to be terminated.

EOP# can be programmed as either an Asynchronous or a Synchronous input. See section 3.4.1 for details on synchronous versus asynchronous operation of this pin.

3.3 Modes of Operation

The 82370 DMA Controller has many independent operating functions. When designing peripheral interfaces for the 82370 DMA Controller, all of the functions or modes must be considered. All of the channels are independent of each other (except in priority of operation) and can operate in any of the modes. Many of the operating modes, though independently programmable, affect the operation of other modes. Because of the large number of combinations possible, each programmable mode is discussed here with its affects on the operation of other modes. The entire list of possible combinations will not be presented.

Table 3-1 shows the categories of DMA features available in the 82370. Each of the five major categories is independent of the others. The sub-categories are the available modes within the major function or mode category. The following sections explain each mode or function and its relation to other features.

Table 3-1. DMA Operating Modes

I. TARGET/REQUESTER DEFINITION
a. Data Transfer Direction
b. Device Type
II. BUFFER PROCESSES
a. Single Buffer Process
b. Buffer Auto-Initialize Process
c. Buffer Chaining Process
III. DATA TRANSFER/HANDSHAKE MODES
a. Single Transfer Mode
b. Demand Transfer Mode
c. Block Transfer Mode
d. Cascade Mode
IV. PRIORITY ARBITRATION
a. Fixed
b. Rotating
c. Programmable Fixed
V. BUS OPERATION
a. Fly-By (Single-Cycle)/Two-Cycle
b. Data Path Width
c. Read, Write, or Verify Cycles

3.3.1 TARGET/REQUESTER DEFINITION

All DMA transfers involve three devices: the DMA Controller, the Requester, and the Target. Since the devices to be accessed by the DMA Controller vary widely, the operating characteristics of the DMA Controller must be tailored to the Requester and Target devices.

The Requester can be defined as either the source or the destination of the data to be transferred. This is done by specifying a Write or a Read transfer, respectively. In a Read transfer, the Target is the data source and the Requester is the destination for the data. In a Write transfer, the Requester is the source and the Target is the destination.

The Requester and Target addresses can each be independently programmed to be incremented, decremented, or held constant. As an example, the 82370 is capable of reversing a string of data by having the Requester address increment and the Target address decrement in a memory-to-memory transfer.

3.3.2 BUFFER TRANSFER PROCESSES

The 82370 DMA Controller allows three programmable Buffer Transfer Processes. These processes define the logical way in which a buffer of data is accessed by the DMA.

The three Buffer Transfer Processes include the Single Buffer Process, the Buffer Auto-Initialize Process, and the Buffer Chaining Process. These processes require special programming considerations. See the DMA Programming section for more details on setting up the Buffer Transfer Processes.

Single Buffer Process

The Single Buffer Process allows the DMA channel to transfer only one buffer of data. When the buffer has been completely transferred (Current Byte Count decremented past zero or EOP# input active), the DMA process ends and the channel becomes idle. In order for that channel to be used again, it must be reprogrammed.

The Single Buffer Process is usually used when the amount of data to be transferred is known exactly, and it is also known that there is not likely to be any data to follow before the operating system can reprogram the channel.

Buffer Auto-Initialize Process

The Buffer Auto-Initialize Process allows multiple groups of data to be transferred to or from a single buffer. This process does not require reprogramming. The Current Registers are automatically reprogrammed from the Base Registers when the current process is terminated, either by an expired Byte Count or by an external EOP# signal. The data transferred will always be between the same Target and Requester.

The auto-initialization/process-execution cycle is repeated until the channel is either disabled or re-programmed.

Buffer Chaining Process

The Buffer Chaining Process is useful for transferring large quantities of data into non-contiguous buffer areas. In this process, a single channel is used to process data from several buffers, while having to program the channel only once. Each new buffer is programmed in a pipelined operation that provides the new buffer information while the old buffer is being processed. The chain is created by loading new buffer information while the 82370 DMA Controller is processing the Current Buffer. When the Current Buffer expires, the 82370 DMA Controller automatically restarts the channel using the new buffer information.

Loading the new buffer information is done by an interrupt routine which is requested by the 82370. Interrupt Request 1 (IRQ1) is tied internally to the 82370 DMA Controller for this purpose. IRQ1 is generated by the 82370 when the new buffer information is loaded into the channel's Current Registers, leaving the Base Registers 'empty'. The interrupt service routine loads new buffer information into the Base Registers. The host processor is required to load the information for another buffer before the current Byte Count expires. The process repeats until the host programs the channel back to single buffer operation, or until the channel runs out of buffers.

The channel runs out of buffers when the Current Buffer expires and the Base Registers have not yet been loaded with new buffer information. When this occurs, the channel must be reprogrammed.

If an external EOP# is encountered while executing a Buffer Chaining Process, the current buffer is considered expired and the new buffer information is loaded into the Current Registers. If the Base Registers are 'empty', the chain is terminated.

The channel uses the Base Target Address Register as an indicator of whether or not the Base Registers are full. When the most significant byte of the Base Target Register is loaded, the channel considers all of the Base Registers loaded, and removes the interrupt request. This requires that the other Base Registers (Base Requester Address, Base Byte Count) must be loaded before the Base Target Address Register. The reason for implementing the reloading process this way is that, for most applications, the Byte Count and the Requester will not change from one buffer to the next, and therefore do not need to be reprogrammed. The details of programming the channel for the Buffer Chaining Process can be found in the section on DMA programming.

3.3.3 DATA TRANSFER MODES

Three Data Transfer modes are available in the 82370 DMA Controller. They are the Single Transfer, Block Transfer, and Demand Transfer Modes. These transfer modes can be used in conjunction with any one of three Buffer Transfer modes: Single Buffer, Auto-Initialized Buffer and Buffer Chaining. Any Data Transfer Mode can be used under any of the Buffer Transfer Modes. These modes are independently available for all DMA channels.

Different devices being serviced by the DMA Controller require different handshaking sequences for data transfers to take place. Three handshaking modes are available on the 82370, giving the designer the opportunity to use the DMA Controller as efficiently as possible. The speed at which data can

be presented or read by a device can affect the way a DMA Controller uses the host's bus, thereby affecting not only data throughput during the DMA process, but also affecting the host's performance by limiting its access to the bus.

Single Transfer Mode

In the Single Transfer Mode, one data transfer to or from the Requester is performed by the DMA Controller at a time. The DREQn input is arbitrated and the HOLD/HLDA sequence is executed for each transfer. Transfers continue in this manner until the Byte Count expires, or until EOP# is sampled active. If the DREQn input is held active continuously, the entire DREQ-HOLD-HLDA-DACK sequence is repeated over and over until the programmed number of bytes has been transferred. Bus control is released to the host between each transfer. Figure 3-4 shows the logical flow of events which make up a buffer transfer using the Single Transfer Mode. Refer to section 3.4 for an explanation of the bus control arbitration procedure.

The Single Transfer Mode is used for devices which require complete handshake cycles with each data access. Data is transferred to or from the Requester only when the Requester is ready to perform the transfer. Each transfer requires the entire DREQ-HOLD-HLDA-DACK handshake cycle. Figure 3-5 shows the timing of the Single Transfer Mode cycle.

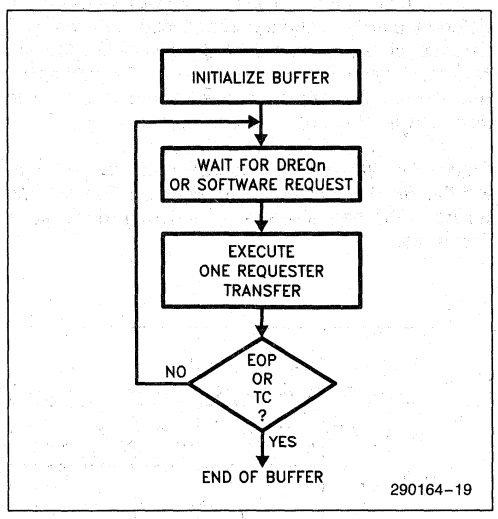
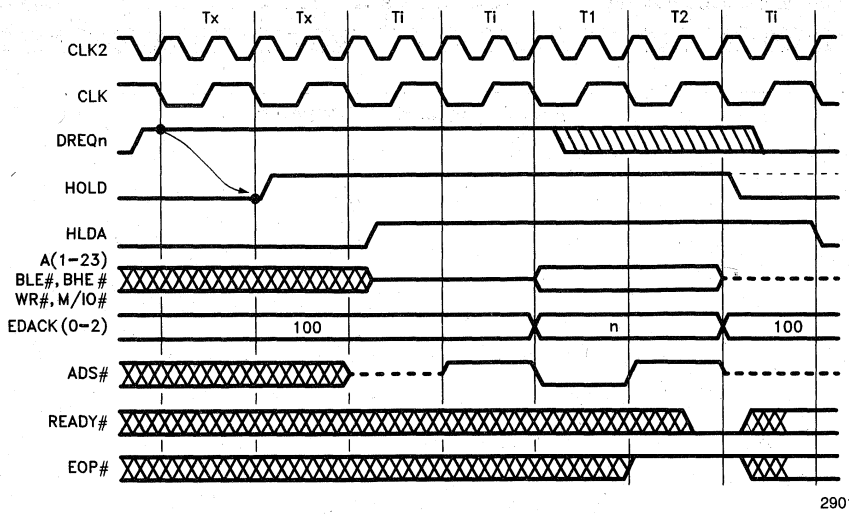


Figure 3-4. Buffer Transfer in Single Transfer Mode



NOTE:
The Single Transfer Mode is more efficient (15%–20%) in the case where the source is the Target. Because of the internal pipeline of the 82370 DMA Controller, two idle states are added at the end of a transfer in the case where the source is the Requester.

Figure 3-5. DMA Single Transfer Mode

Block Transfer Mode

In the Block Transfer Mode, the DMA process is initiated by a DMA request and continues unti the Byte Count expires, or until EOP# is activated by the Requester. The DREQn signal need only be held active until the first Requester access. Only a refresh cycle will interrupt the block transfer process.

Figure 3-6 illustrates the operation of the DMA during the Block Transfer Mode. Figure 3-7 shows the timing of the handshake signals during Block Mode Transfers.

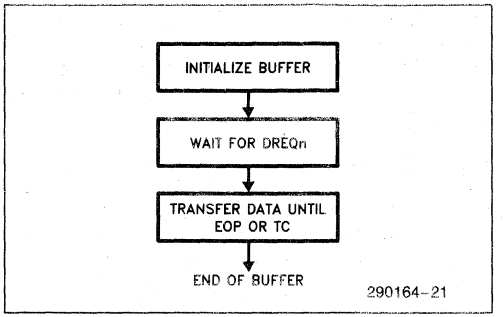


Figure 3-6. Buffer Transfer in Block Transfer Mode

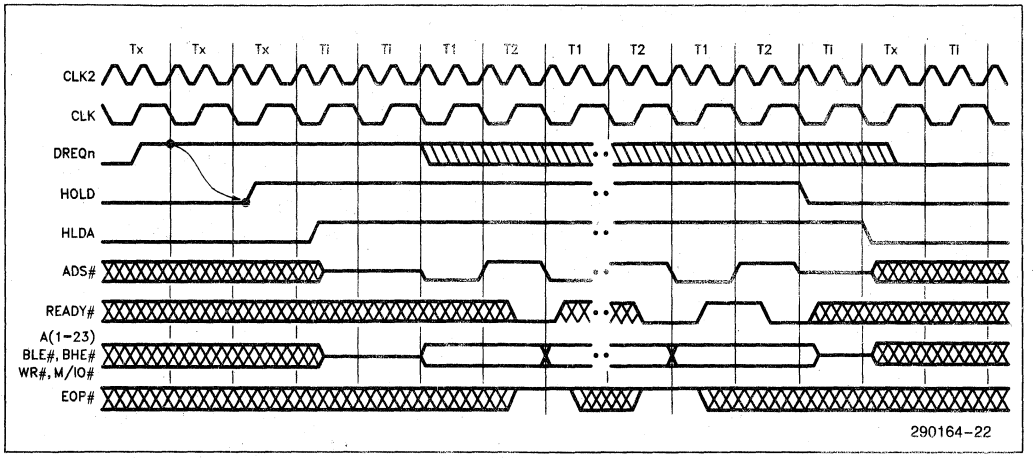


Figure 3-7. Block Mode Transfers

Demand Transfer Mode

The Demand Transfer Mode provides the most flexible handshaking procedures during the DMA process. A Demand Transfer is initiated by a DMA request. The process continues until the Byte Count expires, or an external EOP# is encountered. If the device being serviced (Requester) desires, it can interrupt the DMA process by de-activating the DREQn line. Action is taken on the condition of DREQn during Requester accesses only. The access during which DREQn is sampled inactive is the last Requester access which will be performed during the current transfer. Figure 3-8 shows the flow of events during the transfer of a buffer in the Demand Mode.

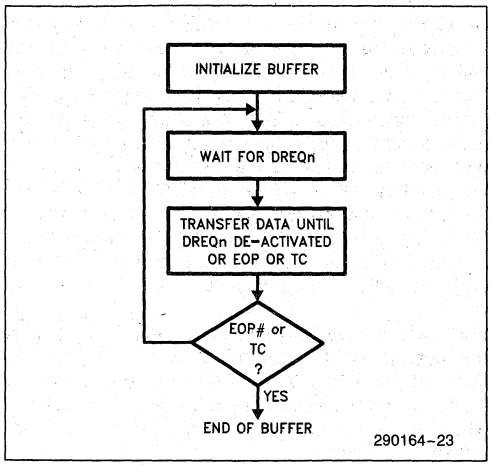


Figure 3-8. Buffer Transfer in Demand Transfer Mode

When the DREQn line goes inactive, the DMA Controller will complete the current transfer, including any necessary accesses to the Target, and relinquish control of the bus to the host. The current process information is saved (byte count, Requester and Target addresses, and Temporary Register).

The Requester can restart the transfer process by reasserting DREQn. The 82370 will arbitrate the request with other pending requests and begin the process where it left off. Figure 3-9 shows the timing of handshake signals during Demand Transfer Mode operation.

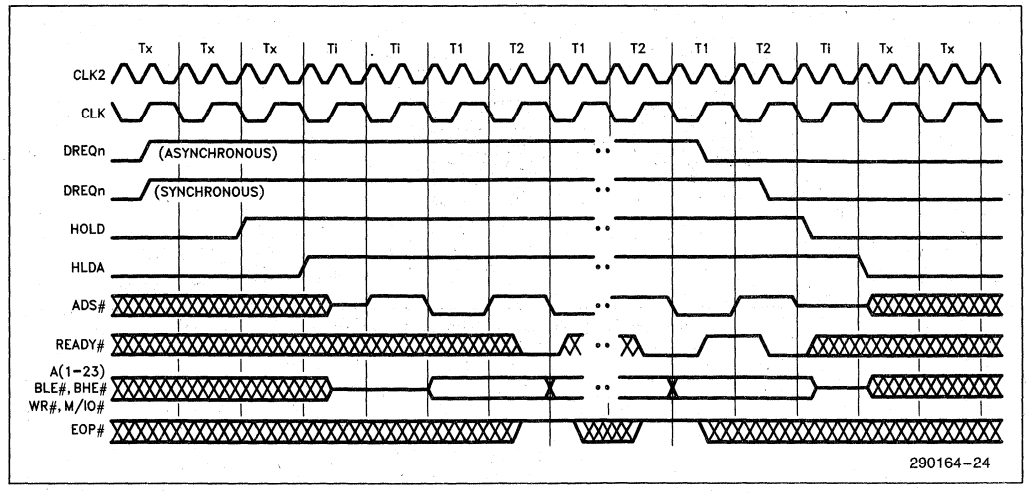


Figure 3-9. Demand Mode Transfers

Using the Demand Transfer Mode allows peripherals to access memory in small, irregular bursts without wasting bus control time. The 82370 is designed to give the best possible bus control latency in the Demand Transfer Mode. Bus control latency is defined here as the time form the last active bus cycle of the previous bus master to the first active bus cycle of the new bus master. The 82370 DMA Controller will perform its first bus access cycle two bus states after HLDA goes active. In the typical configuration, bus control is returned to the host one bus state after the DREQn goes inactive.

There are two cases where there may be more than one bus state of bus control latency at the end of a transfer. The first is at the end of an Auto-Initialize process, and the second is at the end of a process where the source is the Requester and Two-Cycle transfers are used.

When a Buffer Auto-Initialize Porcess is complete, the 82370 requires seven bus states to reload the Current Registers from the Base Registers of the Auto-Initialized channel. The reloading is done while the 82370 is still the bus master so that it is prepared to service the channel immediately after relinquishing the bus, if necessary.

In the case where the Requester is the source, and Two-Cycle transfers are being used, there are two extra idle states at the end of the transfer process. This occurs due to the housekeeping in the DMA's internal pipeline. These two idle states are present only after the very last Requester access, before the DMA Controller de-activates the HOLD signal.

3.3.4 CHANNEL PRIORITY ARBITRATION

DMA channel priority can be programmed into one of two arbitration methods: Fixed or Rotating. The four lower DMA channels and the four upper DMA channels operate as if they were two separate DMA controllers operating in cascade. The lower group of four channels (0–3) is always prioritized between channels 7 and 4 of the upper group of channels (4–7). Figure 3-10 shows a pictorial representation of the priority grouping.

The priority can thus be set up as rotating for one group of channels and fixed for the other, or any other combination. While in Fixed Priority, the programmer can also specify which channel has the lowest priority.

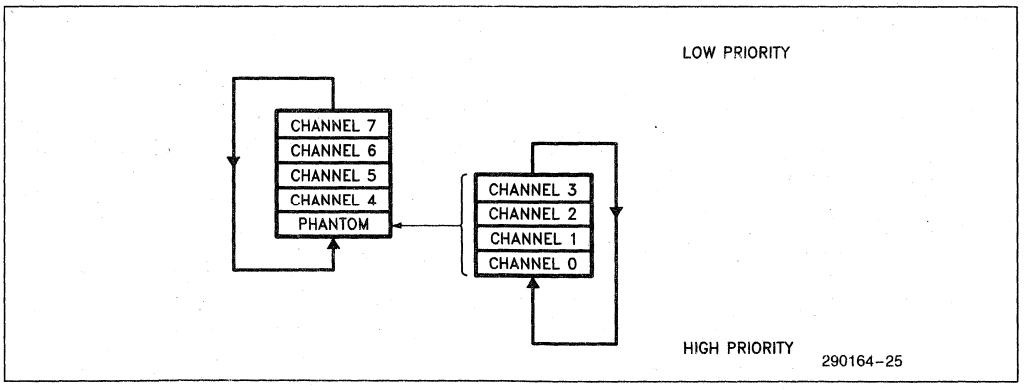


Figure 3-10. DMA Priority Grouping

The 82370 DMA Controller defaults to Fixed Priority. Channel 0 has the highest priority, then 1, 2, 3, 4, 5, 6, 7. Channel 7 has the lowest priority. Any time the DMA Controller arbitrates DMA requests, the requesting channel with the highest priority will be serviced next.

Fixed Priority can be entered into at any time by a software command. The priority levels in effect after the mode switch are determined by the current setting of the Programmable Priority.

Programmable Priority is available for fixing the priority of the DMA channels within a group to levels other than the default. Through a software command, the channel to have the lowest priority in a group can be specified. Each of the two groups of four channels can have the priority fixed in this way. The other channels in the group will follow the natural Fixed Priority sequence. This mode affects only the priority levels while operating with Fixed Priority.

For example, if channel 2 is programmed to have the lowest priority in its group, channel 3 has the highest priority. In descending order, the other channels would have the following priority: (3,0,1,2),4,5,6,7 (channel 2 lowest, channel 3 highest). If the upper group were programmed to have channel 5 as the lowest priority channel, the priority would be (again, highest to lowest): 6,7, (3,0,1,2), 4,5. Figure 3-11 shows this example pictorially. The lower group is always prioritized as a fifth channel of the upper group (between channels 4 and 7).

The DMA Controller will only accept Programmable Priority commands while the addressed group is operating in Fixed Priority. Switching from Fixed to Rotating Priority preserves the current priority levels. Switching from Rotating to Fixed Priority returns the priority levels to those which were last programmed by use of Programmable Priority.

Rotating Priority allows the devices using DMA to share the system bus more evenly. An individual channel does not retain highest priority after being serviced, priority is passed to the next highest priority channel in the group. The channel which was most recently serviced inherits the lowest priority. This rotation occurs each time a channel is serviced. Figure 3-12 shows the sequence of events as priority is passed between channels. Note that the lower group rotates within the upper group, and that servicing a channel within the lower group causes rotation within the group as well as rotation of the upper group.

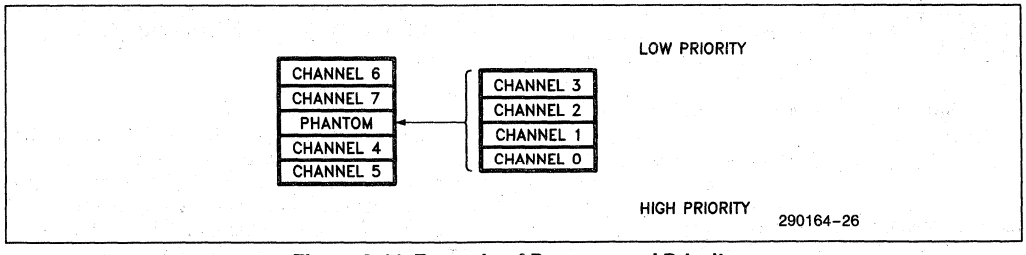


Figure 3-11. Example of Programmed Priority

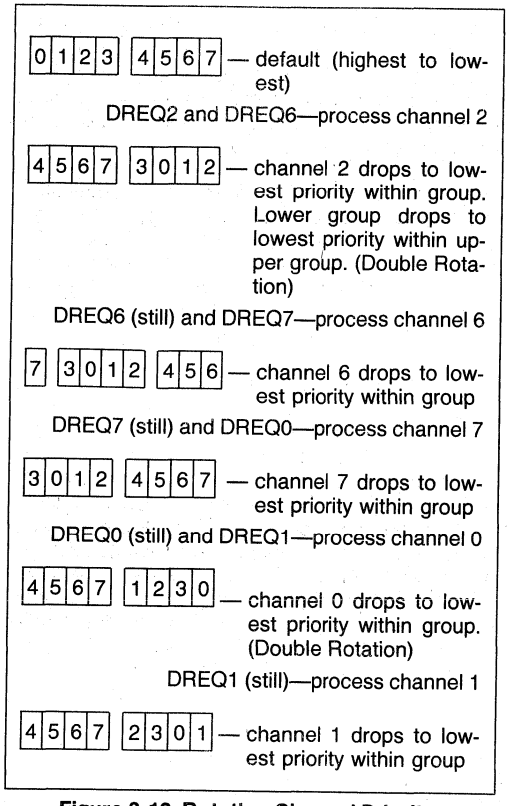


Figure 3-12. Rotating Channel Priority. Lower and upper groups are programmed for the Rotating Priority Mode.

3.3.5 COMBINING PRIORITY MODES

Since the DMA Controller operates as two four-channel controllers in cascade, the overall priority scheme of all eight channels can take on a variety of forms. There are four possible combinations of priority modes between the two groups of channels: Fixed Priority only (default), Fixed Priority upper group/Rotating Priority lower group, Rotating Priority upper group/Fixed Priority lower group, and Rotating Priority only. Figure 3-13 illustrates the operation of the two combined priority methods.

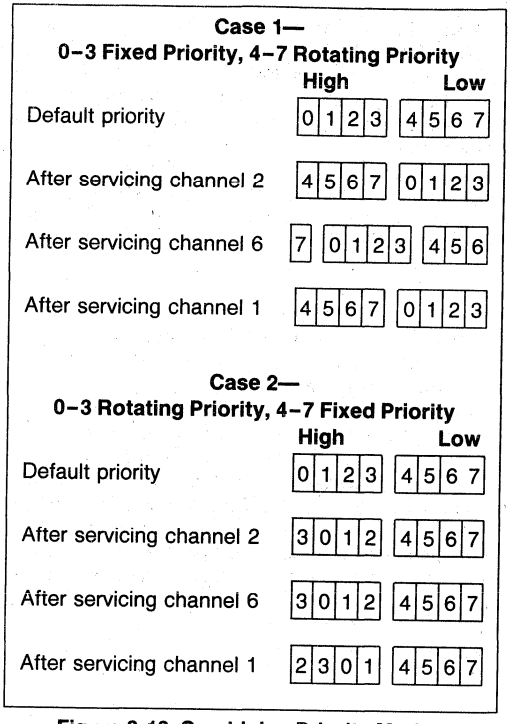


Figure 3-13. Combining Priority Modes

3.3.6 BUS OPERATION

Data may be transferred by the DMA Controller using two different bus cycle operations: Fly-By (one-cycle) and Two-Cycle. These bus handshake methods are selectable independently for each channel through a command register. Device data path widths are independently programmable for both Target and Requester. Also selectable through software is the direction of data transfer. All of these parameters affect the operation of the 82370 on a bus-cycle by bus-cycle basis.

3.3.6.1 Fly-By Transfers

The Fly-By Transfer Mode is the fastest and most efficient way to use the 82370 DMA Controller to transfer data. In this method of transfer, the data is written to the destination device at the same time it is read from the source. Only one bus cycle is used to accomplish the transfer.

In the Fly-By Mode, the DMA acknowledge signal is used to select the Requester. The DMA Controller simultaneously places the address of the Target on the address bus. The state of M/IO# and W/R# during the Fly-By transfer cycle indicate the type of Target and whether the Target is being written to or read from. The Target's Bus Size is used as an incrementer for the Byte Count. The Requester address registers are ignored during Fly-By transfers.

Note that memory-to-memory transfers cannot be done using the Fly-By Mode. Only one memory of I/O address is generated by the DMA Controller at a time during Fly-By transfers. Only one of the devices being accessed can be selected by an address. Also, the Fly-By method of data transfer limits the hardware to accesses of devices with the same data bus width. The Temporary Registers are not affected in the Fly-By Mode.

Fly-By transfers also require that the data paths of the Target and Requester be directly connected. This requires that successive Fly-By access be to word boundaries, or that the Requester be capable of switching its connections to the data bus.

3.3.6.2. Two-Cycle Transfers

Two-Cycle transfers can also be performed by the 82370 DMA Controller. These transfers require at least two bus cycles to execute. The data being transferred is read into the DMA Controller's Temporary Register during the first bus cycle(s). The second bus cycle is used to write the data from the Temporary Register to the destination.

If the addresses of the data being transferred are not word aligned, the 82370 will recognize the situation and read and write the data in groups of bytes, placing them always at the proper destination. This process of collecting the desired bytes and putting them together is called "byte assembly". The reverse process (reading from aligned locations and writing to non-aligned locations) is called "byte disassembly".

The assembly/disassembly process takes place transparent to the software, but can only be done while using the Two-Cycle transfer method. The 82370 will always perform the assembly/disassembly process as necessary for the current data transfer. Any data path widths for either the Requester or Target can be used in the Two-Cycle Mode. This is very convenient for interfacing existing 8- and 16-bit peripherals to the 80376's 16-bit bus.

The 82370 DMA Controller always reads and write data within the word boundaries; i.e. if a word to be read is crossing a word boundary, the DMA Controller will perform two read operations, each reading one byte, to read the 16-bit word into the Temporary Register. Also, the 82370 DMA Controller always attempts to fill the Temporary Register from the source before writing any data to the destination. If the process is terminated before the Temporary Register is filled (TC or EOP#), the 82370 will write the partial data to the destination. If a process is temporarily suspended (such as when DREQn is deactivated during a demand transfer), the contents of a partially filled Temporary Register will be stored within the 82370 until the process is restarted.

For example, if the source is specified as an 8-bit device and the destination as a 32-bit device, there will be four reads as necessary from the 8-bit source to fill the Temporary Register. Then the 82370 will write the 32-bit contents to the destination in two cycles of 16-bit each. This cycle will repeat until the process is terminated or suspended.

With Two-Cycle transfers, the devices that the 82370 accesses can reside at any address within I/O or memory space. The device must be able to decode the byte-enables (BLE#, BHE#). Also, if the device cannot accept data in byte quantities, the programmer must take care not to allow the DMA Controller to access the device on any address other than the device boundary.

3.3.6.3 Data Path Width and Data Transfer Rate Considerations

The number of bus cycles used to transfer a single "word" of data is affected by whether the Two-Cycle or the Fly-By (Single-Cycle) transfer method is used.

The number of bus cycles used to transfer data directly affects the data transfer rate. Inefficient use of bus cycles will decrease the effective data transfer rate that can be obtained. Generally, the data transfer rate is halved by using Two-Cycle transfers instead of Fly-By transfers.

The choice of data path widths of both Target and Requester affects the data transfer rate also. During each bus cycle, the largest pieces of data possible should be transferred.

The data path width of the devices to be accessed must be programmed into the DMA controller. The 82370 defaults after reset to 8-bit-to-8-bit data transfers, but the Target and Requester can have different data path widths, independent of each other and independent of the other channels. Since this is a software programmable function, more discussion of the uses of this feature are found in the section on programming.

3.3.6.4 Read, Write and Verify Cycles

Three different bus cycles types may be used in a data transfer. They are the Read, Write and Verify cycles. These cycle types dictate the way in which the 82370 operates on the data to be transferred.

A Read Cycle transfers data from the Target to the Requester. A Write Cycle transfers data from the Requester to the target. In a Fly-By transfer, the address and bus status signals indicate the access (read of write) to the Target; the access to the Requester is assumed to be the opposite.

The Verify Cycle is used to perform a data read only. No write access is indicated or assumed in a Verify Cycle. The Verify Cycle is useful for validating block fill operations. An external comparator must be provided to do any comparisons on the data read.

3.4 Bus Arbitration and Handshaking

Figure 3-14 shows the flow of events in the DMA request arbitration process. The arbitration sequence starts when the Requester asserts a DREQn (or DMA service is requested by software). Figure 3-15 shows the timing of the sequence of events following a DMA request. This sequence is executed for each channel that is activated. The DREQn signal can be replaced by a software DMA channel request with no change in the sequence.

After the Requester asserts the service request, the 82370 will request control of the bus via the HOLD signal. The 82370 will always assert the HOLD signal one bus state after the service request is asserted. The 80376 responds by asserting the HLDA signal, thus releasing control of the bus to the 82370 DMA Controller.

Priority of pending DMA service requests is arbitrated during the first state after HLDA is asserted by the 80376. The next state will be the beginning of the first transfer access of the highest priority process.

When the 82370 DMA Controller is finished with its current bus activity, it returns control of the bus to the host processor. This is done by driving the HOLD signal inactive. The 82370 does not drive any address or data bus signals after HOLD goes low. It enters the Slave Mode until another DMA process is requested. The processor acknowledges that it has regained control of the bus by forcing the HLDA signal inactive. Note that the 82370's DMA Controller will not re-request control of the bus until the entire HOLD/HLDA handshake sequence is complete.

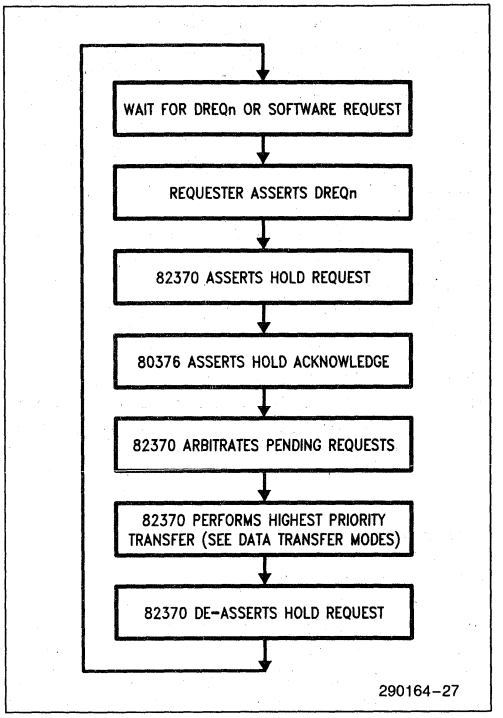
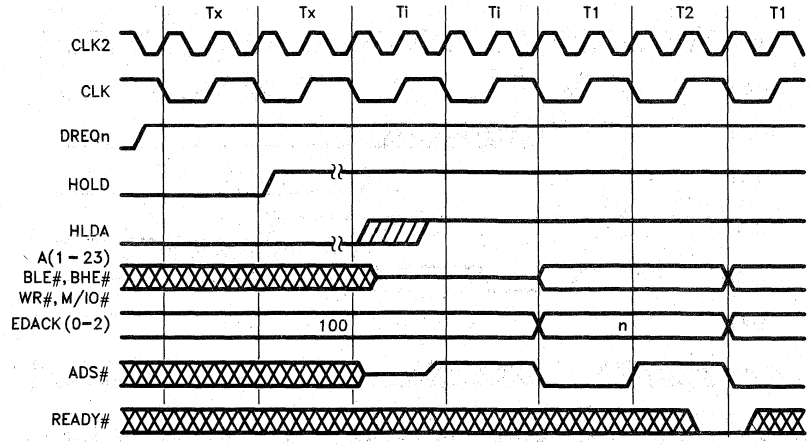


Figure 3-14. Bus Arbitration and DMA Sequence

The 82370 DMA Controller will terminate a current DMA process for one of three reasons: expired byte count, end-of-process command (EOP# activated) from a peripheral, or deactivated DMA request signal. In each case, the controller will de-assert HOLD immediately after completing the data transfer in progress. These three methods of process termination are illustrated in Figures 3-16, 3-19 and 3-18, respectively.

An expired byte count indicates that the current process is complete as programmed and the channel has no further transfers to process. The channel must be restarted according to the currently programmed Buffer Transfer Mode, or reprogrammed completely, including a new Buffer Transfer Mode.



NOTE:
290164-28
Channel priority resolution takes place during the bus state before HOLDA is asserted, allowing the DMA Controller to respond to HLDA without extra idle bus states.

Figure 3-15. Beginning of a DMA process

If the peripheral activates the EOP# signal, it is indicating that it will not accept or deliver any more data for the current buffer. The 82370 DMA Controller considers this as a completion of the channel's current process and interprets the condition the same way as if the byte count expired.

The action taken by the 82370 DMA Controller in response to a de-activated DREQn signal depends on the Data Transfer Mode of the channel. In the Demand Mode, data transfers will take place as long as the DREQn is active and the byte count has not expired. In the Block Mode, the controller will complete the entire block transfer without relinquishing the bus, even if DREQn goes inactive before the transfer is complete. In the Single Mode, the controller will execute single data transfers, relinquishing the bus between each transfer, as long as DREQn is active.

Normal termination of a DMA process due to expiration of the byte count (Terminal Count—TC) is shown if Figure 3-16. The condition of DREQn is ignored until after the process is terminated. If the channel is programmed to auto-initialize, HOLD will be held active for an additional seven clock cycles while the auto-initialization takes place.

Table 3-3 shows the DMA channel activity due to EOP# or Byte Count expiring (Terminal Count).

Table 3-3. DMA Channel Activity Due to Terminal Count or External EOP#

Buffer Process	Single or Chaining-Base Empty		Auto-Initialize		Chaining-Base Loaded	
EVENT						
Terminal Count	True	X	True	X	True	X
EOP#	X	0	X	0	X	0
RESULTS						
Current Registers			Load	Load	Load	Load
Channel Mask	Set	Set				
EOP# Output	0	X	0	X	1	X
Terminal Count Status	Set	Set	Set	Set		
Software Request	CLR	CLR	CLR	CLR		

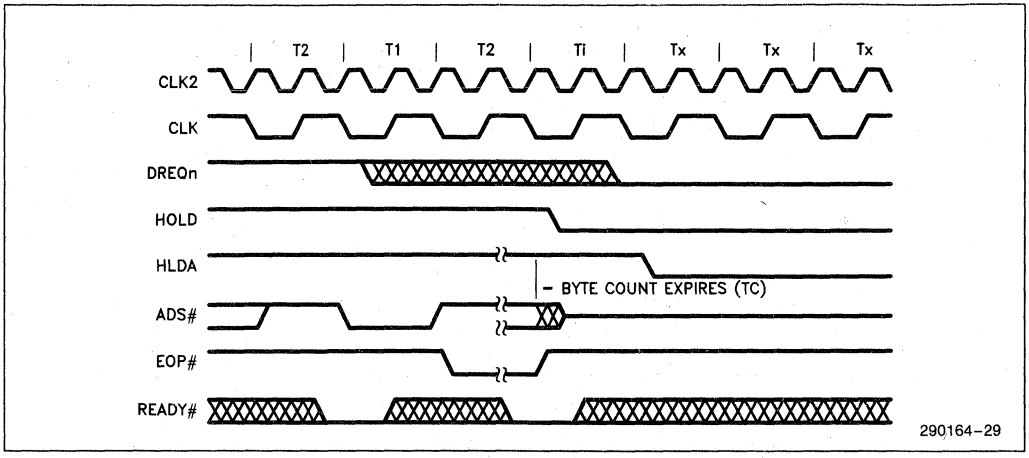


Figure 3-16. Termination of a DMA Process Due to Expiration of Current Byte Count

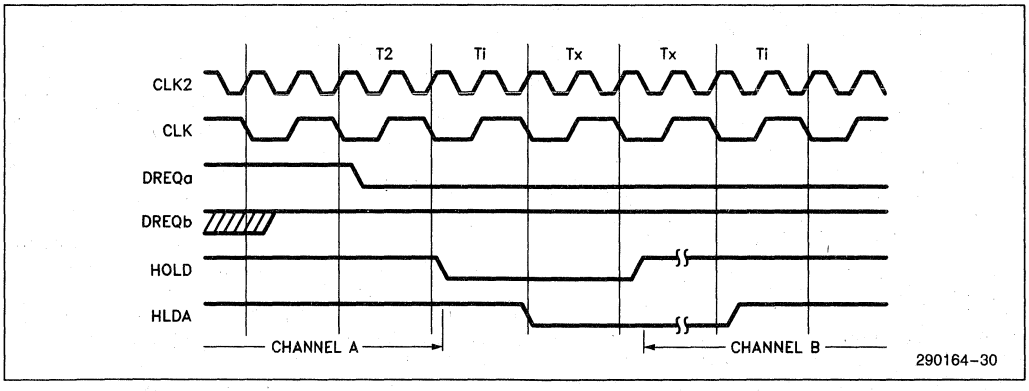


Figure 3-17. Switching between Active DMA Channels

The 82370 always relinquishes control of the bus between channel services. This allows the hardware designer the flexibility to externally arbitrate bus hold requests, if desired. If another DMA request is pending when a higher priority channel service is completed, the 82370 will relinquish the bus until the hold acknowledge is inactive. One bus state after the HLDA signal goes inactive, the 82370 will assert HOLD again. This is illustrated in Figure 3-17.

3.4.1 SYNCHRONOUS AND ASYNCHRONOUS SAMPLING OF DREQn AND EOP#

As an indicator that a DMA service is to be started, DREQn is always sampled asynchronous. It is sampled at the beginning of a bus state and acted upon at the end of the state. Figure 3-15 illustrates the start of a DMA process due to a DREQn input.

The DREQn and EOP# inputs can be programmed to be sampled either synchronously or asynchronously to signal the end of a transfer.

The synchronous mode affords the Requester one bus state of extra time to react to an access. This means the Requester can terminate a process on the current access, without losing any data. The asynchronous mode requires that the input signal be presented prior to the beginning of the last state of the Requester access.

The timing relationships of the DREQn and EOP# signals to the termination of a DMA transfer are shown in Figures 3-18 and 3-19. Figure 3-18 shows the termination of a DMA transfer due to inactive DREQn. Figure 3-19 shows the termination of a DMA process due to an active EOP# input.

In the Synchronous Mode, DREQn and EOP# are sampled at the end of the last state of every Requester data transfer cycle. If EOP# is active or DREQn is inactive at this time, the 82370 recognizes this access to the Requester as the last transfer. At this point, the 82370 completes the transfer in progress, if necessary, and returns bus control to the host.

In the asynchronous mode, the inputs are sampled at the beginning of every state of a Requester access. The 82370 waits until the end of the state to act on the input.

DREQn and EOP# are sampled at the latest possible time when the 82370 can determine if another transfer is required. In the Synchronous Mode, DREQn and EOP# are sampled on the trailing edge of the last bus state before another data access cycle begins. The Asynchronous Mode requires that the signals be valid one clock cycle earlier.

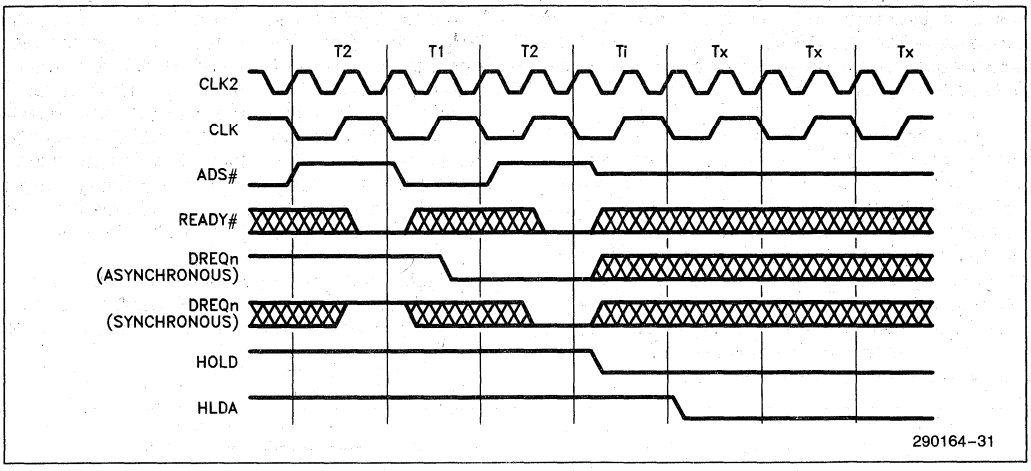


Figure 3-18. Termination of a DMA Process due to De-Asserting DREQn

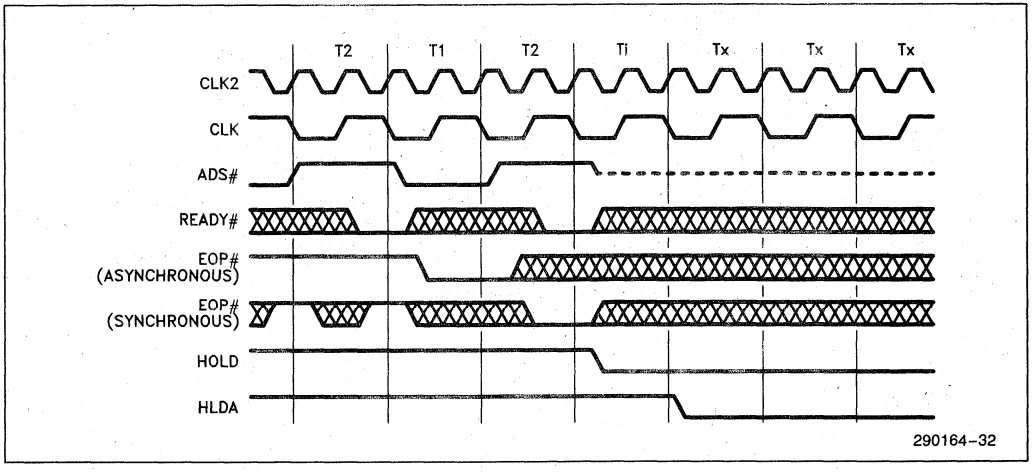


Figure 3-19. Termination of a DMA Process due to an External EOP#

While in the Pipeline Mode, if the NA# signal is sampled active during a transfer, the end of the state where NA# was sampled active is when the 82370 decides whether to commit to another transfer. The device must de-assert DREQn or assert EOP# before NA# is asserted, otherwise the 82370 will commit to another, possibly undesired, transfer.

Synchronous DREQn and EOP# sampling allows the peripheral to prevent the next transfer from occurring by de-activating DREQn or asserting EOP# during the current Requester access, before the 82370 DMA Controller commits itself to another transfer. The DMA Controller will not perform the next transfer if it has not already begun the bus cycle. Asynchronous sampling allows less stringent timing requirements than the Synchronous Mode, but requires that the DREQn signal be valid at the beginning of the next to last bus state of the current Requester access.

Using the Asynchronous Mode with zero wait states can be very difficult. Since the addresses and control signals are driven by the 82370 near half-way through the first bus state of a transfer, and the Asynchronous Mode requires that DREQn be inactive before the end of the state, the peripheral being accessed is required to present DREQn only a few nanoseconds after the control information is available. This means that the peripheral's control logic must be extremely fast (practically non-causal). An alternative is the Synchronous Mode.

3.4.2 ARBITRATION OF CASCADED MASTER REQUESTS

The Cascade Mode allows another DMA-type device to share the bus by arbitrating its bus accesses with the 82370's. Seven of the eight DMA channels (0–3 and 5–7) can be connected to a cascaded device. The cascaded device requests bus control through the DREQn line of the channel which is programmed to operate in Cascade Mode. Bus hold acknowledge is signalled to the cascaded device through the EDACK lines. When the EDACK lines are active with the code for the requested cascade channel, the bus is available to the cascaded master device.

A cascade cycle begins the same way a regular DMA cycle begins. The requesting bus master asserts the DREQn line on the 82370. This bus control request is arbitrated as any other DMA request would be. If any channel receives a DMA request, the 82370 requests control of the bus. When the host acknowledges that it has released bus control, the 82370 acknowledges to the requesting master that it may access the bus. The 82370 enters an idle state until the new master relinquishes control.

A cascade cycle will be terminated by one of two events: DREQn going inactive, or HLDA going inactive. The normal way to terminate the cascade cycle

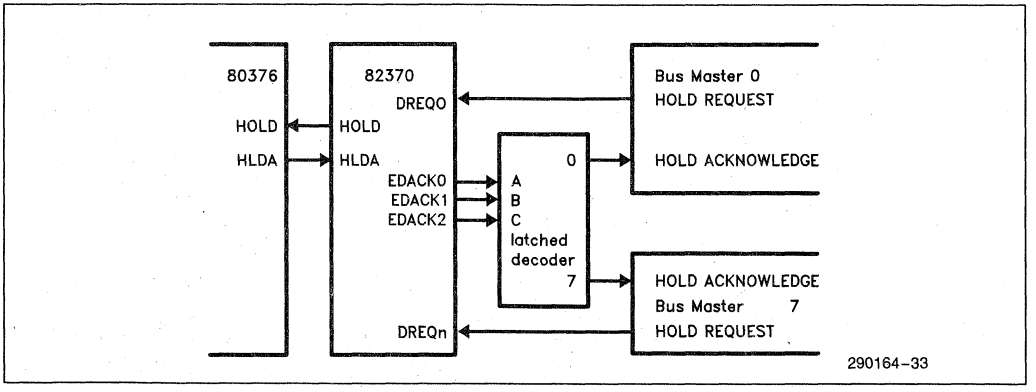


Figure 3-20. Cascaded Bus Master

is for the cascaded master to drop the DREQn signal. Figure 3-21 shows the two cascade cycle termination sequences.

The Refresh Controller may interrupt the cascaded master to perform a refresh cycle. If this occurs, the 82370 DMA Controller will de-assert the EDACK signal (hold acknowledge to cascaded master) and wait for the cascaded master to remove its hold request. When the 82370 regains bus control, it will perform the refresh cycle in its normal fashion. After the refresh cycle has been completed, and if the cascaded device has re-asserted its request, the 82370 will return control to the cascaded master which was interrupted.

The 82370 assumes that it is the only device monitoring the HLDA signal. If the system designer wishes to place other devices on the bus as bus masters, the HLDA from the processor must be intercepted before presenting it to the 82370. Using the Cascade capabililty of the 82370 DMA Controller offers a much better solution.

3.4.3 ARBITRATION OF REFRESH REQUESTS

The arbitration of refresh requests by the DRAM Refresh Controller is slightly different from normal DMA channel request arbitration. The 82370 DRAM Refresh Controller always has the highest priority of any DMA process. It also can interrupt a process in progress. Two types of processes in progress may be encountered: normal DMA, and bus master cascade.

In the event of a refresh request during a normal DMA process, the DMA Controller will complete the data transfer in progress and then execute the refresh cycle before continuing with the current DMA process. The priority of the interrupted process is not lost. If the data transfer cycle interrupted by the Refresh Controller is the last of a DMA process, the refresh cycle will always be executed before control of the bus is transferred back to the host.

When the Refresh Controller request occurs during a cascade cycle, the Refresh Controller must be assured that the cascaded master device has relinquished control of the bus before it can execute the refresh cycle. To do this, the DMA Controller drops the EDACK signal to the cascaded master and waits for the corresponding DREQn input to go inactive. By dropping the DREQn signal, the cascaded master relinquishes the bus. The Refresh Controller then performs the refresh cycle. Control of the bus is returned to the cascaded master if DREQn returns to an active state before the end of the refresh cycle, otherwise control is passed to the processor and the cascaded master loses its priority.

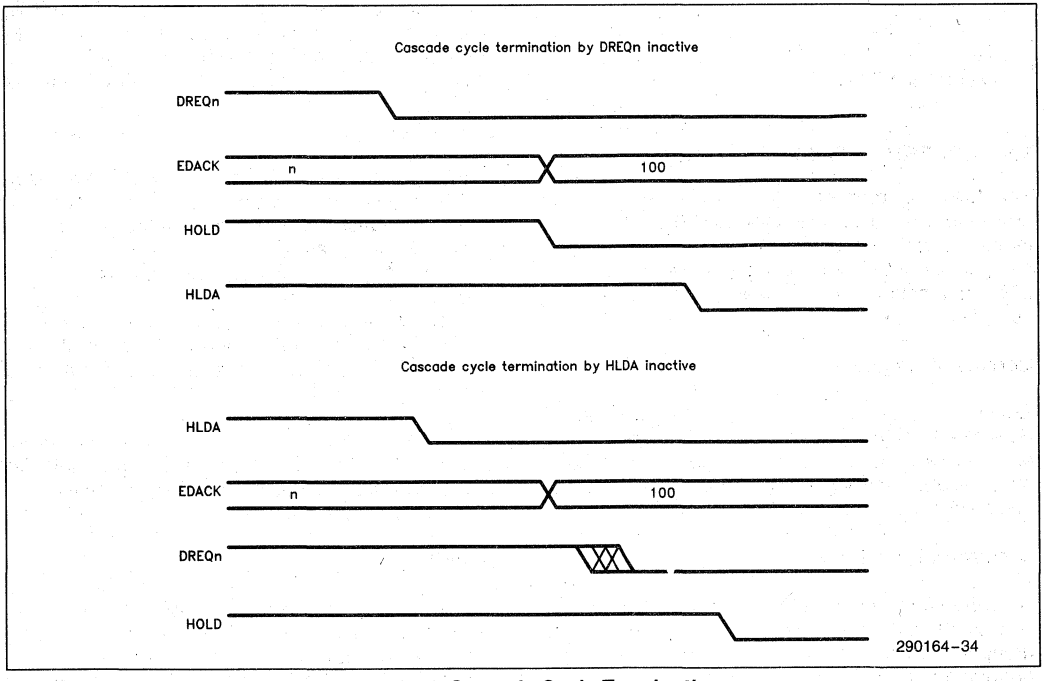


Figure 3-21. Cascade Cycle Termination

3.5 DMA Controller Register Overview

The 82370 DMA Controller contains 44 registers which are accessable to the host processor. Twenty-four of these registers contain the device addresses and data counts for the individual DMA channels (three per channel). The remaining registers are control and status registers for initiating and monitoring the operation of the 82370 DMA Controller. Table 3-4 lists the DMA Controller's registers and their accessability.

Table 3-4. DMA Controller Registers

Register Name	Access
Control/Status Registers—one each per group	
Command Register I	write only
Command Register II	write only
Mode Register I	write only
Mode Register II	write only
Software Request Register	read/write
Mask Set-Reset Register	write only
Mask Read-Write Register	read/write
Status Register	read only
Bus Size Register	write only
Chaining Register	read/write
Channel Registers—one each per channel	
Base Target Address	write only
Current Target Address	read only
Base Requester Address	write only
Current Requester Address	read only
Base Byte Count	write only
Current Byte Count	read only

3.5.1 CONTROL/STATUS REGISTERS

The following registers are available to the host processor for programming the 82370 DMA Controller into its various modes and for checking the operating status of the DMA processes. Each set of four DMA channels has one of each of these registers associated with it.

Command Register I

Enables or disables the DMA channel as a group. Sets the Priority Mode (Fixed or Rotating) of the group. This write-only register is cleared by a hardware reset, defaulting to all channels enabled and Fixed Priority Mode.

Command Register II

Sets the sampling mode of the DREQn and EOP# inputs. Also sets the lowest priority channel for the group in the Fixed Priority Mode. The functions programmed through Command Register II default after a hardware reset to: asynchronous DREQn and EOP#, and channels 3 and 7 lowest priority.

Mode Registers I

Mode Register I is identical in function to the Mode register of the 8237A. It programs the following functions for an individually selected channel:

Type of Transfer—read, write, verify
Auto-Initialize—enable or disable
Target Address Count—increment or decrement
Data Transfer Mode—demand, single, block, cascade

Mode Register I functions default to the following after reset: verify transfer, Auto-Initialize disabled, Increment Target address, Demand Mode.

Mode Register II

Programs the following functions for an individually selected channel:

Target Address Hold—enable or disable
Requester Address Count—increment or decrement
Requester Address Hold—enable or disable
Target Device Type—I/O or Memory
Requester Device Type—I/O or Memory
Transfer Cycles—Two-Cycle or Fly-By

Mode Register II functions are defined as follows after a hardware reset: Disable Target Address Hold, Increment Requester Address, Target (and Requester) in memory, Fly-By Transfer Cycles. Note: Requester Device Type ignored in Fly-By Transfers.

Software Request Register

The DMA Controller can respond to service requests which are initiated by software. Each channel has an internal request status bit associated with it. The host processor can write to this register to set or reset the request bit of a selected channel.

The status of a group's software DMA service requests can be read from this register as well. Each status bit is cleared upon Terminal Count or external EOP#.

The software DMA requests are non-maskable and subject to priority arbitration with all other software and hardware requests. The entire register is cleared by a hardware reset.

Mask Registers

Each channel has associated with it a mask bit which can be set/reset to disable/enable that channel. Two methods are available for setting and clearing the mask bits. The Mask Set/Reset Register is a

write-only register which allows the host to select an individual channel and either set or reset the mask bit for that channel only. The Mask Read/Write Register is available for reading the mask bit status and for writing mask bits in groups of four.

The mask bits of a group may be cleared in one step by executing the Clear Mask Command. See the DMA Programming section for details. A hardware reset sets all of the channel mask bits, disabling all channels.

Status Register

The Status register is a read-only register which contains the Terminal Count (TC) and Service Request status for a group. Four bits indicate the TC status and four bits indicate the hardware request status for the four channels in the group. The TC bits are set when the Byte Count expires, or when and external EOP# is asserted. These bits are cleared by reading from the Status Register. The Service Request bit for a channel indicates when there is a hardware DMA request (DREQn) asserted for that channel. When the request has been removed, the bit is cleared.

Bus Size Register

This write-only register is used to define the bus size of the Target and Requester of a selected channel. The bus sizes programmed will be used to dictate the sizes of the data paths accessed when the DMA channel is active. The values programmed into this register affect the operation of the Temporary Register. When 32-bit bus width is programmed, the 82370 DMA Controller will access the device twice through its 16-bit external Data Bus to perform a 32-bit data transfer. Any byte-assembly required to make the transfers using the specified data path widths will be done in the Temporary Register. The Bus Size register of the Target is used as an increment/decrement value for the Byte Counter and Target Address when in the Fly-By Mode. Upon reset, all channels default to 8-bit Targets and 8-bit Requesters.

Chaining Register

As a command or write register, the Chaining register is used to enable or disable the Chaining Mode for a selected channel. Chaining can either be disabled or enabled for an individual channel, independently of the Chaining Mode status of other channels. After a hardware reset, all channels default to Chaining disabled.

When read by the host, the Chaining Register provides the status of the Chaining Interrupt of each of the channels. These interrupt status bits are cleared when the new buffer information has been loaded.

3.5.2 CHANNEL REGISTERS

Each channel has three individually programmable registers necessary for the DMA process; they are the Base Byte Count, Base Target Address, and Base Requester Address registers. The 24-bit Base Byte Count register contains the number of bytes to be transferred by the channel. The 24-bit Base Target Address Register contains the beginning address (memory or I/O) of the Target device. The 24-bit Base Requester Address register contains the base address (memory or I/O) of the device which is to request DMA service.

Three more registers for each DMA channel exist within the DMA Controller which are directly related to the registers mentioned above. These registers contain the current status of the DMA process. They are the Current Byte Count register, the Current Target Address, and the Current Requester Address. It is these registers which are manipulated (incremented, decremented, or held constant) by the 82370 DMA Controller during the DMA process. The Current registers are loaded from the Base registers at the beginning of a DMA process.

The Base registers are loaded when the host processor writes to the respective channel register addresses. Depending on the mode in which the channel is operating, the Current registers are typically loaded in the same operation. Reading from the channel register addresses yields the contents of the corresponding Current register.

To maintain compatibility with software which accesses an 8237A, a Byte Pointer Flip-Flop is used to control access to the upper and lower bytes of some words of the Channel Registers. These words are accessed as byte pairs at single port addresses. The Byte Pointer Flip-Flop acts as a one-bit pointer which is toggled each time a qualifying Channel Register byte is accessed.

It always points to the next logical byte to be accessed of a pair of bytes.

The Channel registers are arranged as pairs of words, each pair with its own port address. Addressing the port with the Byte Pointer Flip-Flop reset accesses the least significant byte of the pair. The most significant byte is accessed when the Byte Pointer is set.

For compatibility with existing 8237A designs, there is one exception to the above statements about the Byte Pointer Flip-Flop. The third byte (bits 16-23) of the Target Address is accessed through its own port address. The Byte Pointer Flip-Flop is not affected by any accesses to this byte.

The upper eight bits of the Byte Count Register are cleared when the least significant byte of the register is loaded. This provides compatibility with software which accesses an 8237A. The 8237A has 16-bit Byte Count Registers.

NOTE:
The 82370 is a subset of the Intel 82380 32-bit DMA Controller with Integrated System Peripherals.

Although the 82370 has 24 address bits externally, the programming model is actually a full 32 bits wide. For this reason, there are some "hidden" DMA registers in the 82370 register set. These hidden registers correspond to what would be A24–A31 in a 32-bit system.

Think of the 82370 addresses as though they were 32 bits wide, with only the lower 24 bits available externally.

This should be of concern in two areas:

1. Understanding the Byte Pointer Flip Flop
2. Removing the IRQ1 Chaining Interrupt

The byte pointer flip flop will behave as though the hidden upper address bits were accessible.

The IRQ1 Chaining Interrupt will be removed only when the hidden upper address bits are programmed. You will note that since the hidden upper address bits are not available externally, the **value** you program into the registers is not important. The **act** of programming the hidden register is critical in removing the IRQ1 Chaining interrupt for a DMA channel.

The port assignments for these hidden upper address bits come directly from the port assignments of the Intel 82380. For your convenience, those port definitions have been included in this data sheet in section 3.7.

3.5.3 TEMPORARY REGISTERS

Each channel has a 32-bit Temporary Register used for temporary data storage during two-cycle DMA transfers. It is this register in which any necessary byte assembly and disassembly of non-aligned data is performed. Figure 3-22 shows how a block of data will be moved between memory locations with different boundaries. Note that the order of the data does not change.

If the destination is the Requester and an early process termination has been indicated by the EOP# signal or DREQn inactive in the Demand Mode, the Temporary Register is not affected. If data remains in the Temporary Register due to differences in data path widths of the Target and Requester, it will not be transferred or otherwise lost, but will be stored for later transfer.

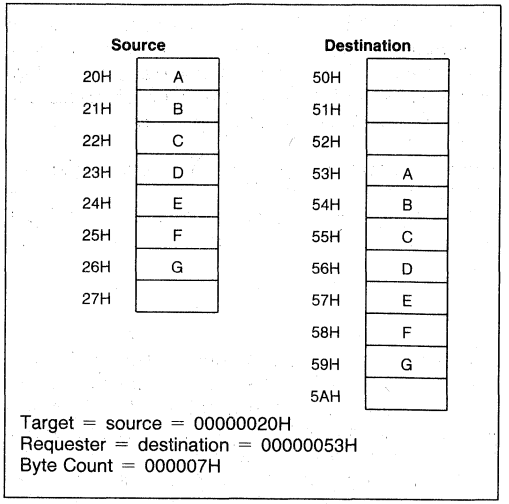


Figure 3-22. Transfer of data between memory locations with different boundaries. This will be the result, independent of data path width.

If the destination is the Target and the EOP# signal is sensed active during the Requester access of a transfer, the DMA Controller will complete the transfer by sending to the Target whatever information is in the Temporary Register at the time of process termination. This implies that the Target could be accessed with partial data in two accesses. For this reason it is advisable to have an I/O device designated as a Requester, unless it is capable of handling partial data transfers.

3.6 DMA Controller Programming

Programming a DMA Channel to perform a needed DMA function is in general a four step process. First the global attributes of the DMA Controller are programmed via the two Command Registers. These global attributes include: priority levels, channel group enables, priority mode, and DREQn/EOP# input sampling.

The second step involves setting the operating modes of the particular channel. The Mode Registers are used to define the type of transfer and the handshaking modes. The Bus Size Register and Chaining Register may also need to be programmed in this step.

The third step in setting up the channel is to load the Base Registers in accordance with the needs of the operating modes chosen in step two. The Current Registers are automatically loaded from the Base Registers, if required by the Buffer Transfer Mode in

effect. The information loaded and the order in which it is loaded depends on the operating mode. A channel used for cascading, for example, needs no buffer information and this step can be skipped entirely.

The last step is to enable the newly programmed channel using one of the Mask Registers. The channel is then available to perform the desired data transfer. The status of the channel can be observed at any time through the Status Register, Mask Register, Chaining Register, and Software Request register.

Once the channel is programmed and enabled, the DMA process may be initiated in one of two ways, either by a hardware DMA request (DREQn) or a software request (Software Request Register).

Once programmed to a particular Process/Mode configuration, the channel will operate in that configuration until programmed otherwise. For this reason, restarting a channel after the current buffer expires does not require complete reprogramming of the channel. Only those parameters which have changed need to be reprogrammed. The Byte Count Register is always changed and must be reprogrammed. A Target or Requester Address Register which is incremented or decremented should be reprogrammed also.

3.6.1 BUFFER PROCESSES

The Buffer Process is determined by the Auto-Initialize bit of Mode Register I and the Chaining Register. If Auto-Initialize is enabled, Chaining should not be used.

3.6.1.1 Single Buffer Process

The Single Buffer Process is programmed by disabling Chaining via the Chaining Register and programming Mode Register I for non-Auto-Initialize.

3.6.1.2 Buffer Auto-Initialize Process

Setting the Auto-Initialize bit in Mode Register I is all that is necessary to place the channel in this mode. Buffer Auto-Initialize must not be enabled simultaneous to enabling the Buffer Chaining Mode as this will have unpredictable results.

Once the Base Registers are loaded, the channel is ready to be enabled. The channel will reload its Current Registers from the Base Registers each time the Current Buffer expires, either by an expired Byte Count or an external EOP#.

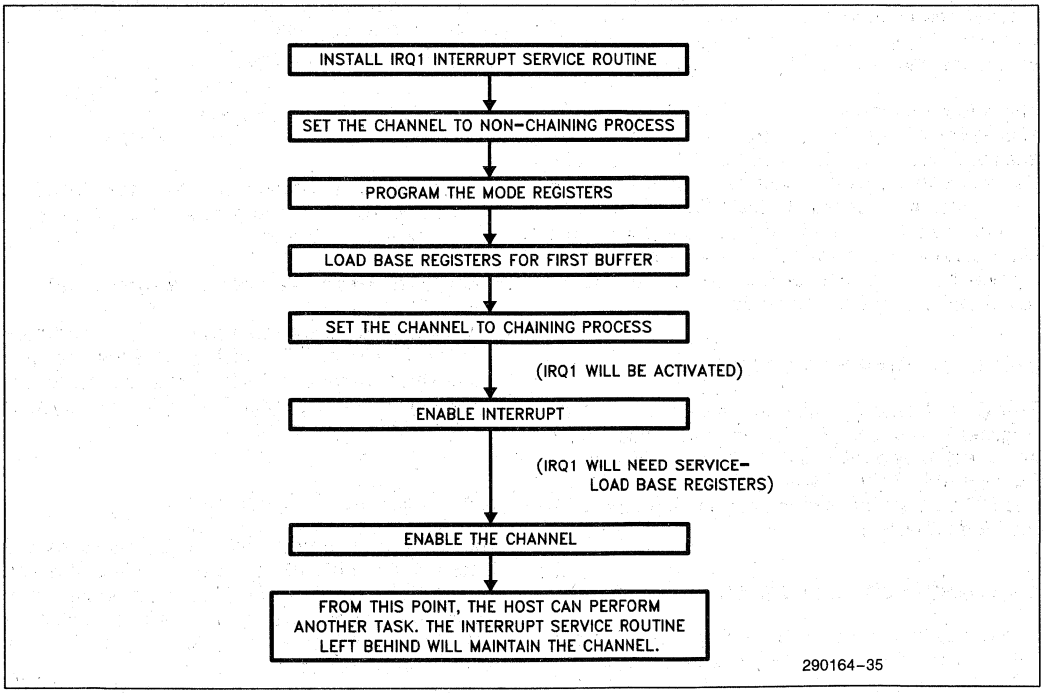


Figure 3-23. Flow of Events in the Buffer Chaining Process

3.6.1.3 Buffer Chaining Process

The Buffer Chaining Process is entered into from the Single Buffer Process. The Mode Registers should be programmed first, with all of the Transfer Modes defined as if the channel were to operate in the Single Buffer Process. The channel's Base Registers are then loaded. When the channel has been set up in this way, and the chaining interrupt service routine is in place, the Chaining Process can be entered by programming the Chaining Register. Figure 3-23 illustrates the Buffer Chaining Process.

An interrupt (IRQ1) will be generated immediately after the Chaining Process is entered, as the channel then perceives the Base Registers as empty and in need of reloading. It is important to have the interrupt service routine in place at the time the Chaining Process is entered into. The interrupt request is removed when the most significant byte of the Base Target Address is loaded.

The interrupt will occur again when the first buffer expires and the Current Registers are loaded from the Base Registers. The cycle continues until the Chaining Process is disabled, or the host fails to respond to IRQ1 before the Current Buffer expires.

Exiting the Chaining Process can be done by resetting the Chaining Mode Register. If an interrupt is pending for the channel when the Chaining Register is reset, the interrupt request will be removed. The Chaining Process can be temporarily disabled by setting the channel's Mask bit in the Mask Register.

The interrupt service routine for IRQ1 has the responsibility of reloading the Base Registers as necessary. It should check the status of the channel to determine the cause of channel expiration, etc. It should also have access to operating system information regarding the channel, if any exists. The IRQ1 service routine should be capable of determining whether the chain should be continued or terminated and act on that information.

3.6.2 DATA TRANSFER MODES

The Data Transfer Modes are selected via Mode Register I. The Demand, Single, and Block Modes are selected by bits D6 and D7. The individual transfer type (Fly-By vs Two-Cycle, Read-Write-Verify, and I/O vs Memory) is programmed through both of the Mode registers.

3.6.3 CASCADED BUS MASTERS

The Cascade Mode is set by writing ones to D7 and D6 of Mode Register I. When a channel is programmed to operate in the Cascade Mode, all of the other modes associated with Mode Registers I and II are ignored. The priority and DREQn/EOP# definitions of the Command Registers will have the same effect on the channel's operation as any other mode.

3.6.4 SOFTWARE COMMANDS

There are five port addresses which, when written to, command certain operations to be performed by the 82370 DMA Controller. The data written to these locations is not of consequence, writing to the location is all that is necessary to command the 82370 to perform the indicated function. Following are descriptions of the command functions.

Clear Byte Pointer Flip-Flop—Location 000CH

Resets the Byte Pointer Flip-Flop. This command should be performed at the beginning of any access to the channel registers in order to be assured of beginning at a predictable place in the register programming sequence.

Master Clear—Location 000DH

All DMA functions are set to their default states. This command is the equivalent of a hardware reset to the DMA Controller. Functions other than those in the DMA Controller section of the 82370 are not affected by this command.

Clear Mask Register—Channels 0–3
— Location 000EH
Channels 4–7
— Location 00CEH

This command simultaneously clears the Mask Bits of all channels in the addressed group, enabling all of the channels in the group.

Clear TC Interrupt Request—Location 001EH

This command resets the Terminal Count Interrupt Request Flip-Flop. It is provided to allow the program which made a software DMA request to acknowledge that it has responded to the expiration of the requested channel(s).

3.7 Register Definitions

The following diagrams outline the bit definitions and functions of the 82370 DMA Controller's Status and Control Registers. The function and programming of the registers is covered in the previous section on DMA Controller Programming. An entry of "X" as a bit value indicates "don't care."

Channel Registers (read Current, write Base)

Channel	Register Name	Address (hex)	Byte Pointer	Bits Accessed
Channel 0	Target Address	00	0	0–7
			1	8–15
		87	x	16–23
		10	0	24–31(*)
	Byte Count	01	0	0–7
			1	8–15
		11	0	16–23
	Requester Address	90	0	0–7
			1	8–15
		91	0	16–23
			1	24–31(*)
Channel 1	Target Address	02	0	0–7
			1	8–15
		83	x	16–23
		12	0	24–31(*)
	Byte Count	03	0	0–7
			1	8–15
		13	0	16–23
	Requester Address	92	0	0–7
			1	8–15
		93	0	16–23
			1	24–31(*)
Channel 2	Target Address	04	0	0–7
			1	8–15
		81	x	16–23
		14	0	24–31(*)
	Byte Count	05	0	0–7
			1	8–15
		15	0	16–23
	Requester Address	94	0	0–7
			1	8–15
		95	0	16–23
			1	24–31(*)
Channel 3	Target Address	06	0	0–7
			1	8–15
		82	x	16–23
		16	0	24–31(*)
	Byte Count	07	0	0–7
			1	8–15
		17	0	16–23
	Requester Address	96	0	0–7
			1	8–15
		97	0	16–23
			1	24–31(*)

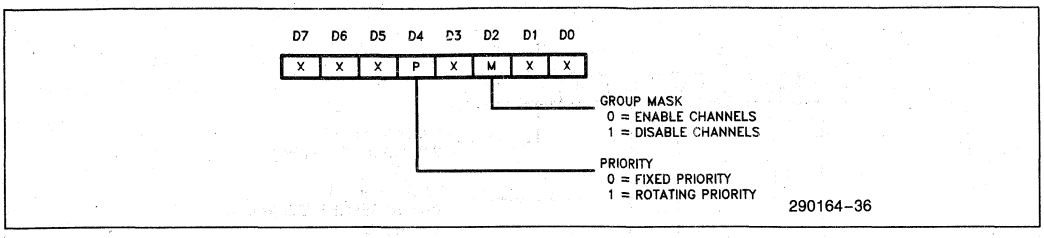
Channel Registers (read Current, write Base) (Continued)

Channel	Register Name	Address (hex)	Byte Pointer	Bits Accessed
Channel 4	Target Address	C0	0	0–7
			1	8–15
		8F	x	16–23
		D0	0	24–31(*)
	Byte Count	C1	0	0–7
			1	8–15
		D1	0	16–23
	Requester Address	98	0	0–7
			1	8–15
		99	0	16–23
			1	24–31(*)
Channel 5	Target Address	C2	0	0–7
			1	8–15
		8B	x	16–23
		D2	0	24–31(*)
	Byte Count	C3	0	0–7
			1	8–15
		D3	0	16–23
	Requester Address	9A	0	0–7
			1	8–15
		9B	0	16–23
			1	24–31(*)
Channel 6	Target Address	C4	0	0–7
			1	8–15
		89	x	16–23
		D4	0	24–31(*)
	Byte Count	C5	0	0–7
			1	8–15
		D5	0	16–23
	Requester Address	9C	0	0–7
			1	8–15
		9D	0	16–23
			1	24–31(*)
Channel 7	Target Address	C6	0	0–7
			1	8–15
		8A	x	16–23
		D6	0	24–31(*)
	Byte Count	C7	0	0–7
			1	8–15
		D7	0	16–23
	Requester Address	9E	0	0–7
			1	8–15
		9F	0	16–23
			1	24–31(*)

NOTE:
(*)These bits are not available externally. You need to be aware of their existence for chaining and Byte Pointer Flip-Flop operations. Please see section 3.5.2 for further details.

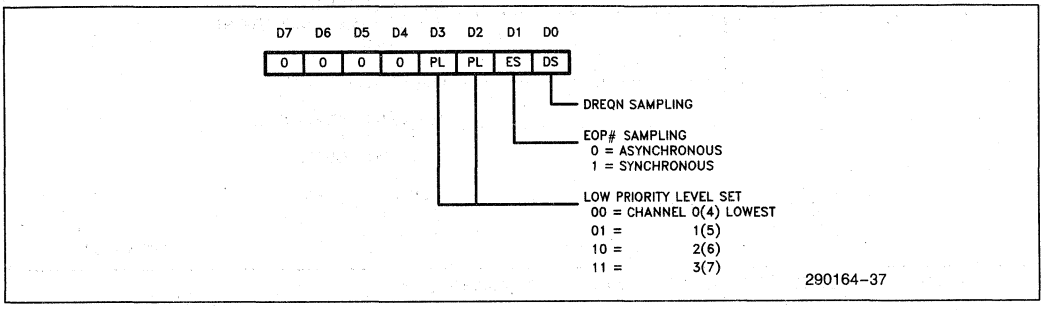
Command Register I (write only)

Port Addresses— Channels 0-3—0008H
Channels 4-7—00C8H



Command Register II (write only)

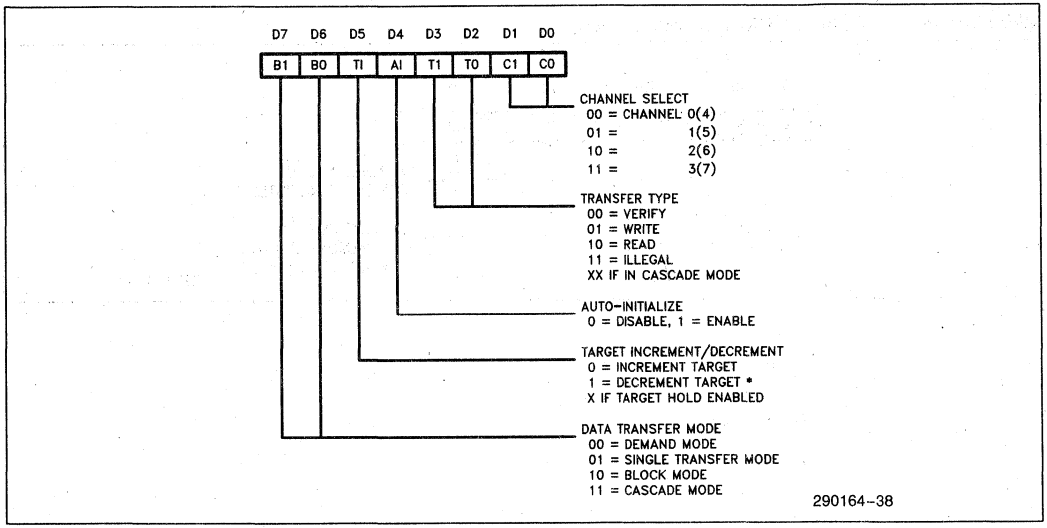
Port Addresses— Channels 0-3—001AH
Channels 4-7—00DAH



5

Mode Register I (write only)

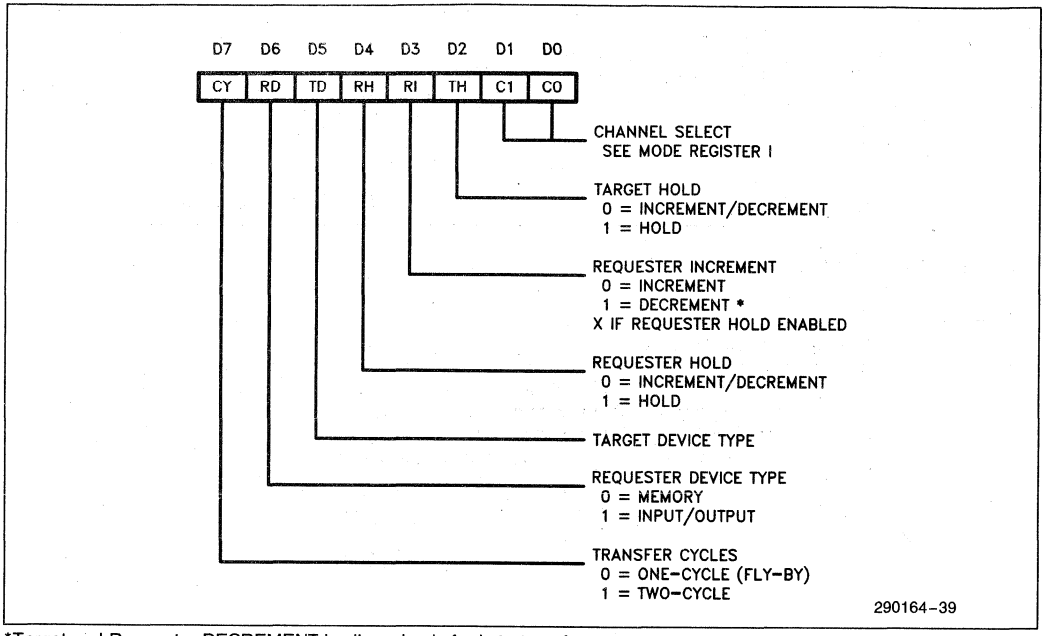
Port Addresses— Channels 0-3—000BH
Channels 4-7—00CBH



*Target and Requester DECREMENT is allowed only for byte transfers.

Mode Register II (write only)

Port Addresses— Channels 0–3—001BH
Channels 4–7—00DBH

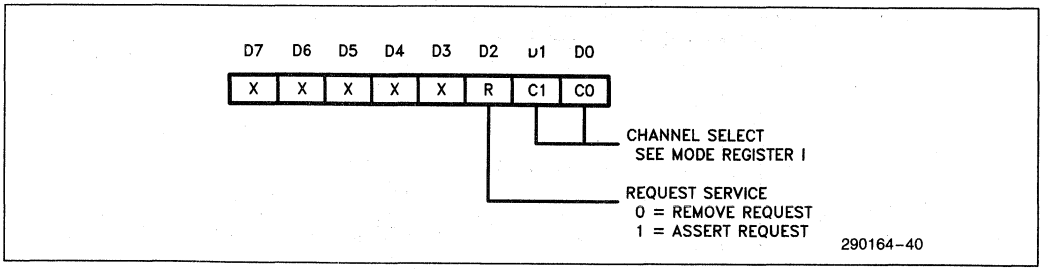


*Target and Requester DECREMENT is allowed only for byte transfers.

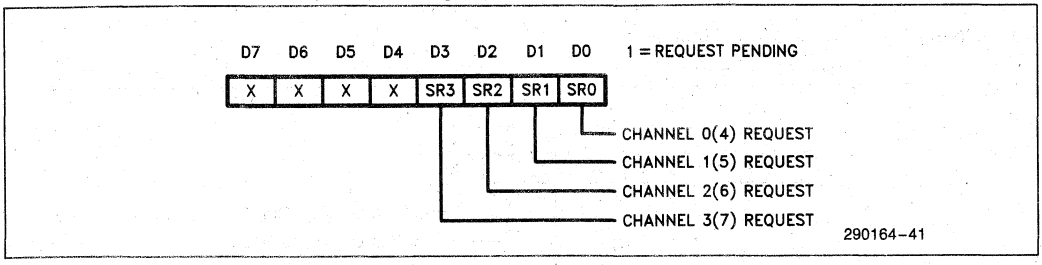
Software Request Register (read/write)

Port Addresses— Channels 0–3—0009H
Channels 4–7—00C9H

Write Format: Software DMA Service Request

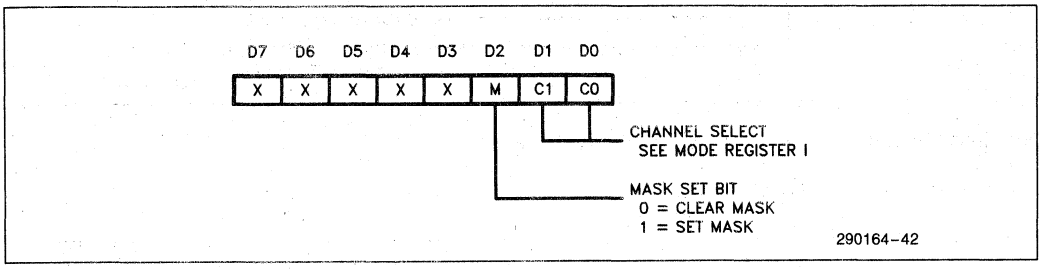


Read Format: Software Requests Pending



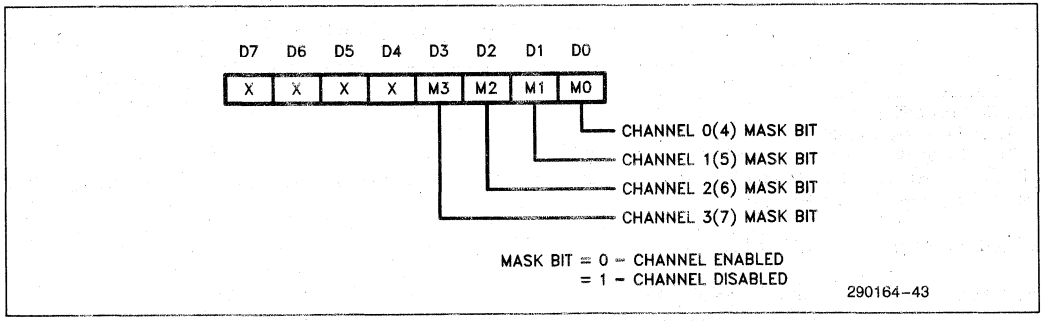
Mask Set/Reset Register **Individual Channel Mask (write only)**

Port Addresses— Channels 0-3—000AH
Channels 4-7—00CAH



Mask Read/Write Register **Group Channel Mask (read/write)**

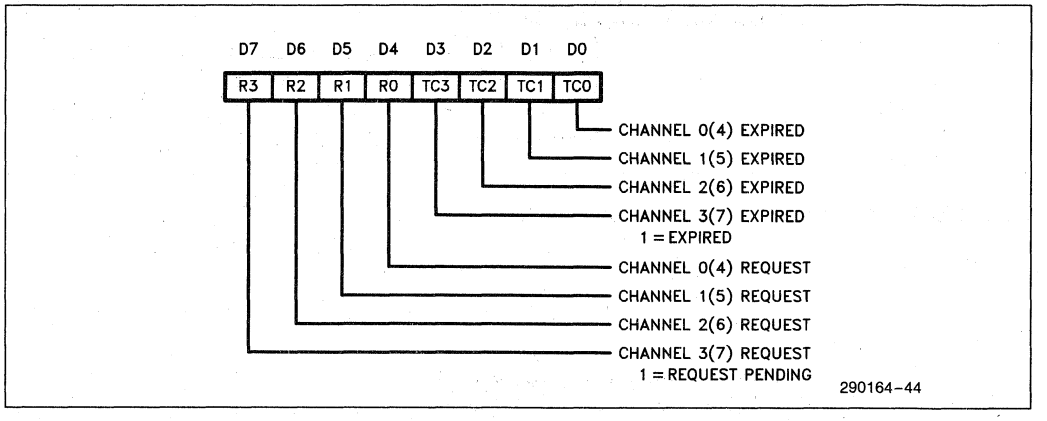
Port Addresses— Channels 0-3—000FH
Channels 4-7—00CFH



5

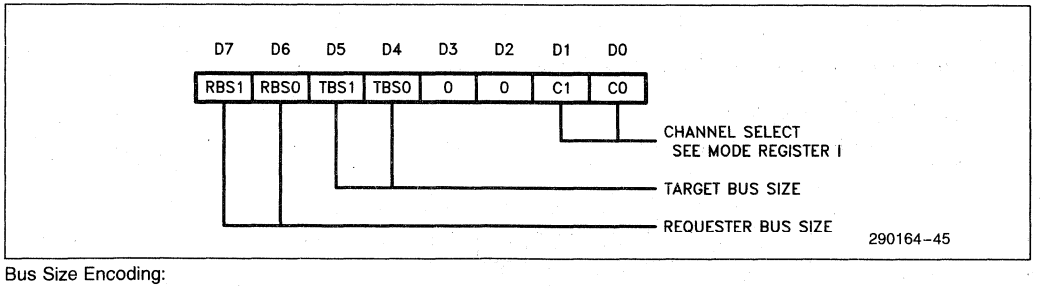
Status Register Channel Process Status (read only)

Port Addresses— Channels 0-3—0008H
Channels 4-7—00C8H



Bus Size Register Set Data Path Width (write only)

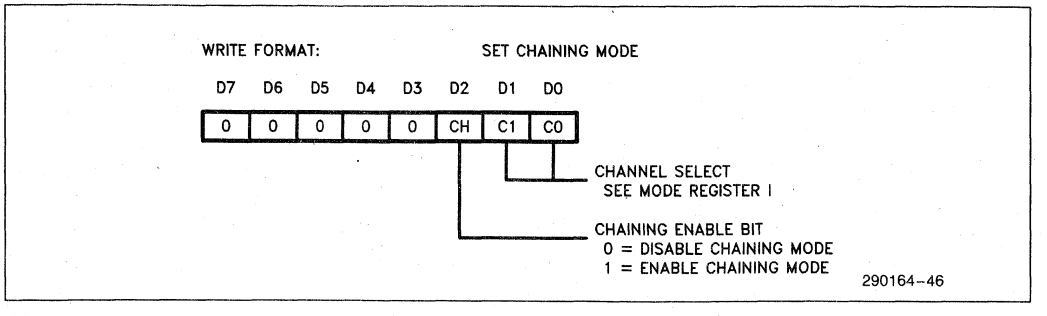
Port Addresses— Channels 0-3—0018H
Channels 4-7—00D8H

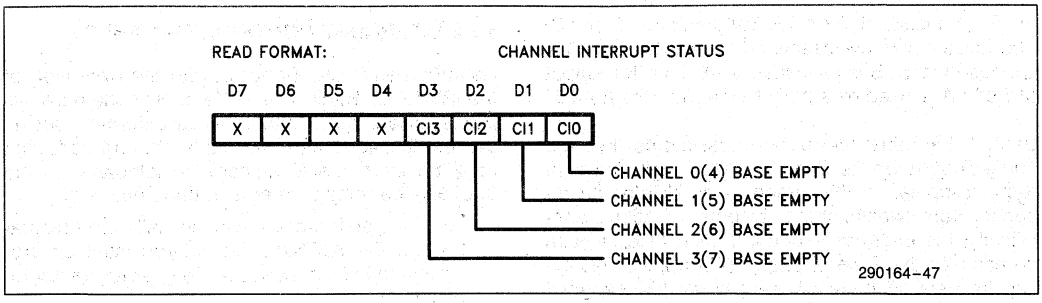


Bus Size Encoding:
00 = Reserved by Intel 10 = 16-bit Bus
01 = 32-bit Bus* 11 = 8-bit Bus
*If programmed as 32-bit bus width, the corresponding device will be accessed in two 16-bit cycles provided that the data is aligned within word boundary.

Chaining Register (read/write)

Port Addresses— Channels 0-3—0019H
Channels 4-7—00D9H





3.8 8237A Compatibility

The register arrangement of the 82370 DMA Controller is a superset of the 8237A DMA Controller. Functionally the 82370 DMA Controller is very different from the 8237A. Most of the functions of the 8237A are performed also by the 82370. The following discussion points out the differences between the 8237A and the 82370.

The 8237A is limited to transfers between I/O and memory only (except in one special case, where two channels can be used to perform memory-to-memory transfers). The 82370 DMA Controller can transfer between any combination of memory and I/O. Several other features of the 8237A are enhanced or expanded in the 82370 and other features are added.

The 8237A is an 8-bit only DMA device. For programming compatibility, all of the 8-bit registers are preserved in the 82370. The 82370 is programmed via 8-bit registers. The address registers in the 82370 are 24-bit registers in order to support the 80376's 24-bit bus. The Byte Count Registers are 24-bit registers, allowing support of larger data blocks than possible with the 8237A.

All of the 8237A's operating modes are supported by the 82370 (except the cumbersome two-channel memory-to-memory transfer). The 82370 performs memory-to-memory transfers using only one channel. The 82370 has the added features of buffer pipelining (Buffer Chaining Process) and programmable priority levels.

The 82370 also adds the feature of address registers for both destination and source. These addresses may be incremented, decremented, or held constant, as required by the application of the individual channel. This allows any combination of destination and source device.

Each DMA channel has associated with it a Target and a Requester. In the 8237A, the Target is the device which can be accessed by the address register, the Requester is the device which is accessed by the DMA Acknowledge signals and must be an I/O device.

4.0 PROGRAMMABLE INTERRUPT CONTROLLER (PIC)

4.1 Functional Description

The 82370 Programmable Interrupt Controller (PIC) consists of three enhanced 82C59A Interrupt Controllers. These three controllers together provide 15 external and 5 internal interrupt request inputs. Each external request input can be cascaded with an additional 82C59A slave controller. This scheme allows the 82370 to support a maximum of 120 (15 x 8) external interrupt request inputs.

Following one or more interrupt requests, the 82370 PIC issues an interrupt signal to the 80376. When the 80376 host processor responds with an interrupt acknowledge signal, the PIC will arbitrate between the pending interrupt requests and place the interrupt vector associated with the highest priority pending request on the data bus.

The major enhancement in the 82370 PIC over the 82C59A is that each of the interrupt request inputs can be individually programmed with its own interrupt vector, allowing more flexibility in interrupt vector mapping.

4.1.1 INTERNAL BLOCK DIAGRAM

The block diagram of the 82370 Programmable Interrupt Controller is shown in Figure 4-1. Internally,

the PIC consists of three 82C59A banks: A, B and C. The three banks are cascaded to one another: C is cascaded to B, B is cascaded to A. The INT output of Bank A is used externally to interrupt the 80376.

Bank A has nine interrupt request inputs (two are unused), and Banks B and C have eight interrupt request inputs. Of the fifteen external interrupt request inputs, two are shared by other functions. Specifically, the Interrupt Request 3 input (IRQ3#) can be used as the Timer 2 output (TOUT2#). This pin can be used in three different ways: IRQ3# input only, TOUT2# output only, or using TOUT2# to generate an IRQ3# interrupt request. Also, the Interrupt Request 9 input (IRQ9#) can be used as DMA Request 4 input (DREQ 4). Typically, only IRQ9# or DREQ4 can be used at a time.

4.1.2 INTERRUPT CONTROLLER BANKS

All three banks are identical, with the exception of the IRQ1.5 on Bank A. Therefore, only one bank will be discussed. In the 82370 PIC, all external requests can be cascaded into and each interrupt controller bank behaves like a master. As compared to the 82C59A, the enhancements in the banks are:

— All interrupt vectors are individually programmable. (In the 82C59A, the vectors must be programmed in eight consecutive interrupt vector locations.)

— The cascade address is provided on the Data Bus (D0-D7). (In the 82C59A, three dedicated control signals (CAS0, CAS1, CAS2) are used for master/slave cascading.)

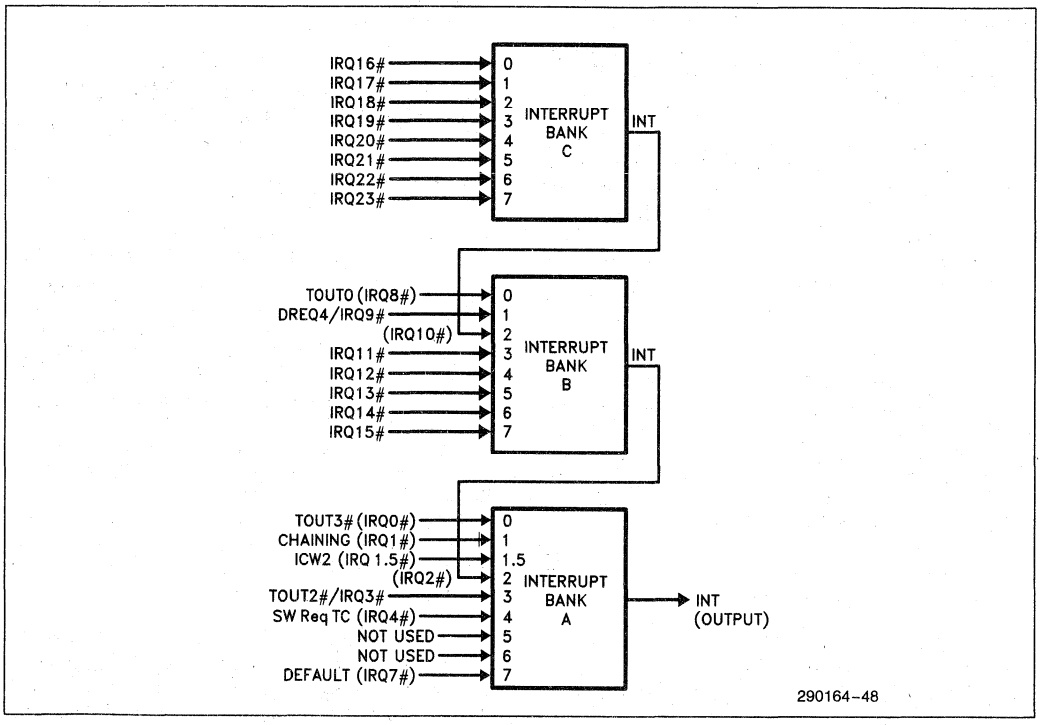


Figure 4-1. Interrupt Controller Block Diagram

The block diagram of a bank is shown in Figure 4-2. As can be seen from this figure, the bank consists of six major blocks: the Interrupt Request Register (IRR), the In-Service Register (ISR), the Interrupt Mask Register (IMR), the Priority Resolver (PR), the Vector Registers (VR), and the Control Logic. The functional description of each block is included below.

INTERRUPT REQUEST (IRR) AND IN-SERVICE REGISTER (ISR)

The interrupts at the Interrupt Request (IRQ) input lines are handled by two registers in cascade, the Interrupt Request Register (IRR) and the In-Service Register (ISR). The IRR is used to store all interrupt levels which are requesting service; and the ISR is used to store all interrupt levels which are being serviced.

PRIORITY RESOLVER (PR)

This logic block determines the priorities of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during an Interrupt Acknowledge cycle.

INTERRUPT MASK REGISTER (IMR)

The IMR stores the bits which mask the interrupt lines to be masked (disabled). The IMR operates on the IRR. Masking of a higher priority input will not affect the interrupt request lines of lower priority.

VECTOR REGISTERS (VR)

This block contains a set of Vector Registers, one for each interrupt request line, to store the pre-programmed interrupt vector number. The corresponding vector number will be driven onto the Data Bus of the 82370 during the Interrupt Acknowledge cycle.

CONTROL LOGIC

The Control Logic coordinates the overall operations of the other internal blocks within the same bank. This logic will drive the Interrupt Output signal (INT) HIGH when one or more unmasked interrupt inputs are active (LOW). The INT output signal goes directly to the 80376 (in bank A) or to another bank to which this bank is cascaded (see Figure 4-1). Also,

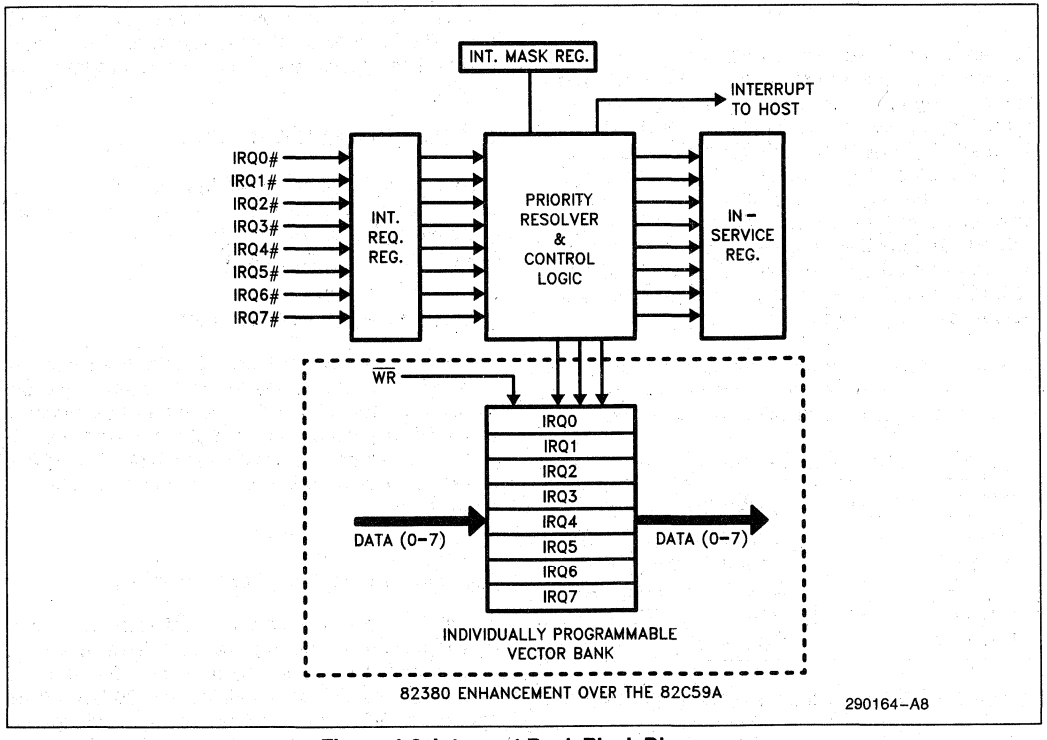


Figure 4-2. Interrupt Bank Block Diagram

this logic will recognize an Interrupt Acknowledge cycle (via M/IO#, D/C# and W/R# signals). During this bus cycle, the Control Logic will enable the corresponding Vector Register to drive the interrupt vector onto the Data Bus.

In bank A, the Control Logic is also responsible for handling the special ICW2 interrupt request input (IRQ1.5).

4.2 Interface Signals

4.2.1 INTERRUPT INPUTS

There are 15 external Interrupt Request inputs and 5 internal Interrupt Requests. The external request inputs are: IRQ3#, IRQ9#, IRQ11# to IRQ23#. They are shown in bold arrows in Figure 4-1. All IRQ inputs are active LOW and they can be programmed (via a control bit in the Initialization Command Word 1 (ICW1)) to be either edge-triggered or level-triggered. In order to be recognized as a valid interrupt request, the interrupt input must be active (LOW) until the first INTA cycle (see Bus Functional Description). Note that all 15 external Interrupt Request inputs have weak internal pull-up resistors.

As mentioned earlier, an 82C59A can be cascaded to each external interrupt input to expand the interrupt capacity to a maximum of 120 levels. Also, two of the interrupt inputs are dual functions: IRQ3# can be used as Timer 2 output (TOUT2#) and IRQ9# can be used as DREQ4 input. IRQ3# is a bidirectional dual function pin. This interrupt request input is wired-OR with the output of Timer 2 (TOUT2#). If only IRQ3# function is to be used, Timer 2 should be programmed so that OUT2 is LOW. Note that TOUT2# can also be used to generate an interrupt request to IRQ3# input.

The five internal interrupt requests serve special system functions. They are shown in Table 4-1. The following paragraphs describe these interrupts.

Table 4-1. 82370 Internal Interrupt Requests

Interrupt Request	Interrupt Source
IRQ0#	Timer 3 Output (TOUT3)
IRQ8#	Timer 0 Output (TOUT0)
IRQ1#	DMA Chaining Request
IRQ4#	DMA Terminal Count
IRQ1.5#	ICW2 Written

TIMER 0 AND TIMER 3 INTERRUPT REQUESTS

IRQ8# and IRQ0# interrupt requests are initiated by the output of Timers 0 and 3, respectively. Each of these requests is generated by an edge-detector flip-flop.

The flip-flops are activated by the following conditions:

Set — Rising edge of timer output (TOUT);

Clear — Interrupt acknowledge for this request; OR Request is masked (disabled); OR Hardware Reset.

CHAINING AND TERMINAL COUNT INTERRUPTS

These interrupt requests are generated by the 82370 DMA Controller. The chaining request (IRQ1#) indicates that the DMA Base Register is not loaded. The Terminal Count request (IRQ4#) indicates that a software DMA request was cleared.

ICW2 INTERRUPT REQUEST

Whenever an Initialization Control Word 2 (ICW2) is written to a Bank, a special ICW2 interrupt request is generated. The interrupt will be cleared when the newly programmed ICW2 Register is read. This interrupt request is in Bank A at level 1.5. This interrupt request is internally ORed with the Cascaded Request from Bank B and is always assigned a higher priority than the Cascaded Request.

This special interrupt is provided to support compatibility with the original 82C59A. A detailed description of this interrupt is discussed in the Programming section.

DEFAULT INTERRUPT (IRQ7#)

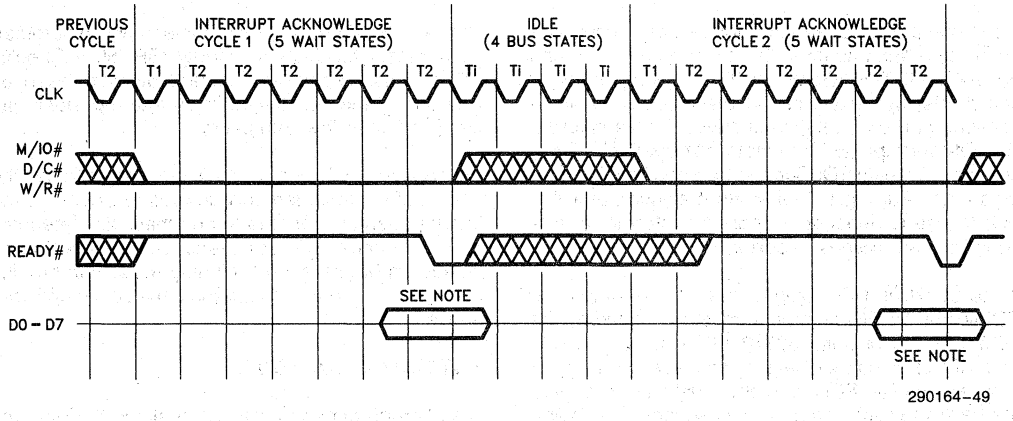
During an Interrupt Acknowledge cycle, if there is no active pending request, the PIC will automatically generate a default vector. This vector corresponds to the IRQ7# vector in bank A.

4.2.2 INTERRUPT OUTPUT (INT)

The INT output pin is taken directly from bank A. This signal should be tied to the Maskable Interrupt Request (INTR) of the 80376. When this signal is active (HIGH), it indicates that one or more internal/external interrupt requests are pending. The 80376 is expected to respond with an interrupt acknowledge cycle.

4.3 Bus Functional Description

The INT output of bank A will be activated as a result of any unmasked interrupt request. This may be a non-cascaded or cascaded request. After the PIC has driven the INT signal HIGH, the 80376 will respond by performing two interrupt acknowledge cycles. The timing diagram in Figure 4-3 shows a typical interrupt acknowledge process between the 82370 and the 80376 CPU.



NOTE:
What is actually driven on the Data Bus depends on if the current interrupt request is a Slave Request.

	INTA Cycle 1	INTA Cycle 2
NON-SLAVE REQUEST	00H	Vector
SLAVE REQUEST	Slave Address	High Impedence*

*Slave will place a vector at this time.

Figure 4-3. Interrupt Acknowledge Cycle

After activating the INT signal, the 82370 monitors the status lines (M/IO#, D/C#, W/R#) and waits for the 80376 to initiate the first interrupt acknowledge cycle. In the 80376 environment, two successive interrupt acknowledge cycles (INTA) marked by M/IO# = LOW, D/C# = LOW, and W/R# = LOW are performed. During the first INTA cycle, the PIC will determine the highest priority request. Assuming this interrupt input has no external Slave Controller cascaded to it, the 82370 will drive the Data Bus with 00H in the first INTA cycle. During the second INTA cycle, the 82370 PIC will drive the Data Bus with the corresponding pre-programmed interrupt vector.

If the PIC determines (from the ICW3) that this interrupt input has an external Slave Controller cascaded to it, it will drive the Data Bus with the specific Slave Cascade Address (instead of 00H) during the first INTA cycle. This Slave Cascade Address is the pre-programmed content in the corresponding Vector Register. This means that no Slave Address should be chosen to be 00H. Note that the Slave Address and Interrupt Vector are different interpretations of the same thing. They are both the contents of the programmable Vector Register. During the second INTA cycle, the Data Bus will be floated so that the external Slave Controller can drive its interrupt vector on the bus. Since the Slave Interrupt Controller resides on the system bus, bus transceiver enable and direction control logic must take this into consideration.

In order to have a successful interrupt service, the interrupt request input must be held valid (LOW) until the beginning of the first interrupt acknowledge cycle. If there is no pending interrupt request when the first INTA cycle is generated, the PIC will generate a default vector, which is the IRQ7 vector (Bank A, level 7).

According to the Bus Cycle definition of the 80376, there will be four Bus Idle States between the two interrupt acknowledge cycles. These idle bus cycles will be initiated by the 80376. Also, during each interrupt acknowledge cycle, the internal Wait State Generator of the 82370 will automatically generate the required number of wait states for internal delays.

4.4 Modes of Operation

A variety of modes and commands are available for controlling the 82370 PIC. All of them are programmable; that is, they may be changed dynamically under software control. In fact, each bank can be programmed individually to operate in different modes. With these modes and commands, many possible configurations are conceivable, giving the user enough versatility for almost any interrupt controlled application.

This section is not intended to show how the 82370 PIC can be programmed. Rather, it describes the operation in different modes.

4.4.1 END-OF-INTERRUPT

Upon completion of an interrupt service routine, the interrupted bank needs to be notified so its ISR can be updated. This allows the PIC to keep track of which interrupt levels are in the process of being serviced and their relative priorities. Three different End-Of-Interrupt (EOI) formats are available. They are: Non-Specific EOI Command, Specific EOI Command, and Automatic EOI Mode. Selection of which EOI to use is dependent upon the interrupt operations the user wishes to perform.

If the 82370 is NOT programmed in the Automatic EOI Mode, an EOI command must be issued by the 80376 to the specific 82370 PIC Controller Bank. Also, if this controller bank is cascaded to another internal bank, an EOI command must also be sent to the bank to which this bank is cascaded. For example, if an interrupt request of Bank C in the 82370 PIC is serviced, an EOI should be written into Bank C, Bank B and Bank A. If the request comes from an external interrupt controller cascaded to Bank C, then an EOI should be written into the external controller as well.

NON-SPECIFIC EOI COMMAND

A Non-Specific EOI command sent from the 80376 lets the 82370 PIC bank know when a service routine has been completed, without specification of its exact interrupt level. The respective interrupt bank automatically determines the interrupt level and resets the correct bit in the ISR.

To take advantage of the Non-Specific EOI, the interrupt bank must be in a mode of operation in which it can predetermine its in-service routine levels. For this reason, the Non-Specific EOI command should only be used when the most recent level acknowledged and serviced is always the highest priority level (i.e. in the Fully Nested Mode structure to be described below). When the interrupt bank receives a Non-Specific EOI command, it simply resets the highest priority ISR bit to indicate that the highest priority routine in service is finished.

Special consideration should be taken when deciding to use the Non-Specific EOI command. Here are two operating conditions in which it is best NOT used since the Fully Nested Mode structure will be destroyed:

— Using the Set Priority command within an interrupt service routine.

— Using a Special Mask Mode.

These conditions are covered in more detail in their own sections, but are listed here for reference.

SPECIFIC EOI COMMAND

Unlike a Non-Specific EOI command which automatically resets the highest priority ISR bit, a Specific EOI command specifies an exact ISR bit to be reset. Any one of the IRQ levels of an interrupt bank can be specified in the command.

The Specific EOI command is needed to reset the ISR bit of a completed service routine whenever the interrupt bank is not able to automatically determine it. The Specific EOI command can be used in all conditions of operation, including those that prohibit Non-Specific EOI command usage mentioned above.

AUTOMATIC EOI MODE

When programmed in the Automatic EOI Mode, the 80376 no longer needs to issue a command to notify the interrupt bank it has completed an interrupt routine. The interrupt bank accomplishes this by performing a Non-Specific EOI automatically at the end of the second INTA cycle.

Special consideration should be taken when deciding to use the Automatic EOI Mode because it may disturb the Fully Nested Mode structure. In the Automatic EOI Mode, the ISR bit of a routine in service is reset right after it is acknowledged, thus leaving no designation in the ISR that a service routine is being executed. If any interrupt request within the same bank occurs during this time and interrupts are enabled, it will get serviced regardless of its priority. Therefore, when using this mode, the 80376 should keep its interrupt request input disabled during execution of a service routine. By doing this, higher priority interrupt levels will be serviced only after the completion of a routine in service. This guideline restores the Fully Nested Mode structure. However, in this scheme, a routine in service cannot be interrupted since the host's interrupt request input is disabled.

4.4.2 INTERRUPT PRIORITIES

The 82370 PIC provides various methods for arranging the interrupt priorities of the interrupt request inputs to suit different applications. The following subsections explain these methods in detail.

4.4.2.1 Fully Nested Mode

The Fully Nested Mode of operation is a general purpose priority mode. This mode supports a multi-level interrupt structure in which all of the Interrupt Request (IRQ) inputs within one bank are arranged from highest to lowest.

Unless otherwise programmed, the Fully Nested Mode is entered by default upon initialization. At this time, IRQ0# is assigned the highest priority (priority=0) and IRQ7# the lowest (priority=7). This default priority can be changed, as will be explained later in the Rotating Priority Mode.

When an interrupt is acknowledged, the highest priority request is determined from the Interrupt Request Register (IRR) and its vector is placed on the bus. In addition, the corresponding bit in the In-Service Register (ISR) is set to designate the routine in service. This ISR bit will remain set until the 80376 issues an End Of Interrupt (EOI) command immediately before returning from the service routine; or alternately, if the Automatic End Of Interrupt (AEOI) bit is set, the ISR bit will be reset at the end of the second INTA cycle.

While the ISR bit is set, all further interrupts of the same or lower priority are inhibited. Higher level interrupts can still generate an interrupt, which will be acknowledged only if the 80376 internal interrupt enable flip-flop has been reenabled (through software inside the current service routine).

4.4.2.2 Automatic Rotation–Equal Priority Devices

Automatic rotation of priorities serves in applications where the interrupting devices are of equal priority within an interrupt bank. In this kind of environment, once a device is serviced, all other equal priority peripherals should be given a chance to be serviced before the original device is serviced again. This is accomplished by automatically assigning a device the lowest priority after being serviced. Thus, in the worst case, the device would have to wait until all other peripherals connected to the same bank are serviced before it is serviced again.

There are two methods of accomplishing automatic rotation. One is used in conjunction with the Non-Specific EOI command and the other is used with the Automatic EOI mode. These two methods are discussed below.

ROTATE ON NON-SPECIFIC EOI COMMAND

When the Rotate On Non-Specific EOI command is issued, the highest ISR bit is reset as in a normal Non-Specific EOI command. However, after it is reset, the corresponding Interrupt Request (IRQ) level is assigned the lowest priority. Other IRQ priorities rotate to conform to the Fully Nested Mode based on the newly assigned low priority.

Figure 4-4 shows how the Rotate On Non-Specific EOI command affects the interrupt priorities. Assume the IRQ priorities were assigned with IRQ0 the highest and IRQ7 the lowest. IRQ6 and IRQ4 are

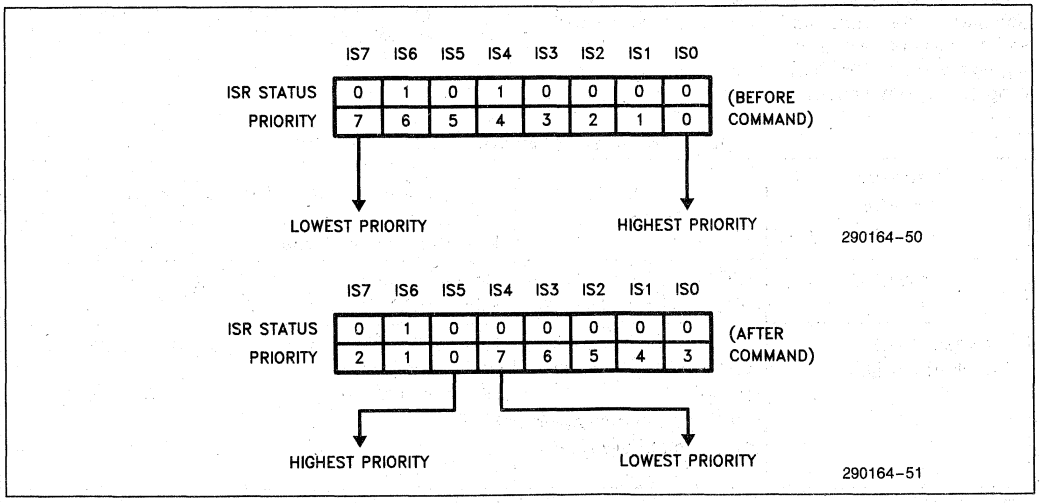


Figure 4-4. Rotate On Non-Specific EOI Command

already in service but neither is completed. Being the higher priority routine, IRQ4 is necessarily the routine being executed. During the IRQ4 routine, a rotate on Non-Specific EOI command is executed. When this happens, Bit 4 in the ISR is reset. IRQ4 then becomes the lowest priority and IRQ5 becomes the highest.

ROTATE ON AUTOMATIC EOI MODE

The Rotate On Automatic EOI Mode works much like the Rotate On Non-Specific EOI Command. The main difference is that priority rotation is done automatically after the second INTA cycle of an interrupt request. To enter or exit this mode, a Rotate-On-Automatic-EOI Set Command and Rotate-On-Automatic-EOI Clear Command is provided. After this mode is entered, no other commands are needed as in the normal Automatic EOI Mode. However, it must be noted again that when using any form of the Automatic EOI Mode, special consideration should be taken. The guideline presented in the Automatic EOI Mode also applies here.

4.4.2.3 Specific Rotation–Specific Priority

Specific rotation gives the user versatile capabilities in interrupt controlled operations. It serves in those applications in which a specific device's interrupt priority must be altered. As opposed to Automatic Rotation which will automatically set priorities after each interrupt request is serviced, specific rotation is completely user controlled. That is, the user selects which interrupt level is to receive the lowest or the highest priority. This can be done during the main program or within interrupt routines. Two specific rotation commands are available to the user: Set Priority Command and Rotate On Specific EOI Command.

SET PRIORITY COMMAND

The Set Priority Command allows the programmer to assign an IRQ level the lowest priority. All other interrupt levels will conform to the Fully Nested Mode based on the newly assigned low priority.

ROTATE ON SPECIFIC EOI COMMAND

The Rotate On Specific EOI Command is literally a combination of the Set Priority Command and the Specific EOI Command. Like the Set Priority Command, a specified IRQ level is assigned lowest priority. Like the Specific EOI Command, a specified level will be reset in the ISR. Thus, this command accomplishes both tasks in one single command.

4.4.2.4 Interrupt Priority Mode Summary

In order to simplify understanding the many modes of interrupt priority, Table 4-2 is provided to bring out their summary of operations.

4.4.3 INTERRUPT MASKING VIA INTERRUPT MASK REGISTER

Each bank in the 82370 PIC has an Interrupt Mask Register (IMR) which enhances interrupt control ca-

Table 4-2. Interrupt Priority Mode Summary

Interrupt Priority Mode	Operation Summary	Effect On Priority After EOI	
		Non-Specific/Automatic	Specific
Fully-Nested Mode	IRQ0# - Highest Priority IRQ7# - Lowest Priority	No change in priority. Highest ISR bit is reset.	Not Applicable.
Automatic Rotation (Equal Priority Devices)	Interrupt level just serviced is the lowest priority. Other priorities rotate to conform to Fully-Nested Mode.	Highest ISR bit is reset and the corresponding level becomes the lowest priority.	Not Applicable.
Specific Rotation (Specific Priority Devices)	User specifies the lowest priority level. Other priorities rotate to conform to Fully-Nested Mode.	Not Applicable.	As described under "Operation Summary".

pabilities. This IMR allows individual IRQ masking. When an IRQ is masked, its interrupt request is disabled until it is unmasked. Each bit in the 8-bit IMR disables one interrupt channel if it is set (HIGH). Bit 0 masks IRQ0, Bit 1 masks IRQ1 and so forth. Masking an IRQ channel will only disable the corresponding channel and does not affect the others' operations.

The IMR acts only on the output of the IRR. That is, if an interrupt occurs while its IMR bit is set, this request is not "forgotten". Even with an IRQ input masked, it is still possible to set the IRR. Therefore, when the IMR bit is reset, an interrupt request to the 80376 will then be generated, providing that the IRQ request remains active. If the IRQ request is removed before the IMR is reset, the Default Interrupt Vector (Bank A, level 7) will be generated during the interrupt acknowledge cycle.

SPECIAL MASK MODE

In the Fully Nested Mode, all IRQ levels of lower priority than the routine in service are inhibited. However, in some applications, it may be desirable to let a lower priority interrupt request to interrupt the routine in service. One method to achieve this is by using the Special Mask Mode. Working in conjunction with the IMR, the Special Mask Mode enables interrupts from all levels except the level in service. This is usually done inside an interrupt service routine by masking the level that is in service and then issuing the Special Mask Mode Command. Once the Special Mask Mode is enabled, it remains in effect until it is disabled.

4.4.4 EDGE OR LEVEL INTERRUPT TRIGGERING

Each bank in the 82370 PIC can be programmed independently for either edge or level sensing for the interrupt request signals. Recall that all IRQ inputs are active LOW. Therefore, in the edge triggered mode, an active edge is defined as an input transition from an inactive (HIGH) to active (LOW) state. The interrupt input may remain active without generating another interrupt. During level triggered mode, an interrupt request will be recognized by an active (LOW) input, and there is no need for edge detection. However, the interrupt request must be removed before the EOI Command is issued, or the 80376 must be disabled to prevent a second false interrupt from occurring.

In either modes, the interrupt request input must be active (LOW) during the first INTA cycle in order to be recognized. Otherwise, the Default Interrupt Vector will be generated at level 7 of Bank A.

4.4.5 INTERRUPT CASCADING

As mentioned previously, the 82370 allows for external Slave interrupt controllers to be cascaded to any of its external interrupt request pins. The 82370 PIC indicates that an external Slave Controller is to be serviced by putting the contents of the Vector Register associated with the particular request on the 80376 Data Bus during the first INTA cycle (instead of 00H during a non-slave service). The external logic should latch the vector on the Data Bus using the INTA status signals and use it to select the external Slave Controller to be serviced (see Figure 4-5). The selected Slave will then respond to the second INTA cycle and place its vector on the Data Bus. This method requires that if external Slave Controllers are used in the system, no vector should be programmed to 00H.

Since the external Slave Cascade Address is provided on the Data Bus during INTA cycle 1, an external latch is required to capture this address for the Slave Controller. A simple scheme is depicted in Figure 4-5 below.

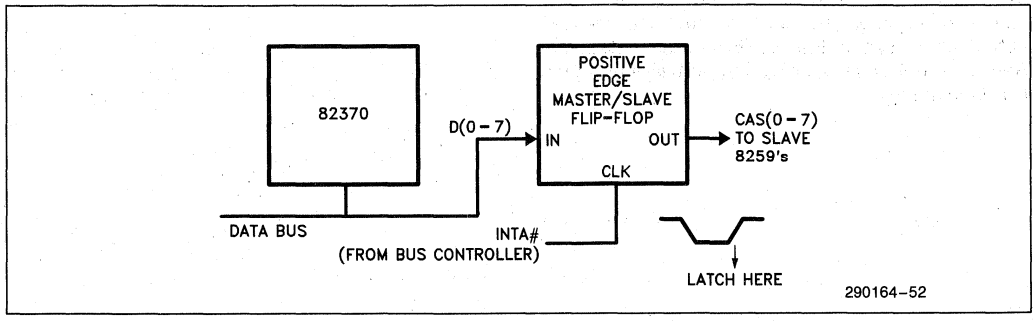


Figure 4-5. Slave Cascade Address Capturing

4.4.5.1 Special Fully Nested Mode

This mode will be used where cascading is employed and the priority is to be conserved within each Slave Controller. The Special Fully Nested Mode is similar to the "regular" Fully Nested Mode with the following exceptions:

— When an interrupt request from a Slave Controller is in service, this Slave Controller is not locked out from the Master's priority logic. Further interrupt requests from the higher priority logic within the Slave Controller will be recognized by the 82370 PIC and will initiate interrupts to the 80376. In comparing to the "regular" Fully Nested Mode, the Slave Controller is masked out when its request is in service and no higher requests from the same Slave Controller can be serviced.

— Before exiting the interrupt service routine, the software has to check whether the interrupt serviced was the only request from the Slave Controller. This is done by sending a Non-Specific EOI Command to the Slave Controller and then reading its In Service Register. If there are no requests in the Slave Controller, a Non-Specific EOI can be sent to the corresponding 82370 PIC bank also. Otherwise, no EOI should be sent.

4.4.6 READING INTERRUPT STATUS

The 82370 PIC provides several ways to read different status of each interrupt bank for more flexible interrupt control operations. These include polling the highest priority pending interrupt request and reading the contents of different interrupt status registers.

4.4.6.1 Poll Command

The 82370 PIC supports status polling operations with the Poll Command. In a Poll Command, the pending interrupt request with the highest priority can be determined. To use this command, the INT output is not used, or the 80376 interrupt is disabled. Service to devices is achieved by software using the Poll Command.

This mode is useful if there is a routine command common to several levels so that the INTA sequence is not needed. Another application is to use the Poll Command to expand the number of priority levels.

Notice that the ICW2 mechanism is not supported for the Poll Command. However, if the Poll Command is used, the programmable Vector Registers are of no concern since no INTA cycle will be generated.

4.4.6.2 Reading Interrupt Registers

The contents of each interrupt register (IRR, ISR, and IMR) can be read to update the user's program on the present status of the 82370 PIC. This can be a versatile tool in the decision making process of a service routine, giving the user more control over interrupt operations.

The reading of the IRR and ISR contents can be performed via the Operation Control Word 3 by using a Read Status Register Command and the content of IMR can be read via a simple read operation of the register itself.

4.5 Register Set Overview

Each bank of the 82370 PIC consists of a set of 8-bit registers to control its operations. The address map of all the registers is shown in Table 4-3 below. Since all three register sets are identical in functions, only one set will be described.

Functionally, each register set can be divided into five groups. They are: the four Initialization Command Words (ICW's), the three Operation Control Words (OCW's), the Poll/Interrupt Request/In-Service Register, the Interrupt Mask Register, and the Vector Registers. A description of each group follows.

Table 4-3. Interrupt Controller Register Address Map

Port Address	Access	Register Description
20H	Write	Bank B ICW1, OCW2, or OCW3
	Read	Bank B Poll, Request or In-Service Status Register
21H	Write	Bank B ICW2, ICW3, ICW4, OCW1
	Read	Bank B Mask Register
22H	Read	Bank B ICW2
28H	Read/Write	IRQ8 Vector Register
29H	Read/Write	IRQ9 Vector Register
2AH	Read/Write	Reserved
2BH	Read/Write	IRQ11 Vector Register
2CH	Read/Write	IRQ12 Vector Register
2DH	Read/Write	IRQ13 Vector Register
2EH	Read/Write	IRQ14 Vector Register
2FH	Read/Write	IRQ15 Vector Register
A0H	Write	Bank C ICW1, OCW2, or OCW3
	Read	Bank C Poll, Request or In-Service Status Register
A1H	Write	Bank C ICW2, ICW3, ICW4, OCW1
	Read	Bank C Mask Register
A2H	Read	Bank C ICW2
A8H	Read/Write	IRQ16 Vector Register
A9H	Read/Write	IRQ17 Vector Register
AAH	Read/Write	IRQ18 Vector Register
ABH	Read/Write	IRQ19 Vector Register
ACH	Read/Write	IRQ20 Vector Register
ADH	Read/Write	IRQ21 Vector Register
AEH	Read/Write	IRQ22 Vector Register
AFH	Read/Write	IRQ23 Vector Register
30H	Write	Bank A ICW1, OCW2, or OCW3
	Read	Bank A Poll, Request or In-Service Status Register
31H	Write	Bank A ICW2, ICW3, ICW4, OCW1
	Read	Bank A Mask Register
32H	Read	Bank ICW2
38H	Read/Write	IRQ0 Vector Register
39H	Read/Write	IRQ1 Vector Register
3AH	Read/Write	IRQ1.5 Vector Register
3BH	Read/Write	IRQ3 Vector Register
3CH	Read/Write	IRQ4 Vector Register
3DH	Read/Write	Reserved
3EH	Read/Write	Reserved
3FH	Read/Write	IRQ7 Vector Register

4.5.1 INITIALIZATION COMMAND WORDS (ICW)

Before normal operation can begin, the 82370 PIC must be brought to a known state. There are four 8-bit Initialization Command Words in each interrupt bank to setup the necessary conditions and modes for proper operation. Except for the second command word (ICW2) which is a read/write register, the other three are write-only registers. Without going into detail of the bit definitions of the command words, the following subsections give a brief description of what functions each command word controls.

ICW1

The ICW1 has three major functions. They are:

— To select between the two IRQ input triggering modes (edge- or level-triggered);

— To designate whether or not the interrupt bank is to be used alone or in the cascade mode. If the cascade mode is desired, the interrupt bank will accept ICW3 for further cascade mode programming. Otherwise, no ICW3 will be accepted;

— To determine whether or not ICW4 will be issued; that is, if any of the ICW4 operations are to be used.

ICW2

ICW2 is provided for compatibility with the 82C59A only. Its contents do not affect the operation of the interrupt bank in any way. Whenever the ICW2 of any of the three banks is written into, an interrupt is generated from bank A at level 1.5. The interrupt request will be cleared after the ICW2 register has been read by the 80376. The user is expected to program the corresponding vector register or to use it as an indicator that an attempt was made to alter the contents. Note that each ICW2 register has different addresses for read and write operations.

ICW3

The interrupt bank will only accept an ICW3 if programmed in the external cascade mode (as indicated in ICW1). ICW3 is used for specific programming within the cascade mode. The bits in ICW3 indicate which interrupt request inputs have a Slave cascaded to them. This will subsequently affect the interrupt vector generation during the interrupt acknowledge cycles as described previously.

ICW4

The ICW4 is accepted only if it was selected in ICW1. This command word register serves two functions:

— To select either the Automatic EOI mode or software EOI mode;

— To select if the Special Nested mode is to be used in conjunction with the cascade mode.

4.5.2 OPERATION CONTROL WORDS (OCW)

Once initialized by the ICW's, the interrupt banks will be operating in the Fully Nested Mode by default and they are ready to accept interrupt requests. However, the operations of each interrupt bank can be further controlled or modified by the use of OCW's. Three OCW's are available for programming various modes and commands. Note that all OCW's are 8-bit write-only registers.

The modes and operations controlled by the OCW's are:

— Fully Nested Mode;

— Rotating Priority Mode;

— Special Mask Mode;

— Poll Mode;

— EOI Commands;

— Read Status Commands.

OCW1

OCW1 is used solely for masking operations. It provides a direct link to the Internal Mask Register (IMR). The 80376 can write to this OCW register to enable or disable the interrupt inputs. Reading the pre-programmed mask can be done via the Interrupt Mask Register which will be discussed shortly.

OCW2

OCW2 is used to select End-Of-Interrupt, Automatic Priority Rotation, and Specific Priority Rotation operations. Associated commands and modes of these operations are selected using the different combinations of bits in OCW2.

Specifically, the OCW2 is used to:

— Designate an interrupt level (0–7) to be used to reset a specific ISR bit or to set a specific priority. This function can be enabled or disabled;

— Select which software EOI command (if any) is to be executed (i.e. Non-Specific or Specific EOI);

— Enable one of the priority rotation operations (i.e. Rotate On Non-Specific EOI, Rotate On Automatic EOI, or Rotate On Specific EOI).

OCW3

There are three main categories of operation that OCW3 controls. They are summarized as follows:

— To select and execute the Read Status Register Commands, either reading the Interrupt Request Register (IRR) or the In-Service Register (ISR);

— To issue the Poll Command. The Poll Command will override a Read Register Command if both functions are enabled simultaneously;

— To set or reset the Special Mask Mode.

4.5.3 POLL/INTERRUPT REQUEST/IN-SERVICE STATUS REGISTER

As the name implies, this 8-bit read-only register has multiple functions. Depending on the command issued in the OCW3, the content of this register reflects the result of the command executed. For a Poll Command, the register read contains the binary code of the highest priority level requesting service (if any). For a Read IRR Command, the register content will show the current pending interrupt request(s). Finally, for a Read ISR Command, this register will specify all interrupt levels which are being serviced.

4.5.4 INTERRUPT MASK REGISTER (IMR)

This is a read-only 8-bit register which, when read, will specify all interrupt levels within the same bank that are masked.

4.5.5 VECTOR REGISTERS (VR)

Each interrupt request input has an 8-bit read/write programmable vector register associated with it. The registers should be programmed to contain the interrupt vector for the corresponding request. The contents of the Vector Register will be placed on the Data Bus during the INTA cycles as described previously.

4.6 Programming

Programming the 82370 PIC is accomplished by using two types of command words: ICW's and OCW's. All modes and commands explained in the previous sections are programmable using the ICW's and OCW's. The ICW's are issued from the 80376 in a sequential format and are used to setup the banks in the 82370 PIC in an initial state of operation. The OCW's are issued as needed to vary and control the 82370 PIC's operations.

Both ICW's and OCW's are sent by the 80376 to the interrupt banks via the Data Bus. Each bank distinguishes between the different ICW's and OCW's by the I/O address map, the sequence they are issued (ICW's only), and by some dedicated bits among the ICW's and OCW's.

An example of programming the 82370 interrupt controllers is given in Appendix C (Programming the 82370 Interrupt Controllers).

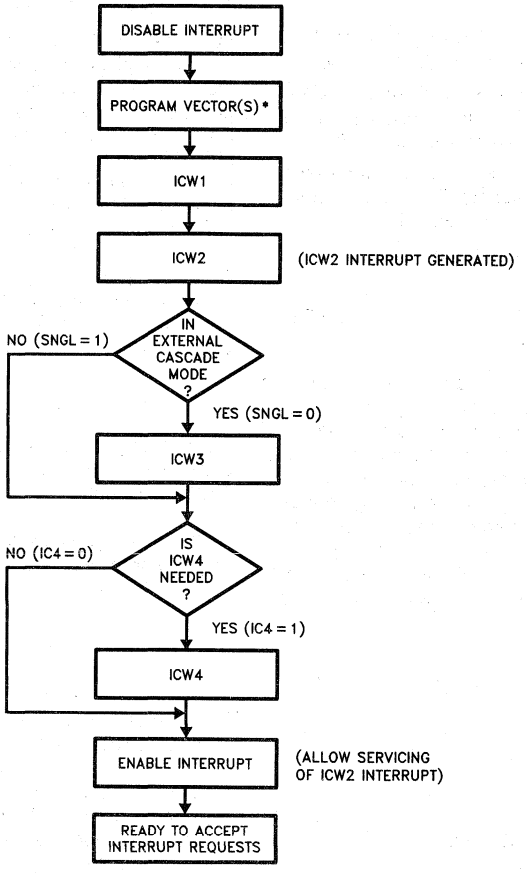
All three interrupt banks are programmed in a similar way. Therefore, only a single bank will be described in the following sections.

4.6.1 INITIALIZATION (ICW)

Before normal operation can begin, each bank must be initialized by programming a sequence of two to four bytes written into the ICW's.

Figure 4-6 shows the initialization flow for an interrupt bank. Both ICW1 and ICW2 must be issued for any form of operation. However, ICW3 and ICW4 are used only if designated in ICW1. Once initialized, if any programming changes within the ICW's are to be made, the entire ICW sequence must be reprogrammed, not just an individual ICW.

Note that although the ICW2's in the 82370 PIC do not effect the Bank's operation, they still must be programmed in order to preserve the compatibility with the 82C59A. The contents programmed are not relevant to the overall operations of the interrupt banks. Also, whenever one of the three ICW2's is programmed, an interrupt level 1.5 in Bank A will be generated. This interrupt request will be cleared upon reading of the ICW2 registers. Since the three ICW2's share the same interrupt level and the system may not know the origin of the interrupt, all three ICW2's must be read.



*ICW2 vector address must be programmed now.
Other vector addresses may be programmed via ICW2 interrupt service routine.

Figure 4-6. Initialization Sequence

Certain internal setup conditions occur automatically within the interrupt bank after the first ICW (ICW1) has been issued. These are:

— The edge sensitive circuit is reset, which means that following initialization, an interrupt request input must make a HIGH-to-LOW transition to generate an interrupt;

— The Interrupt Mask Register (IMR) is cleared; that is, all interrupt inputs are enabled;

— IRQ7 input of each bank is assigned priority 7 (lowest);

— Special Mask Mode is cleared and Status Read is set to IRR;

— If no ICW4 is needed, then no Automatic-EOI is selected.

4.6.2 VECTOR REGISTERS (VR)

Each interrupt request input has a separate Vector Register. These Vector Registers are used to store the pre-programmed vector number corresponding to their interrupt sources. In order to guarantee proper interrupt handling, all Vector Registers must be programmed with the predefined vector numbers. Since an interrupt request will be generated whenever an ICW2 is written during the initialization sequence, it is important that the Vector Register of IRQ1.5 in Bank A should be initialized and the interrupt service routine of this vector is set up before the ICW's are written.

4.6.3 OPERATION CONTROL WORDS (OCW)

After the ICW's are programmed, the operations of each interrupt controller bank can be changed by writing into the OCW's as explained before. There is no special programming sequence required for the OCW's. Any OCW may be written at any time in order to change the mode of or to perform certain operations on the interrupt banks.

4.6.3.1 Read Status and Poll Commands (OCW3)

Since the reading of IRR and ISR status as well as the result of a Poll Command are available on the same read-only Status Register, a special Read Status/Poll Command must be issued before the Poll/Interrupt Request/In-Service Status Register is read. This command can be specified by writing the required control word into OCW3. As mentioned earlier, if both the Poll Command and the Status Read Command are enabled simultaneously, the Poll Command will override the Status Read. That is, after the command execution, the Status Register will contain the result of the Poll Command.

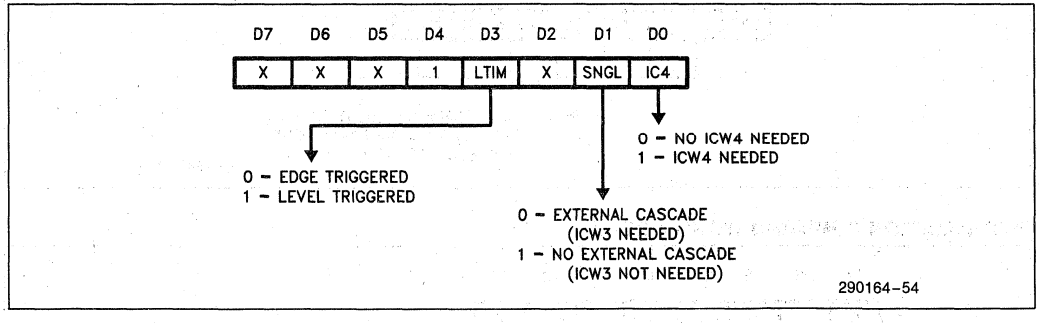
Note that for reading IRR and ISR, there is no need to issue a Read Status Command to the OCW3 every time the IRR or ISR is to be read. Once a Read Status Command is received by the interrupt bank, it "remembers" which register is selected. However, this is not true when the Poll Command is used.

In the Poll Command, after the OCW3 is written, the 82370 PIC treats the next read to the Status Register as an interrupt acknowledge. This will set the appropriate IS bit if there is a request and read the priority level. Interrupt Request input status remains unchanged from the Poll Command to the Status Read.

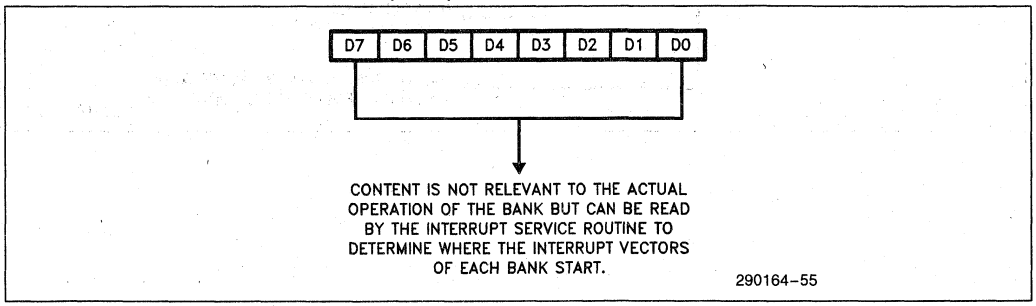
In addition to the above read commands, the Interrupt Mask Register (IMR) can also be read. When read, this register reflects the contents of the pre-programmed OCW1 which contains information on which interrupt request(s) is(are) currently disabled.

4.7 Register Bit Definition

INITIALIZATION COMMAND WORD 1 (ICW1)

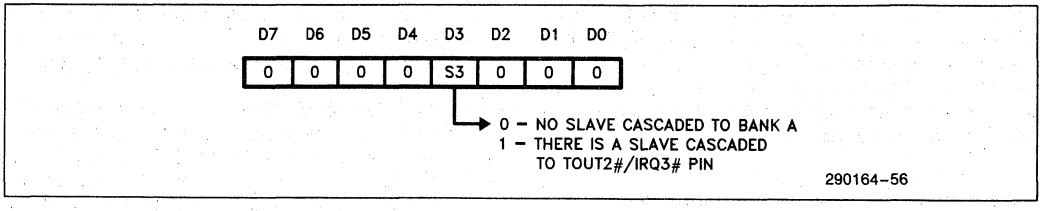


INITIALIZATION COMMAND WORD 2 (ICW2)

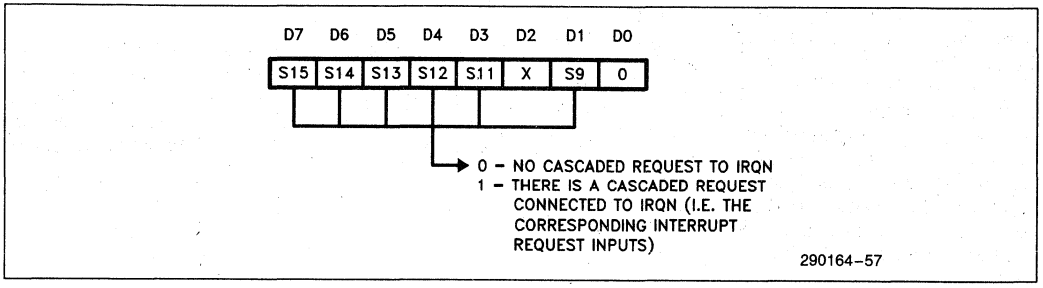


INITIALIZATION COMMAND WORD 3 (ICW3)

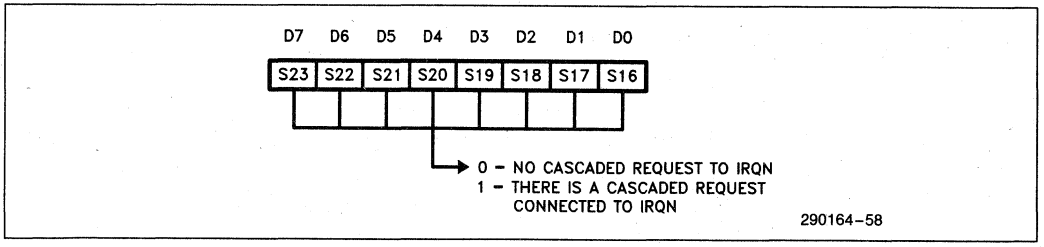
ICW3 for Bank A:



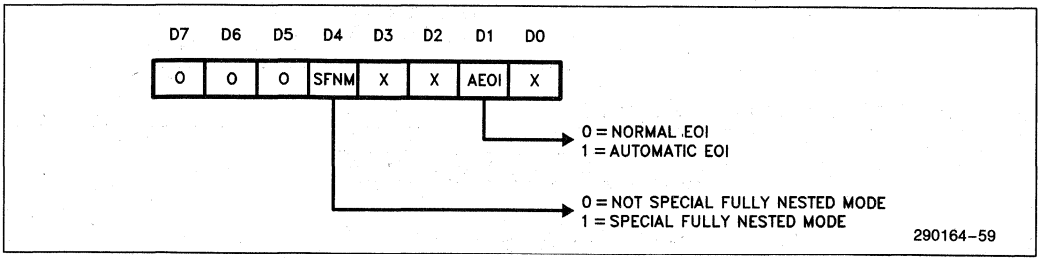
ICW3 for Bank B:



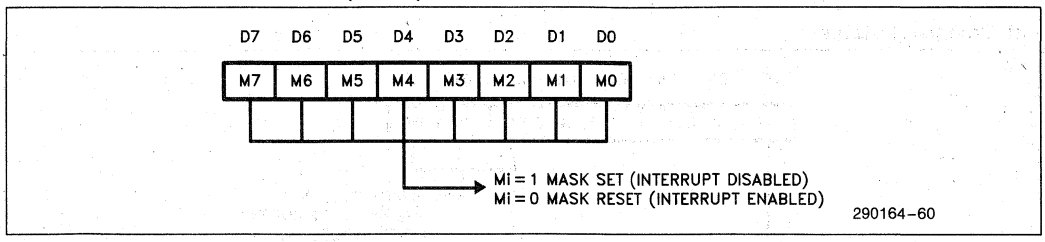
ICW3 for Bank C:



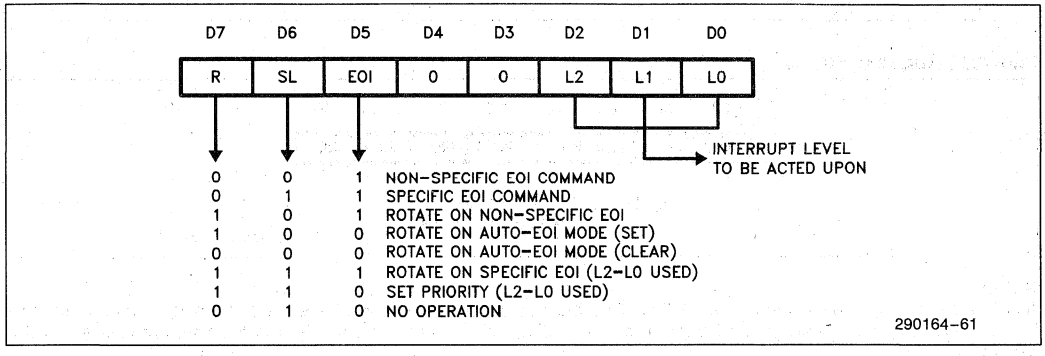
INITIALIZATION COMMAND WORD 4 (ICW4)



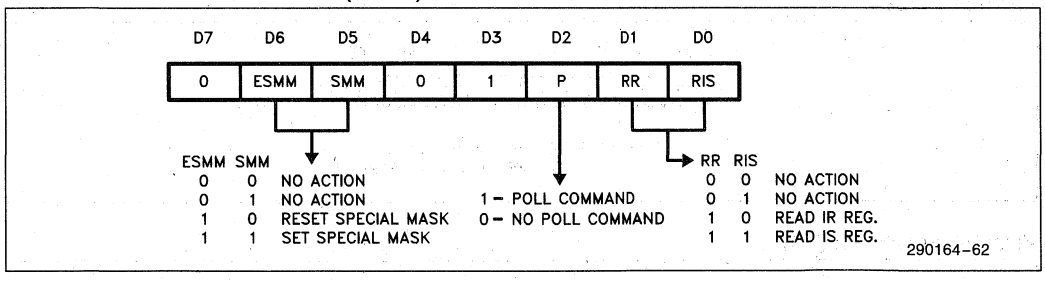
OPERATION CONTROL WORD 1 (OCW1)



OPERATION CONTROL WORD 2 (OCW2)



OPERATION CONTROL WORD 3 (OCW3)

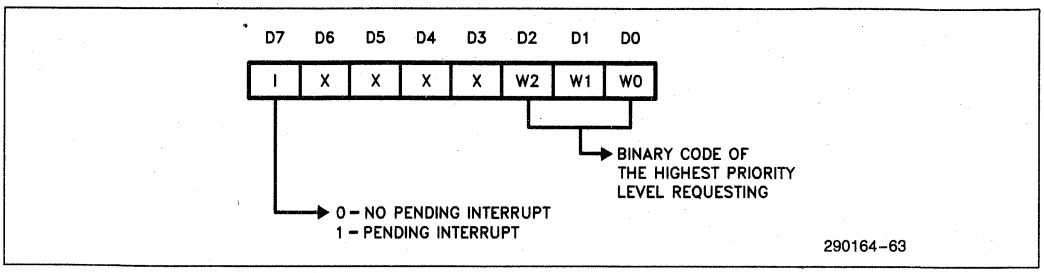


ESMM — Enable Special Mask Mode. When this bit is set to 1, it enables the SMM bit to set or reset the Special Mask Mode. When this bit is set to 0, SMM bit becomes don't care.

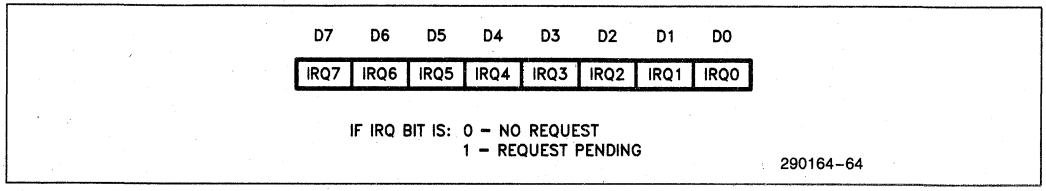
SMM — Special Mask Mode. If ESMM = 1 and SMM = 1, the interrupt controller bank will enter Special Mask Mode. If ESMM = 1 and SMM = 0, the bank will revert to normal mask mode. When ESMM = 0, SMM has no effect.

POLL/INTERRUPT REQUEST/IN-SERVICE STATUS REGISTER

Poll Command Status

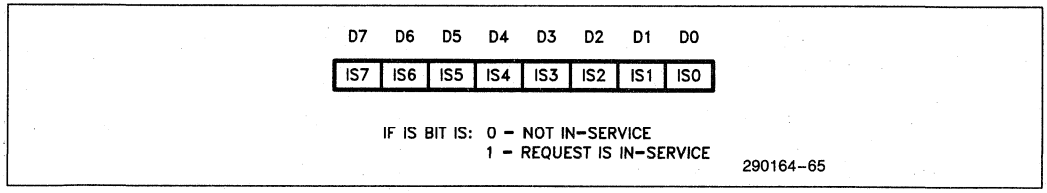


Interrupt Request Status



NOTE:
Although all Interrupt Request inputs are active LOW, the internal logical will invert the state of the pins so that when there is a pending interrupt request at the input, the corresponding IRQ bit will be set to HIGH in the Interrupt Request Status register.

In-Service Status



VECTOR REGISTER (VR)

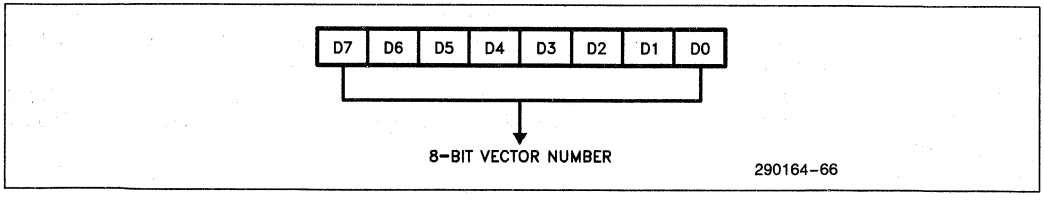


Table 4-4. Register Operational Summary

Operational Description	Command Words	Bits
Fully Nested Mode	OCW-Default	
Non-specific EOI Command	OCW2	EOI
Specific EOI Command	OCW2	SL, EOI, L0–L2
Automatic EOI Mode	ICW1, ICW4	IC4, AEOI
Rotate On Non-Specific EOI Command	OCW2	EOI
Rotate On Automatic EOI Mode	OCW2	R, SL, EOI
Set Priority Command	OCW2	L0–L2
Rotate On Specific EOI Command	OCW2	R, SL, EOI
Interrupt Mask Register	OCW1	M0–M7
Special Mask Mode	OCW3	ESMM, SMM
Level Triggered Mode	ICW1	LTIM
Edge Triggered Mode	ICW1	LTIM
Read Register Command, IRR	OCW3	RR, RIS
Read Register Command, ISR	OCW3	RR, RIS
Read IMR	IMR	M0–M7
Poll Command	OCW3	P
Special Fully Nested Mode	ICW1, ICW4	IC4, SFNM

4.8 Register Operational Summary

For ease of reference, Table 4-4 gives a summary of the different operating modes and commands with their corresponding registers.

5.0 PROGRAMMABLE INTERVAL TIMER

5.1 Functional Description

The 82370 contains four independently Programmable Interval Timers: Timer 0–3. All four timers are functionally compatible to the Intel 82C54. The first three timers (Timer 0–2) have specific functions. The fourth timer, Timer 3, is a general purpose timer. Table 5-1 depicts the functions of each timer. A brief description of each timer's function follows.

Table 5-1. Programmable Interval Timer Functions

Timer	Output	Function
0	IRQ8	Event Based IRQ8 Generator
1	TOUT1/REF#	Gen. Purpose/DRAM Refresh Req.
2	TOUT2/IRQ3#	Gen. Purpose/Speaker Out/IRQ3#
3	TOUT3#	Gen. Purpose/IRQ0 Generator

TIMER 0—Event Based Interrupt Request 8 Generator

Timer 0 is intended to be used as an Event Counter. The output of this timer will generate an Interrupt Request 8 (IRQ8) upon a rising edge of the timer output (TOUT0). Normally, this timer is used to implement a time-of-day clock or system tick. The Timer 0 output is not available as an external signal.

TIMER 1—General Purpose/DRAM Refresh Request

The output of Timer 1, TOUT1, can be used as a general purpose timer or as a DRAM Refresh Request signal. The rising edge of this output creates a DRAM refresh request to the 82370 DRAM Refresh Controller. Upon reset, the Refresh Request function is disabled, and the output pin is the Timer 1 output.

TIMER 2—General Purpose/Speaker Out/IRQ3#

The Timer 2 output, TOUT2#, could be used to support tone generation to an external speaker. This pin is a bidirectional signal. When used as an input, a logic LOW asserted at this pin will generate an Interrupt Request 3 (IRQ3#) (see Programmable Interrupt Controller).

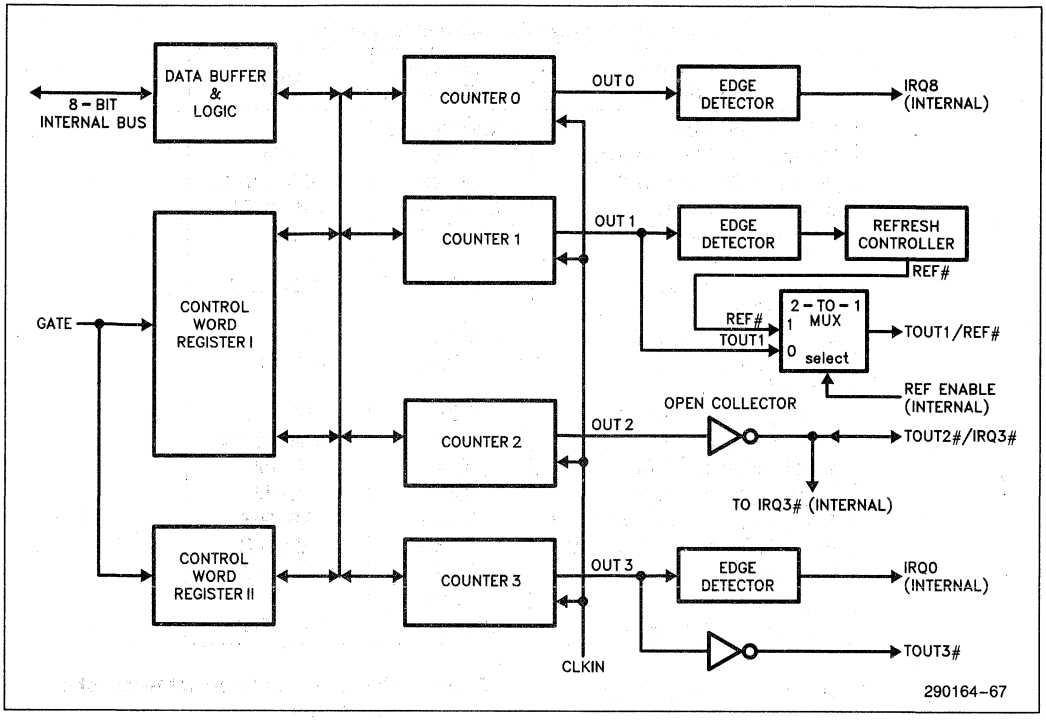


Figure 5-1. Block Diagram of Programmable Interval Timer

TIMER 3—General Purpose/Interrupt Request 0 Generator

The output of Timer 3 is fed to an edge detector and generates an Interrupt Request 0 (IRQ0) in the 82370. The inverted output of this timer (TOUT3#) is also available as an external signal for general purpose use.

5.1.1 INTERNAL ARCHITECTURE

The functional block diagram of the Programmable Interval Timer section is shown in Figure 5-1. Following is a description of each block.

DATA BUFFER & READ/WRITE LOGIC

This part of the Programmable Interval Timer is used to interface the four timers to the 82370 internal bus. The Data Buffer is for transferring commands and data between the 8-bit internal bus and the timers.

The Read/Write Logic accepts inputs from the internal bus and generates signals to control other functional blocks within the timer section.

CONTROL WORD REGISTERS I & II

The Control Word Registers are write-only registers. They are used to control the operating modes of the timers. Control Word Register I controls Timers 0, 1 and 2, and Control Word Register II controls Timer 3. Detailed description of the Control Word Registers will be included in the Register Set Overview section.

COUNTER 0, COUNTER 1, COUNTER 2, COUNTER 3

Counters 0, 1, 2, and 3 are the major parts of Timers 0, 1, 2, and 3, respectively. These four functional blocks are identical in operation, so only a single counter will be described. The internal block diagram of one counter is shown in Figure 5-2.

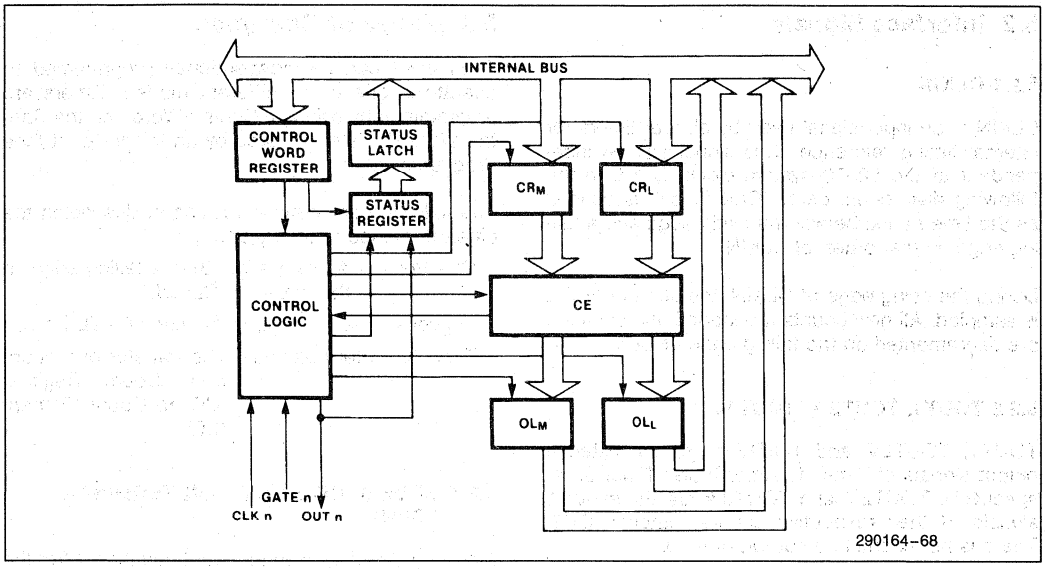


Figure 5-2. Internal Block Diagram of a Counter

The four counters share a common clock input (CLKIN), but otherwise are fully independent. Each counter is programmable to operate in a different mode.

Although the Control Word Register is shown in the figure, it is not part of the counter itself. Its programmed contents are used to control the operations of the counters.

The Status Register, when latched, contains the current contents of the Control Word Register and status of the output and Null Count Flag (see Read Back Command).

The Counting Element (CE) is the actual counter. It is a 16-bit presettable synchronous down counter.

The Output Latches (OL) contain two 8-bit latches (OLM and OLL). Normally, these latches "follow" the content of the CE. OLM contains the most significant byte of the counter and OLL contains the least significant byte. If the Counter Latch Command is sent to the counter, OL will latch the present count until read by the 80376 and then return to follow the CE. One latch at a time is enabled by the timer's Control Logic to drive the internal bus. This is how the 16-bit Counter communicates over the 8-bit internal bus. Note that CE cannot be read. Whenever the count is read, it is one of the OL's that is being read.

When a new count is written into the counter, the value will be stored in the Count Registers (CR), and transferred to CE. The transferring of the contents from CR's to CE is defined as "loading" of the counter. The Count Register contains two 8-bit registers: CRM (which contains the most significant byte) and CRL (which contains the least significant byte). Similar to the OL's, the Control Logic allows one register at a time to be loaded from the 8-bit internal bus. However, both bytes are transferred from the CR's to the CE simultaneously. Both CR's are cleared when the Counter is programmed. This way, if the Counter has been programmed for one byte count (either the most significant or the least significant byte only), the other byte will be zero. Note that CE cannot be written into directly. Whenever a count is written, it is the CR that is being written.

As shown in the diagram, the Control Logic consists of three signals: CLKIN, GATE, and OUT. CLKIN and GATE will be discussed in detail in the section that follows. OUT is the internal output of the counter. The external outputs of some timers (TOUT) are the inverted version of OUT (see TOUT1, TOUT2#, TOUT3#). The state of OUT depends on the mode of operation of the timer.

5

5.2 Interface Signals

5.2.1 CLKIN

CLKIN is an input signal used by all four timers for internal timing reference. This signal can be independent of the 82370 system clock, CLK2. In the following discussion, each "CLK Pulse" is defined as the time period between a rising edge and a falling edge, in that order, of CLKIN.

During the rising edge of CLKIN, the state of GATE is sampled. All new counts are loaded and counters are decremented on the falling edge of CLKIN.

5.2.2 TOUT1, TOUT2#, TOUT3#

TOUT1, TOUT2# and TOUT3# are the external output signals of Timer 1, Timer 2 and Timer 3, respectively. TOUT2# and TOUT3# are the inverted signals of their respective counter outputs, OUT. There is no external output for Timer 0.

If Timer 2 is to be used as a tone generator of a speaker, external buffering must be used to provide sufficient drive capability.

The Outputs of Timer 2 and 3 are dual function pins. The output pin of Timer 2 (TOUT2#/IRQ3#), which is a bidirectional open-collector signal, can also be used as interrupt request input. When the interrupt function is enabled (through the Programmable Interrupt Controller), a LOW on this input will generate an Interrupt Request 3# to the 82370 Programmable Interrupt Controller. This pin has a weak internal pull-up resistor. To use the IRQ3# function, Timer 2 should be programmed so that OUT2 is LOW. Additionally, OUT3 of Timer 3 is connected to an edge detector which will generate an Interrupt Request 0 (IRQ0) to the 82370 after the rising edge of OUT3 (see Figure 5-1).

5.2.3 GATE

GATE is not an externally controllable signal. Rather, it can be software controlled with the Internal Control Port. The state of GATE is always sampled on the rising edge of CLKIN. Depending on the mode of operation, GATE is used to enable/disable counting or trigger the start of an operation.

For Timer 0 and 1, GATE is always enabled (HIGH). For Timer 2 and 3, GATE is connected to Bit 0 and 6, respectively, of an Internal Control Port (at address 61H) of the 82370. After a hardware reset, the state of GATE of Timer 2 and 3 is disabled (LOW).

5.3 Modes of Operation

Each timer can be independently programmed to operate in one of six different modes. Timers are programmed by writing a Control Word into the Control Word Register followed by an Initial Count (see Programming).

The following are defined for use in describing the different modes of operation.

CLK Pulse— A rising edge, then a falling edge, in that order, of CLKIN.

Trigger— A rising edge of a timer's GATE input.

Timer/Counter Loading— The transfer of a count from Count Register (CR) to Count Element (CE).

5.3.1 MODE 0–INTERRUPT ON TERMINAL COUNT

Mode 0 is typically used for event counting. After the Control Word is written, OUT is initially LOW, and will remain LOW until the counter reaches zero. OUT then goes HIGH and remains HIGH until a new count or a new Mode 0 Control Word is written into the counter.

In this mode, GATE=HIGH enables counting; GATE = LOW disables counting. However, GATE has no effect on OUT.

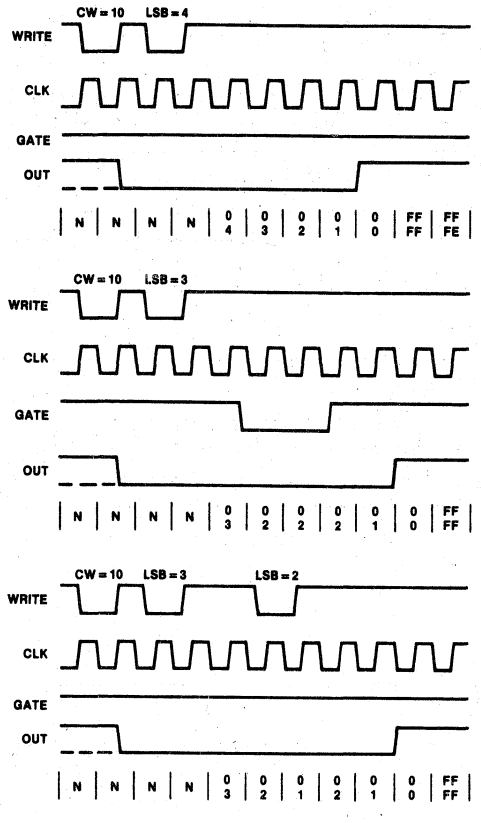
After the Control Word and initial count are written to a timer, the initial count will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not go HIGH until N+1 CLK pulses after the initial count is written.

If a new count is written to the timer, it will be loaded on the next CLK pulse and counting will continue from the new count. If a two-byte count is written, the following happens:

1. Writing the first byte disables counting, OUT is set LOW immediately (i.e. no CLK pulse required).

2. Writing the second byte allows the new count to be loaded on the next CLK pulse.

This allows the counting sequence to be synchronized by software. Again, OUT does not go HIGH until N+1 CLK pulses after the new count of N is written.



NOTES:
The following conventions apply to all mode timing diagrams.
1. Counters are programmed for binary (not BCD) counting and for reading/writing least significant byte (LSB) only.
2. The counter is always selected (CS# always low).
3. CW stands for "Control Word"; CW = 10 means a control word of 10, Hex is written to the counter.
4. LSB stands for "Least significant byte" of count.
5. Numbers below diagrams are count values.
The lower number is the least significant byte.
The upper number is the most significant byte. Since the counter is programmed to read/write LSB only, the most significant byte cannot be read.
N stands for an undefined count.
Vertical lines show transitions between count values.

Figure 5-3. Mode 0

If an initial count is written while GATE is LOW, the counter will be loaded on the next CLK pulse. When GATE goes HIGH, OUT will go HIGH N CLK pulses later; no CLK pulse is needed to load the counter as this has already been done.

5.3.2 MODE 1–GATE RETRIGGERABLE ONE-SHOT

In this mode, OUT will be initially HIGH. OUT will go LOW on the CLK pulse following a trigger to start the one-shot operation. The OUT signal will then remain LOW until the timer reaches zero. At this point, OUT will stay HIGH until the next trigger comes in. Since the state of GATE signals of Timer 0 and 1 are internally set to HIGH.

After writing the Control Word and initial count, the timer is considered "armed". A trigger results in loading the timer and setting OUT LOW on the next CLK pulse. Therefore, an initial count of N will result in a one-shot pulse width of N CLK cycles. Note

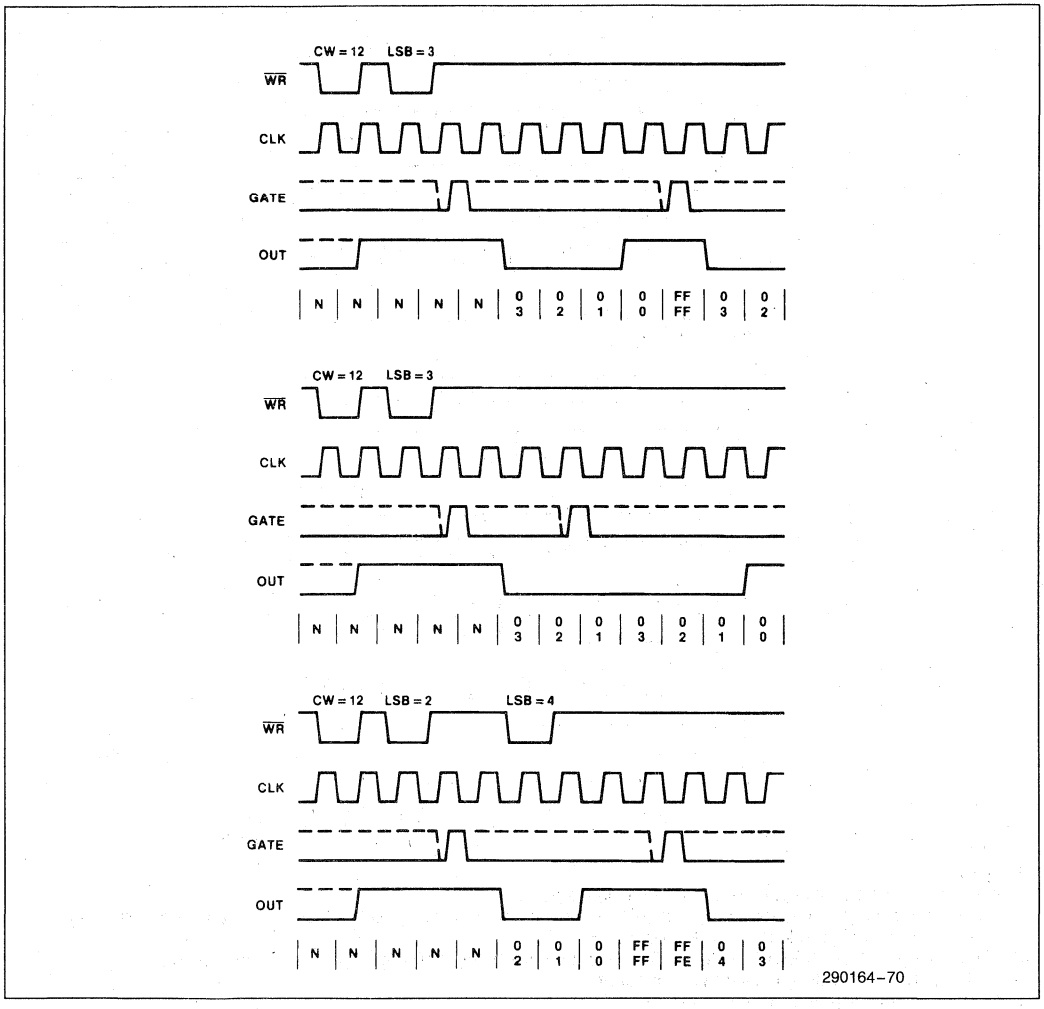


Figure 5-4. Mode 1

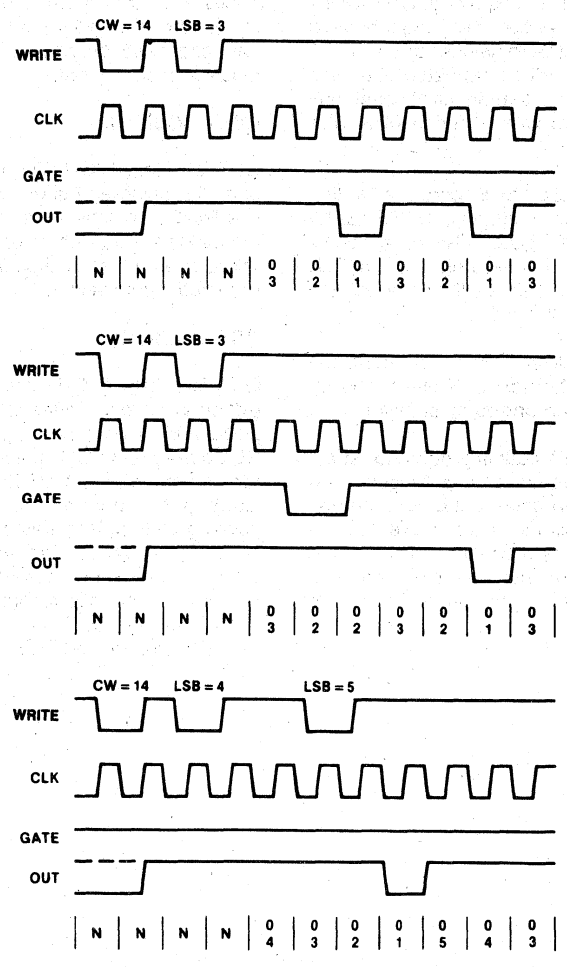
that this one-shot operation is retriggerable; i.e. OUT will remain LOW for N CLK pulses after every trigger. The one-shot operation can be repeated without rewriting the same count into the timer.

If a new count is written to the timer during a one-shot operation, the current one-shot pulse width will not be affected until the timer is retriggered. This is because loading of the new count to CE will occur only when the one-shot is triggered.

5.3.3 MODE 2–RATE GENERATOR

This mode is a divide-by-N counter. It is typically used to generate a Real Time Clock interrupt. OUT will initially be HIGH. When the initial count has decremented to 1, OUT goes LOW for one CLK pulse, then OUT goes HIGH again. Then the timer reloads the initial count and the process is repeated. In other words, this mode is periodic since the same sequence is repeated itself indefinitely. For an initial count of N, the sequence repeats every N CLK cycles.

Similar to Mode 0, GATE = HIGH enables counting, where GATE = LOW disables counting. If GATE goes LOW during an output pulse (LOW), OUT is set HIGH immediately. A trigger (rising edge on GATE) will reload the timer with the initial count on the next CLK pulse. Then, OUT will go LOW (for one CLK pulse) N CLK pulses after the new trigger. Thus, GATE can be used to synchronize the timer.



NOTE:
A GATE transition should not occur one clock prior to terminal count.

Figure 5-5. Mode 2

After writing a Control Word and initial count, the timer will be loaded on the next CLK pulse. OUT goes LOW (for one CLK pulse) N CLK pulses after the initial count is written. This is another way the timer may be synchronized by software.

Writing a new count while counting does not affect the current counting sequence because the new count will not be loaded until the end of the current counting cycle. If a trigger is received after writing a new count but before the end of the current period, the timer will be loaded with the new count on the next CLK pulse after the trigger, and counting will continue with the new count.

5.3.4 MODE 3–SQUARE WAVE GENERATOR

Mode 3 is typically used for Baud Rate generation. Functionally, this mode is similar to Mode 2 except

for the duty cycle of OUT. In this mode, OUT will be initially HIGH. When half of the initial count has expired, OUT goes low for the remainder of the count. The counting sequence will be repeated, thus this mode is also periodic. Note that an initial count of N results in a square wave with a period of N CLK pulses.

The GATE input can be used to synchronize the timer. GATE = HIGH enables counting; GATE = LOW disables counting. If GATE goes LOW while OUT is LOW, OUT is set HIGH immediately (i.e. no CLK pulse is required). A trigger reloads the timer with the initial count on the next CLK pulse.

After writing a Control Word and initial count, the timer will be loaded on the next CLK pulse. This allows the timer to be synchronized by software.

Writing a new count while counting does not affect the current counting sequence. If a trigger is received after writing a new count but before the end of the current half-cycle of the square wave, the timer will be loaded with the new count on the next CLK pulse and counting will continue from the new count. Otherwise, the new count will be loaded at the end of the current half-cycle.

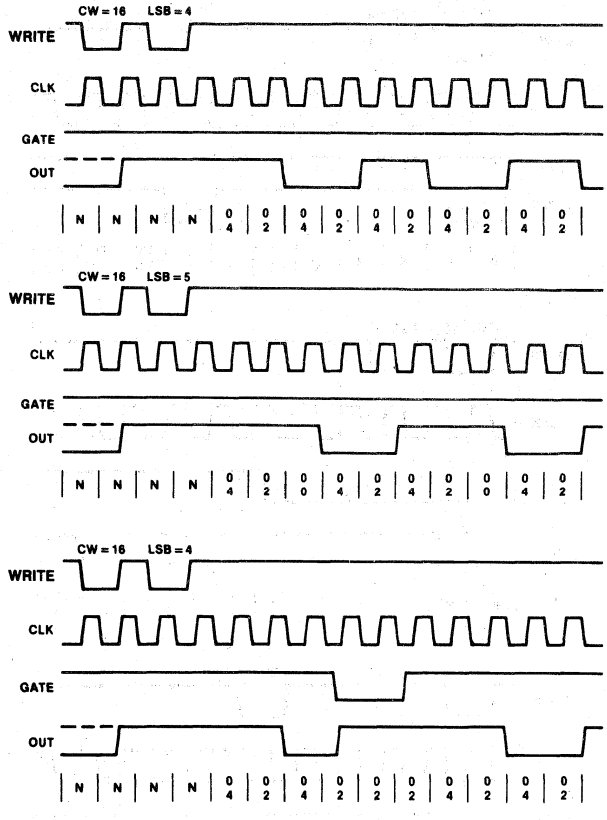
There is a slight difference in operation depending on whether the initial count is EVEN or ODD. The following description is to show exactly how this mode is implemented.

EVEN COUNTS:

OUT is initially HIGH. The initial count is loaded on one CLK pulse and is decremented by two on succeeding CLK pulses. When the count expires (decremented to 2), OUT changes to LOW and the timer is reloaded with the initial count. The above process is repeated indefinitely.

ODD COUNTS:

OUT is initially HIGH. The initial count minus one (which is an even number) is loaded on one CLK pulse and is decremented by two on succeeding CLK pulses. One CLK pulse after the count expires (decremented to 2), OUT goes LOW and the timer is loaded with the initial count minus one again. Succeeding CLK pulses decrement the count by two. When the count expires, OUT goes HIGH immediately and the timer is reloaded with the initial count minus one. The above process is repeated indefinitely. So for ODD counts, OUT will HIGH or $(N+1)/2$ counts and LOW for $(N-1)/2$ counts.



NOTE:
A GATE transition should not occur one clock prior to terminal count.

Figure 5-6. Mode 3

5.3.5 MODE 4–INITIAL COUNT TRIGGERED STROBE

This mode allows a strobe pulse to be generated by writing an initial count to the timer. Initially, OUT will be HIGH. When a new initial count is written into the timer, the counting sequence will begin. When the initial count expires (decremented to 1), OUT will go LOW for one CLK pulse and then go HIGH again.

Again, GATE=HIGH enables counting while GATE = LOW disables counting. GATE has no effect on OUT.

After writing the Control Word and initial count, the timer will be loaded on the next CLK pulse. This CLK pulse does not decrement the count, so for an initial count of N, OUT does not strobe LOW until N+1 CLK pulses after initial count is written.

If a new count is written during counting, it will be loaded in the next CLK pulse and counting will continue from the new count.

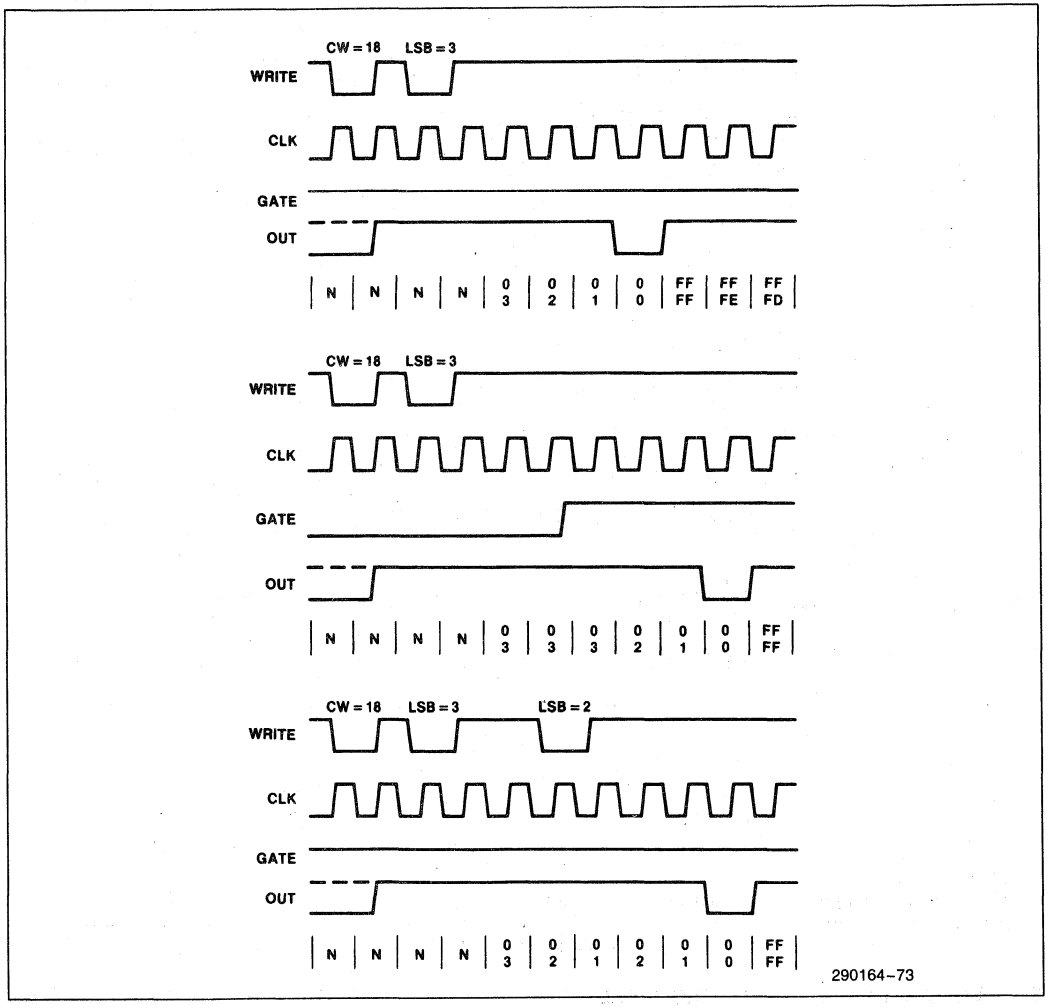


Figure 5-7. Mode 4

If a two-byte count is written, the following will occur:

1. Writing the first byte has no effect on counting.

2. Writing the second byte allows the new count to be loaded on the next CLK pulse.

OUT will strobe LOW N+1 CLK pulses after the new count of N is written. Therefore, when the strobe pulse will occur after a trigger depends on the value of the initial count loaded.

5.3.6 MODE 5–GATE RETRIGGERABLE STROBE

Mode 5 is very similar to Mode 4 except the count sequence is triggered by the gate signal instead of by writing an initial count. Initially, OUT will be HIGH. Counting is triggered by a rising edge of GATE. When the initial count has expired (decremented to 1), OUT will go LOW for one CLK pulse and then go HIGH again.

After loading the Control Word and initial count, the Count Element will not be loaded until the CLK pulse after a trigger. This CLK pulse does not decrement the count. Therefore, for an initial count of N, OUT does not strobe LOW until N+1 CLK pulses after a trigger.

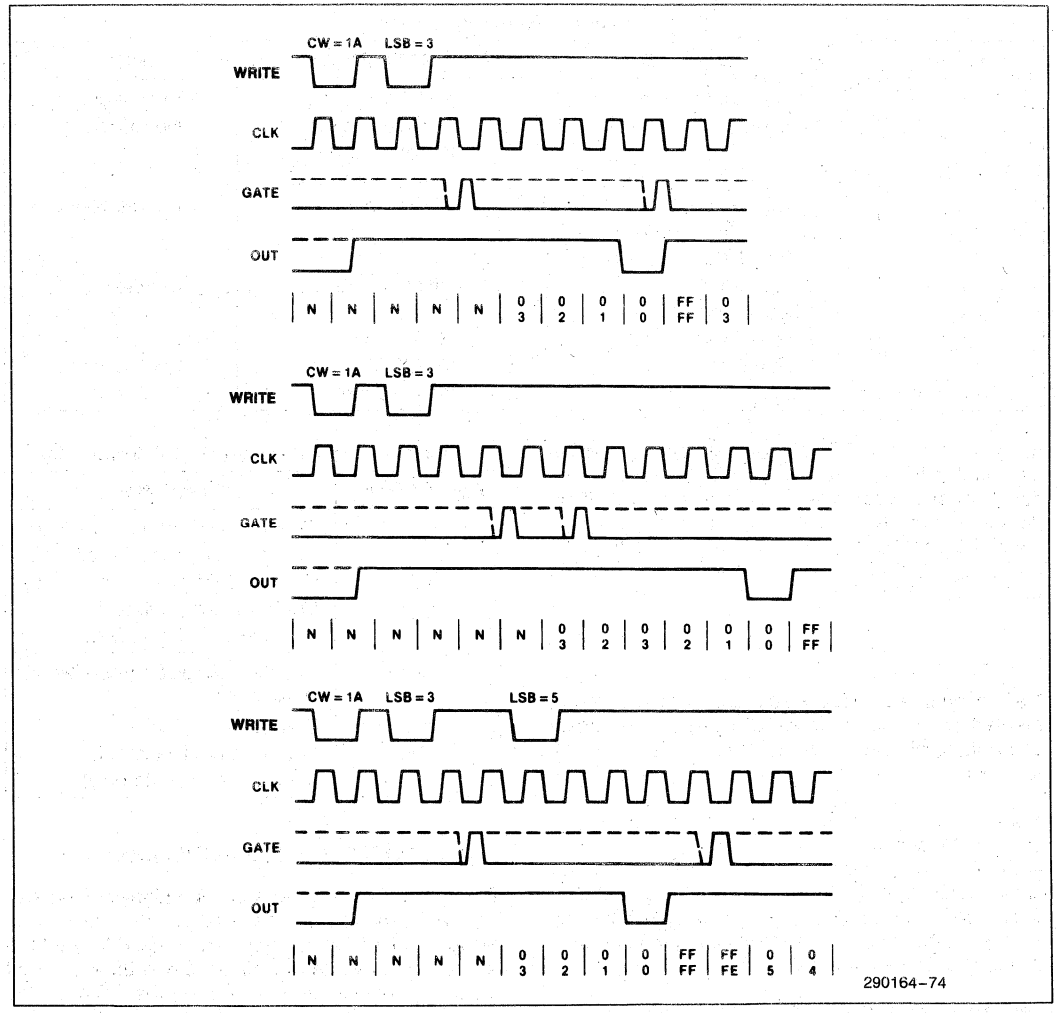


Figure 5-8. Mode 5

The counting sequence is retriggerable. Every trigger will result in the timer being loaded with the initial count on the next CLK pulse.

If the new count is written during counting, the current counting sequence will not be affected. If a trigger occurs after the new count is written but before the current count expires, the timer will be loaded with the new count on the next CLK pulse and a new count sequence will start from there.

5.3.7 OPERATION COMMON TO ALL MODES

5.3.7.1 GATE

The GATE input is always sampled on the rising edge of CLKIN. In Modes 0, 2, 3 and 4, the GATE input is level sensitive. The logic level is sampled on the rising edge of CLKIN. In Modes 1, 2, 3 and 5, the GATE input is rising edge sensitive. In these modes,

Summary of Gate Operations

Mode	GATE LOW or Going LOW	GATE Rising	HIGH
0	Disable count	No Effect	Enable count
1	No Effect	1. Initiate count 2. Reset output after next clock	No Effect
2	1. Disable count 2. Sets output HIGH immediately	Initiate count	Enable count
3	1. Disable count 2. Sets output HIGH immediately	Initiate count	Enable count
4	Disable count	No Effect	Enable count
5	No Effect	Initiate count	No Effect

a rising edge of GATE (trigger) sets an edge sensitive flip-flop in the timer. The flip-flop is reset immediately after it is sampled. This way, a trigger will be detected no matter when it occurs; i.e. a HIGH logic level does not have to be maintained until the next rising edge of CLKIN. Note that in Modes 2 and 3, the GATE input is both edge and level sensitive.

5.3.7.2 Counter

New counts are loaded and counters are decremented on the falling edge of CLKIN. The largest possible initial count is 0. This is equivalent to 2**16 for binary counting and 10**4 for BCD counting.

Note that the counter does not stop when it reaches zero. In Modes 0, 1, 4 and 5, the counter 'wraps around' to the highest count: either FFFF Hex for binary counting or 9999 for BCD counting, and continues counting. Modes 2 and 3 are periodic. The counter reloads itself with the initial count and continues counting from there.

The minimum and maximum initial count in each counter depends on the mode of operation. They are summarized below.

Mode	Min	Max
0	1	0
1	1	0
2	2	0
3	2	0
4	1	0
5	1	0

5.4 Register Set Overview

The Programmable Interval Timer module of the 82370 contains a set of six registers. The port address map of these registers is shown in Table 5-2.

Table 5-2. Timer Register Port Address Map

Port Address	Description
40H	Counter 0 Register (read/write)
41H	Counter 1 Register (read/write)
42H	Counter 2 Register (read/write)
43H	Control Word Register I (Counter 0, 1 & 2) (write-only)
44H	Counter 3 Register (read/write)
45H	Reserved
46H	Reserved
47H	Control Word Register II (Counter 3) (write-only)

5.4.1 COUNTER 0, 1, 2, 3 REGISTERS

These four 8-bit registers are functionally identical. They are used to write the initial count value into the respective timer. Also, they can be used to read the latched count value of a timer. Since they are 8-bit registers, reading and writing of the 16-bit initial count must follow the count format specified in the Control Word Registers; i.e. least significant byte only, most significant byte only, or least significant byte then most significant byte (see Programming).

5.4.2 CONTROL WORD REGISTER I & II

There are two Control Word Registers associated with the Timer section. One of the two registers (Control Word Register I) is used to control the operations of Counters 0, 1 and 2 and the other (Control Word Register II) is for Counter 3. The major functions of both Control Word Registers are listed below:

— Select the timer to be programmed.
— Define which mode the selected timer is to operate in.
— Define the count sequence; i.e. if the selected timer is to count as a Binary Counter or a Binary Coded Decimal (BCD) Counter.
— Select the byte access sequence during timer read/write operations; i.e. least significant byte only, most significant only, or least significant byte first, then most significant byte.

Also, the Control Word Registers can be programmed to perform a Counter Latch Command or a Read Back Command which will be described later.

5.5 Programming

5.5.1 INITIALIZATION

Upon power-up or reset, the state of all timers is undefined. The mode, count value, and output of all timers are random. From this point on, how each timer operates is determined solely by how it is programmed. Each timer must be programmed before it can be used. Since the outputs of some timers can generate interrupt signals to the 82370, all timers should be initialized to a known state.

Counters are programmed by writing a Control Word into their respective Control Word Registers. Then, an Initial Count can be written into the corresponding Count Register. In general, the programming procedure is very flexible. Only two conventions need to be remembered:

1. For each timer, the Control Word must be written before the initial count is written.

2. The 16-bit initial count must follow the count format specified in the Control Word (least significant byte only, most significant byte only, or least significant byte first, followed by most significant byte).

Since the two Control Word Registers and the four Counter Registers have separate addresses, and each timer can be individually selected by the appropriate Control Word Register, no special instruction sequence is required. Any programming sequence that follows the conventions above is acceptable.

A new initial count may be written to a timer at any time without affecting the timer's programmed mode in any way. Count sequence will be affected as described in the Modes of Operation section. Note that the new count must follow the programmed count format.

If a timer is previously programmed to read/write two-byte counts, the following precaution applies. A program must not transfer control between writing the first and second byte to another routine which also writes into the same timer. Otherwise, the read/write will result in incorrect count.

Whenever a Control Word is written to a timer, all control logic for that timer(s) is immediately reset (i.e. no CLK pulse is required). Also, the corresponding output in, TOUT#, goes to a known initial state.

5.5.2 READ OPERATION

Three methods are available to read the current count as well as the status of each timer. They are: Read Counter Registers, Counter Latch Command and Read Back Command. Below is a description of these methods.

READ COUNTER REGISTERS

The current count of a timer can be read by performing a read operation on the corresponding Counter Register. The only restriction of this read operation is that the CLKIN of the timers must be inhibited by using external logic. Otherwise, the count may be in the process of changing when it is read, giving an undefined result. Note that since all four timers are sharing the same CLKIN signal, inhibiting CLKIN to read a timer will unavoidably disable the other timers also. This may prove to be impractical. Therefore, it is suggested that either the Counter Latch Command or the Read Back Command can be used to read the current count of a timer.

Another alternative is to temporarily disable a timer before reading its Counter Register by using the GATE input. Depending on the mode of operation, GATE=LOW will disable the counting operation. However, this option is available on Timer 2 and 3 only, since the GATE signals of the other two timers are internally enabled all the time.

COUNTER LATCH COMMAND

A Counter Latch Command will be executed whenever a special Control Word is written into a Control Word Register. Two bits written into the Control Word Register distinguish this command from a 'regular' Control Word (see Register Bit Definition). Also, two other bits in the Control Word will select which counter is to be latched.

Upon execution of this command, the selected counter's Output Latch (OL) latches the count at the time the Counter Latch Command is received. This

count is held in the latch until it is read by the 80376, or until the timer is reprogrammed. The count is then unlatched automatically and the OL returns to "following" the Counting Element (CE). This allows reading the contents of the counters "on the fly" without affecting counting in progress. Multiple Counter Latch Commands may be used to latch more than one counter. Each latched count is held until it is read. Counter Latch Commands do not affect the programmed mode of the timer in any way.

If a counter is latched, and at some time later, it is latched again before the prior latched count is read, the second Counter Latch Command is ignored. The count read will then be the count at the time the first command was issued.

In any event, the latched count must be read according to the programmed format. Specifically, if the timer is programmed for two-byte counts, two bytes must be read. However, the two bytes do not have to be read right after the other. Read/write or programming operations of other timers may be performed between them.

Another feature of this Counter Latch Command is that read and write operations of the same timer may be interleaved. For example, if the timer is programmed for two-byte counts, the following sequence is valid.

1. Read least significant byte.
2. Write new least significant byte.
3. Read most significant byte.
4. Write new most significant byte.

If a timer is programmed to read/write two-byte counts, the following precaution applies. A program must not transfer control between reading the first and second byte to another routine which also reads from that same timer. Otherwise, an incorrect count will be read.

READ BACK COMMAND

The Read Back Command is another special Command Word operation which allows the user to read the current count value and/or the status of the selected timer(s). Like the Counter Latch Command, two bits in the Command Word identify this as a Read Back Command (see Register Bit Definition).

The Read Back Command may be used to latch multiple counter Output Latches (OL's) by selecting more than one timer within a Command Word. This single command is functionally equivalent to several Counter Latch Commands, one for each counter to be latched. Each counter's latched count will be held until it is read by the 80376 or until the timer is reprogrammed. The counter is automatically unlatched when read, but other counters remain latched until they are read. If multiple Read Back commands are issued to the same timer without reading the count, all but the first are ignored; i.e. the count read will correspond to the very first Read Back Command issued.

As mentioned previously, the Read Back Command may also be used to latch status information of the selected timer(s). When this function is enabled, the status of a timer can be read from the Counter Register after the Read Back Command is issued. The status information of a timer includes the following:

1. Mode of timer:

   This allows the user to check the mode of operation of the timer last programmed.

2. State of TOUT pin of the timer:

   This allows the user to monitor the counter's output pin via software, possibly eliminating some hardware from a system.

3. Null Count/Count available:

   The Null Count Bit in the status byte indicates if the last count written to the Count Register (CR) has been loaded into the Counting Element (CE). The exact time this happens depends on the mode of the timer and is described in the Programming section. Until the count is loaded into the Counting Element (CE), it cannot be read from the timer. If the count is latched or read before this occurs, the count value will not reflect the new count just written.

If multiple status latch operations of the timer(s) are performed without reading the status, all but the first command are ignored; i.e. the status read in will correspond to the first Read Back Command issued.

Both the current count and status of the selected timer(s) may be latched simultaneously by enabling both functions in a single Read Back Command. This is functionally the same as issuing two separate Read Back Commands at once. Once again, if multiple read commands are issued to latch both the count and status of a timer, all but the first command will be ignored.

If both count and status of a timer are latched, the first read operation of that timer will return the latched status, regardless of which was latched first. The next one or two (if two count bytes are to be read) read operations return the latched count. Note that subsequent read operations on the Counter Register will return the unlatched count (like the first read method discussed).

5.6 Register Bit Definitions

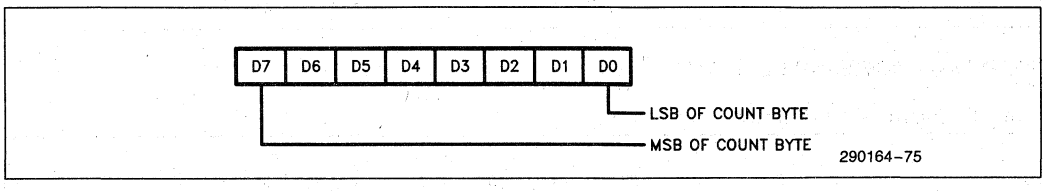
COUNTER 0, 1, 2, 3 REGISTER (READ/WRITE)

Port Address	Description
40H	Counter 0 Register (read/write)
41H	Counter 1 Register (read/write)
42H	Counter 2 Register (read/write)
44H	Counter 3 Register (read/write)
45H	Reserved
46H	Reserved

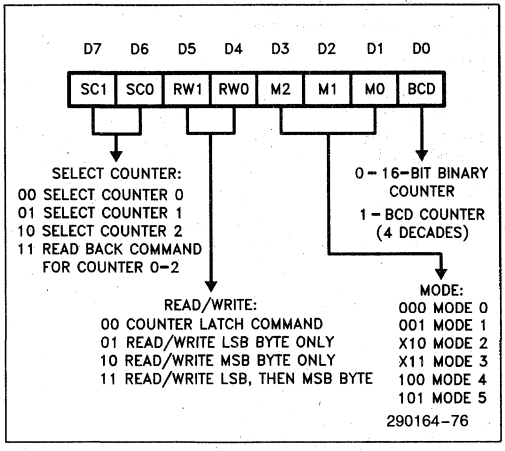
Note that these 8-bit registers are for writing and reading of one byte of the 16-bit count value, either the most significant or the least significant byte.

CONTROL WORD REGISTER I & II (WRITE-ONLY)

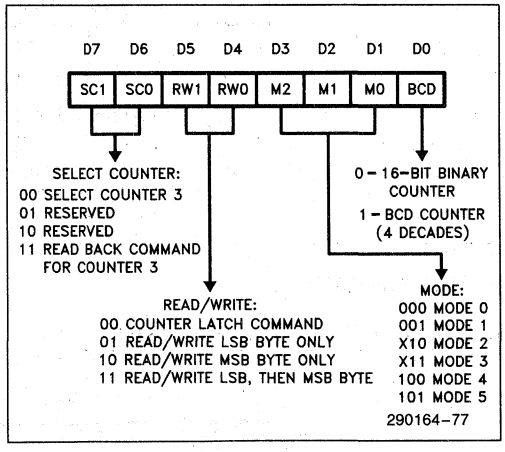
Port Address	Description
43H	Control Word Register I (Counter 0, 1, 2 (write-only)
47H	Control Word Register II (Counter 3) (write-only)



Control Word Register I



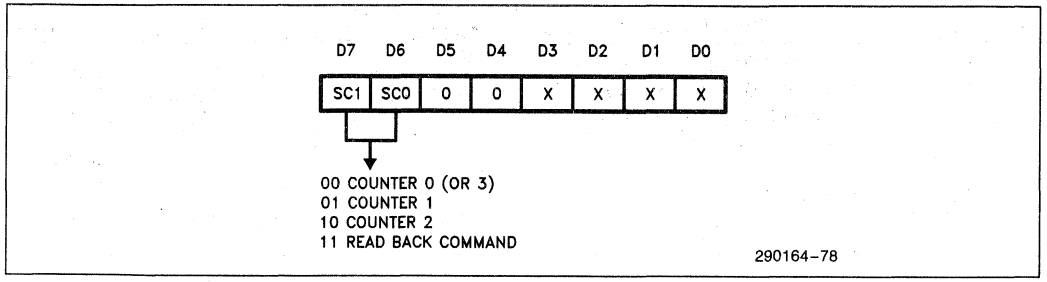
Control Word Register II



5

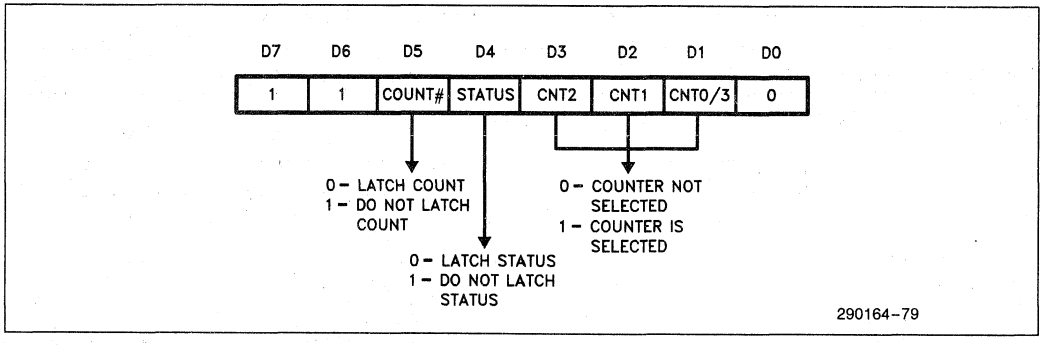
COUNTER LATCH COMMAND FORMAT

(Write to Control Word Register)



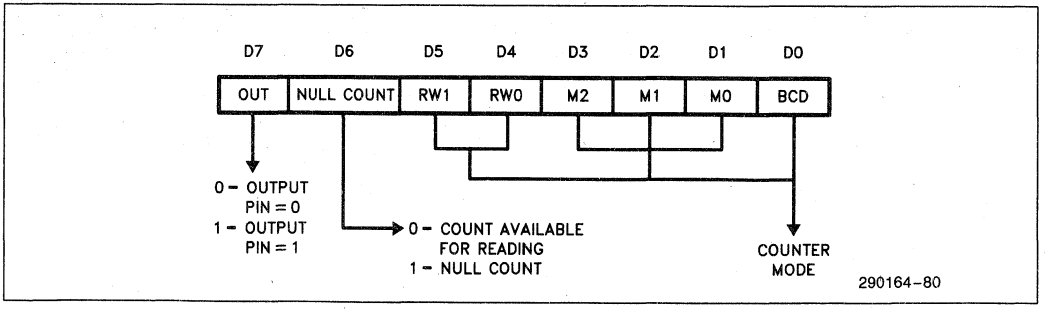
READ BACK COMMAND FORMAT

(Write to Control Word Register)



STATUS FORMAT

(Returned from Read Back Command)



6.0 WAIT STATE GENERATOR

6.1 Functional Description

The 82370 contains a programmable Wait State Generator which can generate a pre-programmed number of wait states during both CPU and DMA initiated bus cycles. This Wait State Generator is capable of generating 1 to 16 wait states in non-pipelined mode, and 0 to 15 wait states in pipelined mode. Depending on the bus cycle type and the two Wait State Control inputs (WSC 0–1), a pre-programmed number of wait states in the selected Wait State Register will be generated.

The Wait State Generator can also be disabled to allow the use of devices capable of generating their own READY# signals. Figure 6-1 is a block diagram of the Wait State Generator.

6.2 Interface Signals

The following describes the interface signals which affect the operation of the Wait State Generator. The READY#, WSC0 and WSC1 signals are inputs. READYO# is the ready output signal to the host processor.

6.2.1 READY#

READY# is an active LOW input signal which indicates to the 82370 the completion of a bus cycle. In the Master mode (e.g. 82370 initiated DMA transfer), this signal is monitored to determine whether a peripheral or memory needs wait states inserted in the current bus cycle. In the Slave mode, it is used (together with the ADS# signal) to trace CPU bus cycles to determine if the current cycle is pipelined.

6.2.2 READYO#

READYO# (Ready Out#) is an active LOW output signal and is the output of the Wait State Generator. The number of wait states generated depends on the WSC(0–1) inputs. Note that special cases are handled for access to the 82370 internal registers and for the Refresh cycles. For 82370 internal register access, READYO# will be delayed to take into the command recovery time of the register. One or more wait states will be generated in a pipelined cycle. During refresh, the number of wait states will be determined by the preprogrammed value in the Refresh Wait State Register.

In the simplest configuration, READYO# can be connected to the READY# input of the 82370 and the 80376 CPU. This is, however, not always the case. If external circuitry is to control the READY# inputs as well, additional logic will be required (see Application Issues).

6.2.3 WSC(0–1)

These two Wait State Control inputs, together with the M/IO# input, select one of the three pre-programmed 8-bit Wait State Registers which determines the number of wait states to be generated. The most significant half of the three Wait State Registers corresponds to memory accesses, the least significant half to I/O accesses. The combination WSC(0–1) = 11 disables the Wait State Generator.

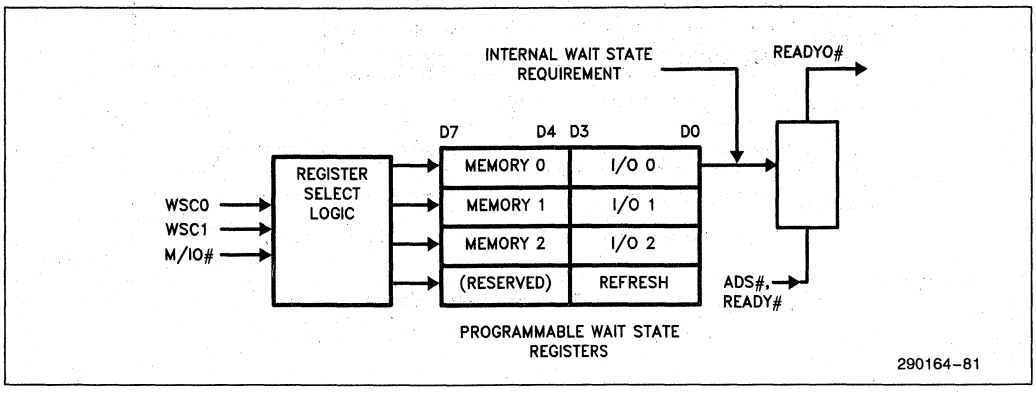


Figure 6-1. Wait State Generator Block Diagram

6.3 Bus Function

6.3.1 WAIT STATES IN NON-PIPELINED CYCLE

The timing diagram of two typical non-pipelined cycles with 82370 generated wait states is shown in Figure 6-2. In this diagram, it is assumed that the internal registers of the 82370 are not addressed. During the first T2 state of each bus cycle, the Wait State Control and the M/IO# inputs are sampled to determine which Wait State Register (if any) is selected. If the WSC inputs are active (i.e. not both are driven HIGH), the pre-programmed number of wait states corresponding to the selected Wait State Register will be requested. This is done by driving the READYO# output HIGH during the end of each T2 state.

The WSC (0–1) inputs need only be valid during the very first T2 state of each non-pipelined cycle. As a general rule, the WSC inputs are sampled on the rising edge of the next clock (82384 CLK) after the last state when ADS# (Address Status) is asserted.

The number of wait states generated depends on the type of bus cycle, and the number of wait states requested. The various combinations are discussed below.

1. Access the 82370 internal registers: 2 to 5 wait states, depending upon the specific register addressed. Some back-to-back sequences to the Interrupt Controller will require 7 wait states.

2. Interrupt Acknowledge to the 82370: 5 wait states.

3. Refresh: As programmed in the Refresh Wait State Register (see Register Set Overview). Note that if WCS (0–1) = 11, READYO# will stay inactive.

4. Other bus cycles: Depending on WCS (0–1) and M/IO# inputs, these inputs select a Wait State Register in which the number of wait states will be equal to the pre-programmed wait state count in the register plus 1. The Wait State Register selection is defined as follows (Table 6-1).

Table 6-1. Wait State Register Selection

M/IO#	WSC(0–1)	Register Selected
0	00	WAIT REG 0 (I/O half)
0	01	WAIT REG 1 (I/O half)
0	10	WAIT REG 2 (I/O half)
1	00	WAIT REG 0 (MEM half)
1	01	WAIT REG 1 (MEM half)
1	10	WAIT REG 2 (MEM half)
X	11	Wait State Gen. Disabled

The Wait State Control signals, WSC (0–1), can be generated with the address decode and the Read/Write control signals as shown in Figure 6-3.

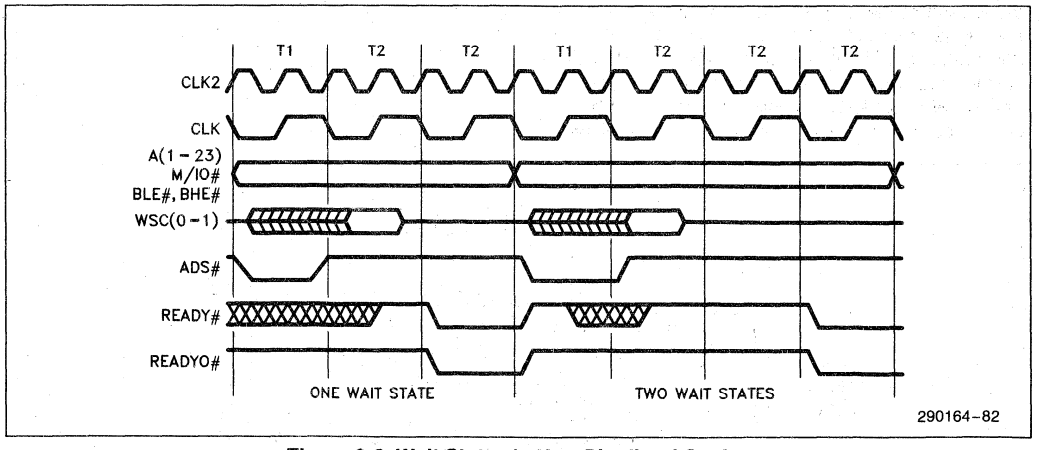


Figure 6-2. Wait States in Non-Pipelined Cycles

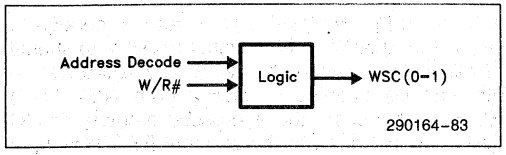


Figure 6-3. WSC (0-1) Generation

Note that during HALT and SHUTDOWN, the number of wait states will depend on the WSC (0-1) inputs, which will select the memory half of one of the Wait State Registers (see CPU Reset and Shutdown Detect).

6.3.2 WAIT STATES IN PIPELINED CYCLES

The timing diagram of two typical pipelined cycles with 82370 generated wait states is shown in Figure 6-4. Again, in this diagram, it is assumed that the 82370 internal registers are not addressed. As defined in the timing of the 80376 processor, the Address (A1-23), Byte Enable (BHE#, BLE#), and other control signals (M/IO#, ADS#) are asserted one T-state earlier than in a non-pipelined cycle; i.e. they are asserted at T2P. Similar to the non-pipelined case, the Wait State Control (WSC) inputs are sampled in the middle of the state after the last state the ADS# signal is asserted. Therefore, the WSC inputs should be asserted during the T1P state of each pipelined cycle (which is one T-state earlier than in the non-pipelined cycle).

The number of wait states generated in a pipelined cycle is selected in a similar manner as in the non-pipelined case discussed in the previous section. The only difference here is that the actual number of wait states generated will be one less than that of the non-pipelined cycle. This is done automatically by the Wait State Generator.

6.3.3 EXTENDING AND EARLY TERMINATING BUS CYCLE

The 82370 allows external logic to either add wait states or cause early termination of a bus cycle by controlling the READY# input to the 82370 and the host processor. A possible configuration is shown in Figure 6-5.

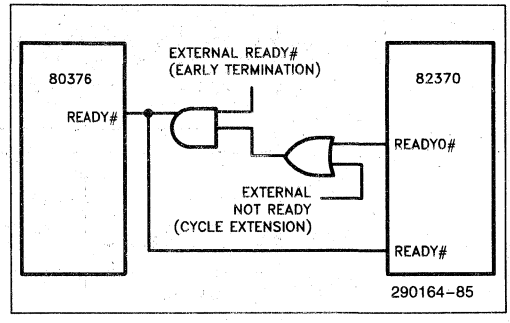


Figure 6-5. External 'READY' Control Logic

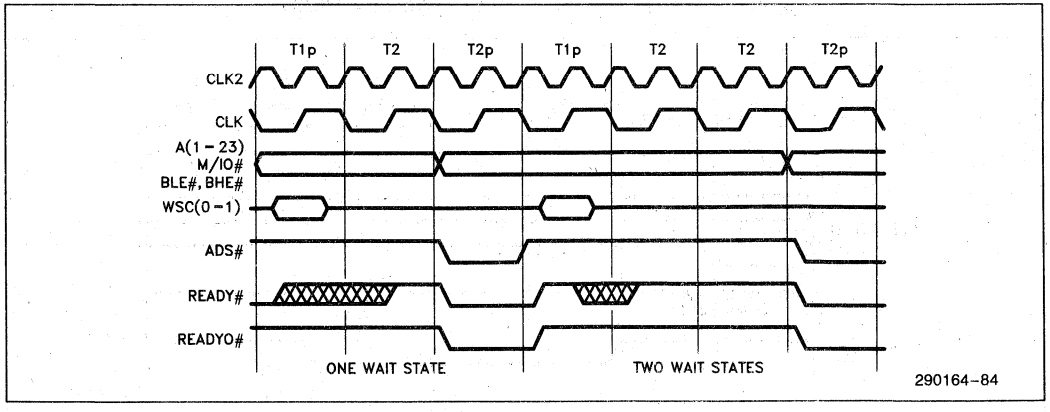


Figure 6-4. Wait States in Pipelined Cycles

The EXT. RDY# (External Ready) signal of Figure 6-5 allows external devices to cause early termination of a bus cycle. When this signal is asserted LOW, the output of the circuit will also go LOW (even though the READYO# of the 82370 may still be HIGH). This output is fed to the READY# input of the 80376 and the 82370 to indicate the completion of the current bus cycle.

Similarly, the EXT. NOT READY (External Not Ready) signal is used to delay the READY# input of the processor and the 82370. As long as this signal is driven HIGH, the output of the circuit will drive the READY# input HIGH. This will effectively extend the duration of a bus cycle. However, it is important to note that if the two-level logic is not fast enough to satisfy the READY# setup time, the OR gate should be eliminated. Instead, the 82370 Wait State Generator can be disabled by driving both WSC (0–1) HIGH. In this case, the addressed memory or I/O device should activate the external READY# input whenever it is ready to terminate the current bus cycle.

Figures 6-6 and 6-7 show the timing relationships of the ready signals for the early termination and extension of the bus cycles. Section 6-7, Application Issues, contains a detailed timing analysis of the external circuit.

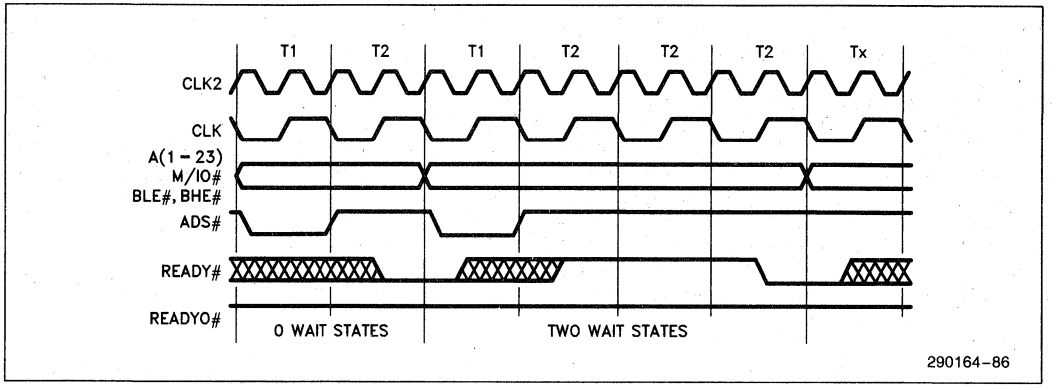


Figure 6-6. Early Termination of Bus Cycle By 'READY#'

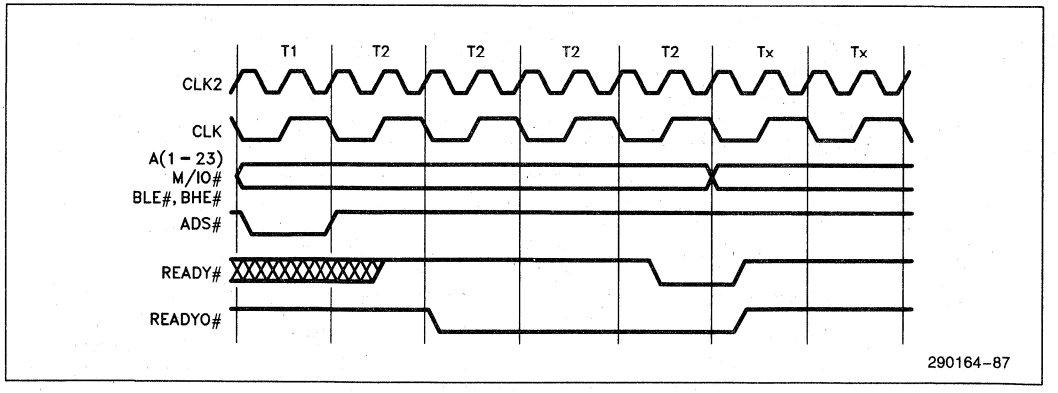


Figure 6-7. Extending Bus Cycle by 'READY#'

Due to the following implications, it should be noted that early termination of bus cycles in which 82370 internal registers are accessed is not recommended.

1. Erroneous data may be read from or written into the addressed register.

2. The 82370 must be allowed to recover either before HLDA (Hold Acknowledge) is asserted or before another bus cycle into an 82370 internal register is initiated.

The recovery time, in clock periods, equals the remaining wait states that were avoided plus 4.

6.4 Register Set Overview

Altogether, there are four 8-bit internal registers associated with the Wait State Genertor. The port address map of these registers is shown below in Table 6-2. A detailed description of each follows.

Table 6-2. Register Address Map

Port Address	Description
72H	Wait State Reg 0 (read/write)
73H	Wait State Reg 1 (read/write)
74H	Wait State Reg 2 (read/write)
75H	Ref. Wait State Reg (read/write)

WAIT STATE REGISTER 0, 1, 2

These three 8-bit read/write registers are functionally identical. They are used to store the pre-programmed wait state count. One half of each register contains the wait state count for I/O accesses while the other half contains the count for memory accesses. The total number of wait states generated will depend on the type of bus cycle. For a non-pipelined cycle, the actual number of wait states requested is equal to the wait state count plus 1. For a pipelined cycle, the number of wait states will be equal to the wait state count in the selected register. Therefore, the Wait State Generator is capable of generating 1 to 16 wait states in non-pipelined mode, and 0 to 15 wait states in pipelined mode.

Note that the minimum wait state count in each register is 0. This is equivalent to 0 wait states for a pipelined cycle and 1 wait state for a non-pipelined cycle.

REFRESH WAIT STATE REGISTER

Similar to the Wait State Registers discussed above, this 4-bit register is used to store the number of wait states to be generated during a DRAM refresh cycle.

Note that the Refresh Wait State Register is not selected by the WSC inputs. It will automatically be chosen whenever a DRAM refresh cycle occurs. If the Wait State Generator is disabled during the refresh cycle (WSC (0-1) = 11), READYO# will stay inactive and the Refresh Wait State Register is ignored.

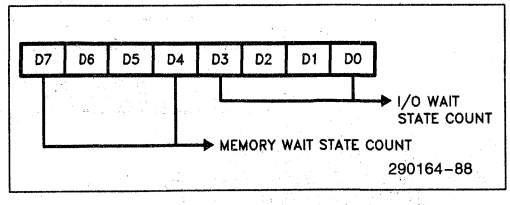
6.5 Programming

Using the Wait State Generator is relatively straightforward. No special programming sequence is required. In order to ensure the expected number of wait states will be generated when a register is selected, the registers to be used must be programmed after power-up by writing the appropriate wait state count into each register. Note that upon hardware reset, all Wait State Registers are initialized with the value FFH, giving the maximum number of wait states possible. Also, each register can be read to check the wait state count previously stored in the register.

6.6 Register Bit Definition

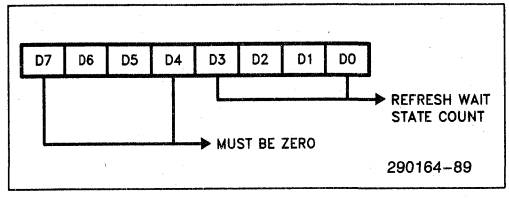
WAIT STATE REGISTER 0, 1, 2

Port Address	Description
72H	Wait State Register 0 (read/write)
73H	Wait State Register 1 (read/write)
74H	Wait State Register 2 (read/write)



REFRESH WAIT STATE REGISTER

Port Address: 75H (Read/Write)



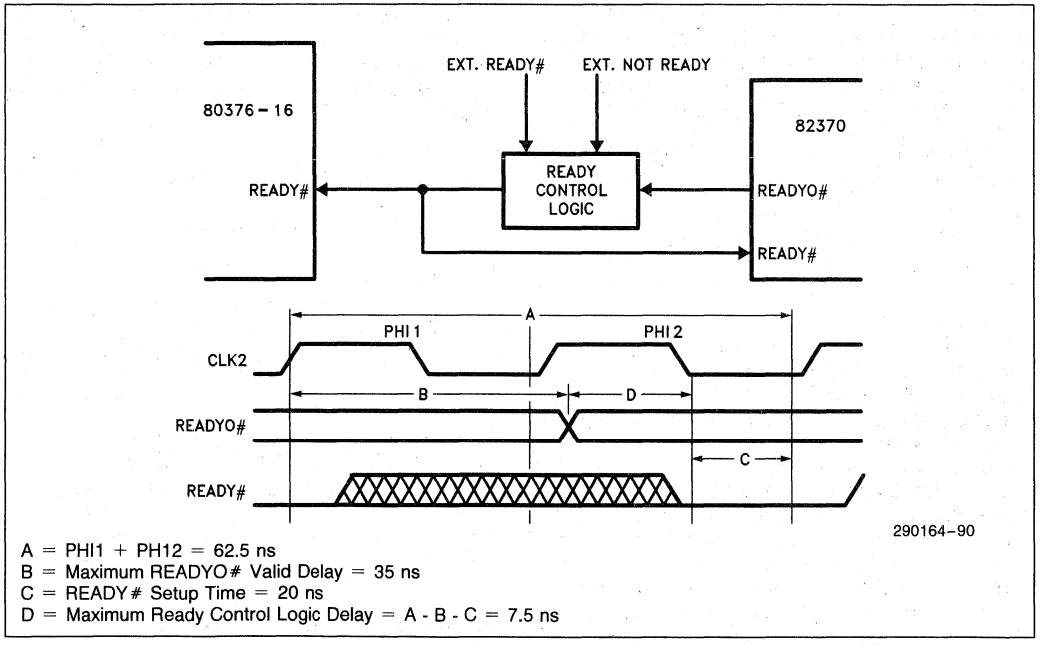
6.7 Application Issues

6.7.1 EXTERNAL 'READY' CONTROL LOGIC

As mentioned in section 6.3.3, wait state cycles generated by the 82370 can be terminated early or extended longer by means of additional external logic (see Figure 6-5). In order to ensure that the READY# input timing requirement of the 80376 and the 82370 is satisfied, special care must be taken when designing this external control logic. This section addresses the design requirements.

A simplified block diagram of the external logic along with the READY# timing diagram is shown in Figure 6-8. The purpose is to determine the maximum delay time allowed in the external control logic in order to satisfy the READY# setup time.

First, it will be assumed that the 80376 is running at 16 MHz (i.e. CLK2 is 32 MHz). Therefore, one bus state (two CLK2 periods) will be equivalent to 62.5 ns. According to the AC specifications of the 82370, the maximum delay time for valid READYO# signal is 31 ns after the rising edge of CLK2 in the beginning of T2 (for non-pipelined cycle) or T2P (for pipelined cycle). Also, the minimum READY# setup time of the 80376 and the 82370 should be 19 ns before the rising edge of CLK2 at the beginning of the next bus state. This limits the total delay time for the external READY# control logic to be 12.5 ns (62.5–31–19) in order to meet the READY# setup timing requirement.



A = PHI1 + PH12 = 62.5 ns
B = Maximum READYO# Valid Delay = 35 ns
C = READY# Setup Time = 20 ns
D = Maximum Ready Control Logic Delay = A - B - C = 7.5 ns

Figure 6-8. 'READY' Timing Consideration

7.0 DRAM REFRESH CONTROLLER

7.1 Functional Description

The 82370 DRAM Refresh Controller consists of a 24-bit Refresh Address Counter and Refresh Request logic for DRAM refresh operations (see Figure 7-1). TIMER 1 can be used as a trigger signal to the DRAM Refresh Request logic. The Refresh Bus Size can be programmed to be 8- or 16-bit wide. Depending on the Refresh Bus Size, the Refresh Address Counter will be incremented with the appropriate value after every refresh cycle. The internal logic of the 82370 will give the Refresh operation the highest priority in the bus control arbitration process. Bus control is not released and re-requested if the 82370 is already a bus master.

7.2 Interface Signals

7.2.1 TOUT1/REF#

The dual function output pin of TIMER 1 (TOUT1/REF#) can be programmed to generate DRAM Refresh signal. If this feature is enabled, the rising edge of TIMER 1 output (TOUT1#) will trigger the DRAM Refresh Request logic. After some delay for gaining access of the bus, the 82370 DRAM Controller will generate a DRAM Refresh signal by driving REF# output LOW. This signal is cleared after the refresh cycle has taken place, or by a hardware reset.

If the DRAM Refresh feature is disabled, the TOUT1/REF# output pin is simply the TIMER 1 output. Detailed information of how TIMER 1 operates is discussed in section 6—Programmable Interval Timer, and will not be repeated here.

7.3 Bus Function

7.3.1 ARBITRATION

In order to ensure data integrity of the DRAMs, the 82370 gives the DRAM Refresh signal the highest priority in the arbitration logic. It allows DRAM Refresh to interrupt DMA in progress in order to perform the DRAM Refresh cycle. The DMA service will be resumed after the refresh is done.

In case of a DRAM Refresh during a DMA process, the cascaded device will be requested to get off the bus. This is done by de-asserting the EDACK signal. Once DREQn goes inactive, the 82370 will perform the refresh operation. Note that the DMA controller does not completely relinquish the system bus during refresh. The Refresh Generator simply "steals" a bus cycle between DMA accesses.

Figure 7-2 shows the timing diagram of a Refresh Cycle. Upon expiration of TIMER 1, the 82370 will try to take control of the system bus by asserting HOLD. As soon as the 82370 see HLDA go active, the DRAM Refresh Cycle will be carried out by activating the REF# signal as well as the address and control signals on the system bus (Note that REF# will not be active until two CLK periods HLDA is asserted). The address bus will contain the 24-bit ad-

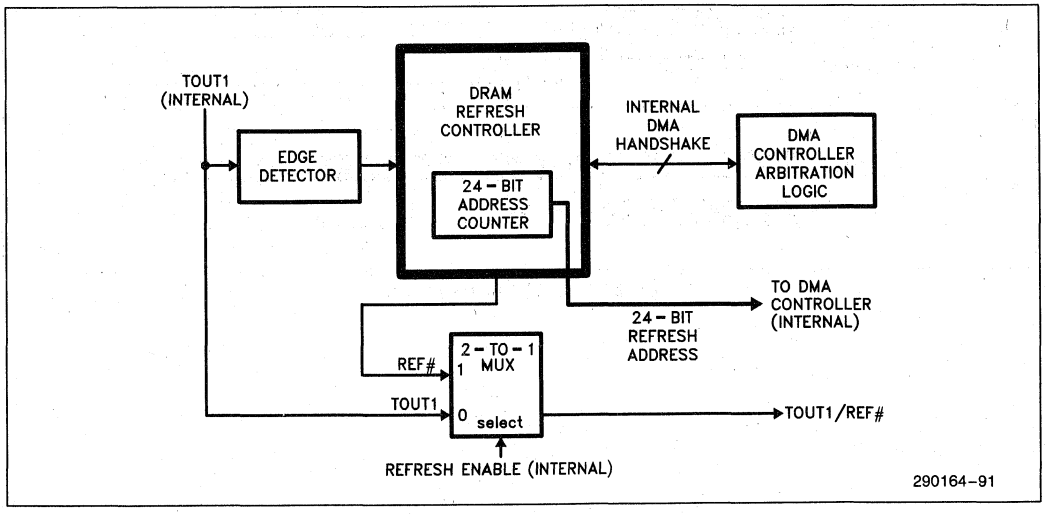


Figure 7-1. DRAM Refresh Controller

dress currently in the Refresh Address Counter. The control signals are driven the same way as in a Memory Read cycle. This "read" operation is complete when the READY# signal is driven LOW. Then, the 82370 will relinquish the bus by de-asserting HOLD. Typically, a Refresh Cycle without wait states will take five bus states to execute. If "n" wait states are added, the Refresh Cycle will last for five plus "n" bus states.

How often the Refresh Generator will initiate a refresh cycle depends on the frequency of CLKIN as will as TIMER 1's programmed mode of operation. For this specific application, TIMER 1 should be programmed to operate in Mode 2 to generate a constant clock rate. See section 6—Programmable Interval Timer for more information on programming the timer. One DRAM Refresh Cycle will be generated each time TIMER 1 expires (when TOUT1 changes from LOW to HIGH).

The Wait State Generator can be used to insert wait states during a refresh cycle. The 82370 will automatically insert the desired number of wait states as programmed in the Refresh Wait State Register (see Wait State Generator).

7.4 Modes of Operation

7.4.1 WORD SIZE AND REFRESH ADDRESS COUNTER

The 82370 supports 8- and 16-bit refresh cycle. The bus width during a refresh cycle is programmable (see Programming). The bus size can be programmed via the Refresh Control Register (see Register Overview). If the DRAM bus size is 8- or 16-bits, the Refresh Address Counter will be incremented by 1 or 2, respectively.

The Refresh Address Counter is cleared by a hardware reset.

7.5 Register Set Overview

The Refresh Generator has two internal registers to control its operation. They are the Refresh Control Register and the Refresh Wait State Register. Their port address map is shown in Table 7-1 below.

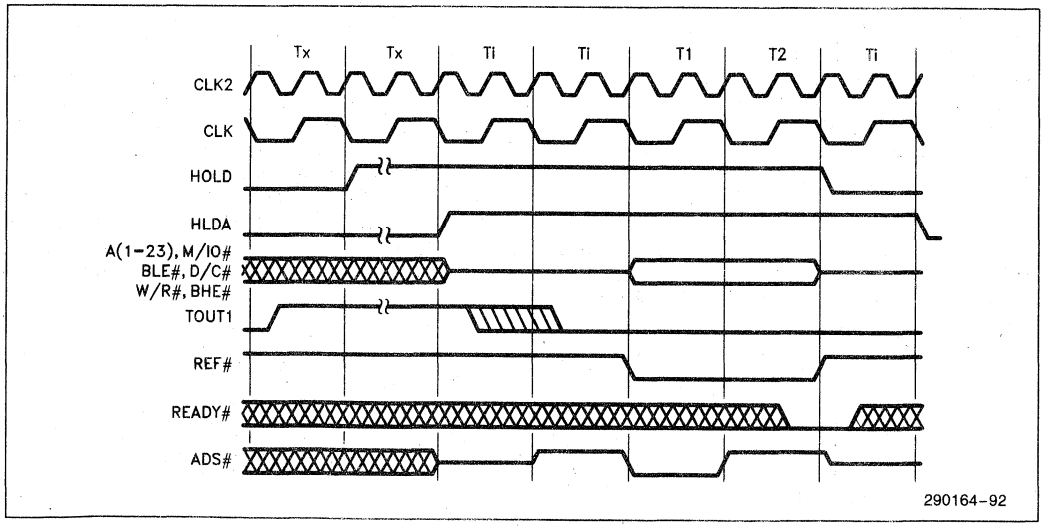


Figure 7-2. 82370 Refresh Cycle

Table 7-1. Register Address Map

Port Address	Description
1CH	Refresh Control Reg. (read/write)
75H	Ref. Wait State Reg. (read/write)

The Refresh Wait State Register is not part of the Refresh Generator. It is only used to program the number of wait states to be inserted during a refresh cycle. This register is discussed in detailed in section 7 (Wait State Generator) and will not be repeated here.

REFRESH CONTROL REGISTER

This 2-bit register serves two functions. First, it is used to enable/disable the DRAM Refresh function output. If disabled, the output of TIMER 1 is simply used as a general purpose timer. The second function of this register is to program the DRAM bus size for the refresh operation. The programmed bus size also determines how the Refresh Address Counter will be incremented after each refresh operation.

7.6 Programming

Upon hardware reset, the DRAM Refresh function is disabled (the Refresh Control Register is cleared). The following programming steps are needed before the Refresh Generator can be used. Since the rate of refresh cycles depends on how TIMER 1 is programmed, this timer must be initialized with the desired mode of operation as well as the correct refresh interval (see Programming Interval Timer). Whether or not wait states are to be generated during a refresh cycle, the Refresh Wait State Register must also be programmed with the appropriate value. Then, the DRAM Refresh feature must be enabled and the DRAM bus width should be defined. These can be done in one step by writing the appropriate control word into the Refresh Control Register (see Register Bit Definition). After these steps are done, the refresh operation will automatically be invoked by the Refresh Generator upon expiration of Timer 1.

In addition to the above programming steps, it should be noted that after reset, although the TOUT1/REF# becomes the Time 1 output, the state of this pin in undefined. This is because the Timer module has not been initialized yet. Therefore, if this output is used as a DRAM Refresh signal, this pin should be disqualified by external logic until the Refresh function is enabled. One simple solution is to logically AND this output with HLDA, since HLDA should not be active after reset.

7.7 Register Bit Definition

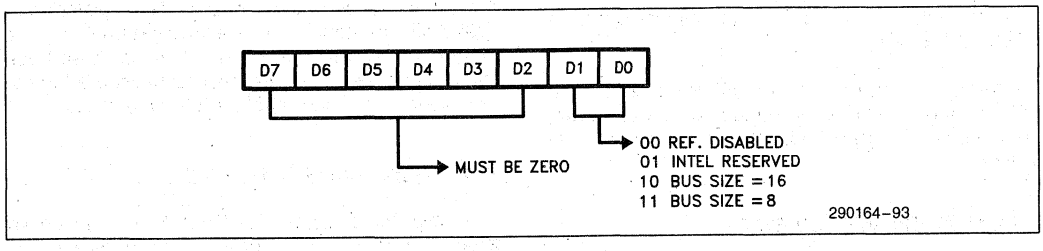
REFRESH CONTROL REGISTER

Port Address: 1CH (Read/Write)

8.0 RELOCATION REGISTER, ADDRESS DECODE, AND CHIP-SELECT (CHPSEL#)

8.1 Relocation Register

All the integrated peripheral devices in the 82370 are controlled by a set of internal registers. These registers span a total of 256 consecutive address locations (although not all the 256 locations are used). The 82370 provides a Relocation Register which allows the user to map this set of internal registers into either the memory or I/O address space. The function of the Relocation Register is to define the base address of the internal register set of the 82370 as well as if the registers are to be memory- or I/O-mapped. The format of the Relocation Register is depicted in Figure 8-1.



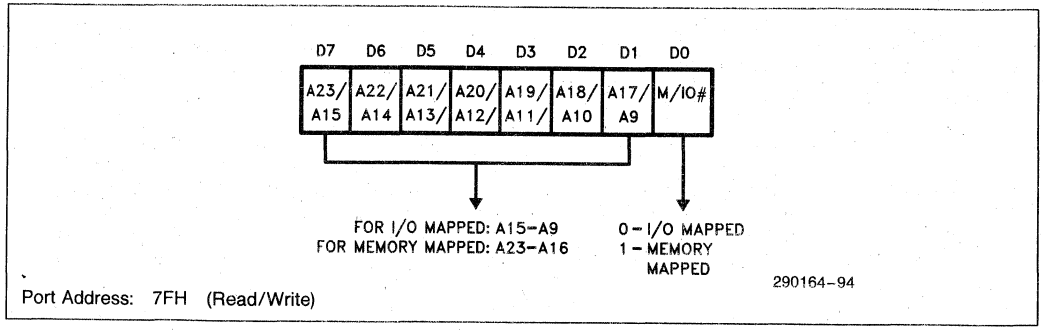


Figure 8-1. Relocation Register

Note that the Relocation Register is part of the internal register set of the 82370. It has a port address of 7FH. Therefore, any time the content of the Relocation Register is changed, the physical location of this register will also be moved. Upon reset of the 82370, the content of the Relocation Register will be cleared. This implies that the 82370 will respond to its I/O addresses in the range of 0000H to 00FFH.

8.1.1 I/O-MAPPED 82370

As shown in the figure, Bit 0 of the Relocation Register determines whether the 82370 registers are to be memory-mapped or I/O mapped. When Bit 0 is set to '0', the 82370 will respond to I/O Addresses. Address signals BHE#, BLE#, A1–A7 will be used to select one of the internal registers to be accessed. Bit 1 to Bit 7 of the Relocation Register will correspond to A9 to A15 of the Address bus, respectively. Together with A8 implied to be '0', A15 to A8 will be fully decoded by the 82370. The following shows how the 82370 is mapped into the I/O address space.

Example

Relocation Register = 11001110 (0CEH)

82370 will respond to I/O address range from 0CE00H to 0CEFFH.

Therefore, this I/O mapping mechanism allows the 82370 internal registers to be located on any even, contiguous, 256 byte boundary of the system I/O space.

8.1.2 MEMORY-MAPPED 82370

When Bit 0 of the Relocation Register is set to '1', the 82370 will respond to memory addresses. Again, Address signals BHE#, BLE#, A1–A7 will be used to select one of the internal registers to be accessed. Bit 1 to Bit 7 of the Relocation Register will correspond to A17–A23, respectively. A16 is assumed to be '0', and A8–A15 are ignored. Consider the following example.

Example

Relocation Register = 10100111 (0A7H)

The 82370 will respond to memory addresses in the range of A6XX00H to A60XXFFH (where 'X' is don't care).

This scheme implies that the internal registers can be located in any even, contiguous, 2**16 byte page of the memory space.

8.2 Address Decoding

As mentioned previously, the 82370 internal registers do not occupy the entire contiguous 256 address locations. Some of the locations are 'unoccupied'. The 82370 always decodes the lower 8 address signals (BHE#, BLE#, A1–A7) to determine if any one of its registers is being accessed. If the address does not correspond to any of its registers, the 82370 will not respond. This allows external devices to be located within the 'holes' in the 82370 address space. Note that there are several unused addresses reserved for future Intel peripheral devices.

8.3 Chip-Select (CHPSEL#)

The Chip-Select signal (CHPSEL#) will go active when the 82370 is addressed in a Slave bus

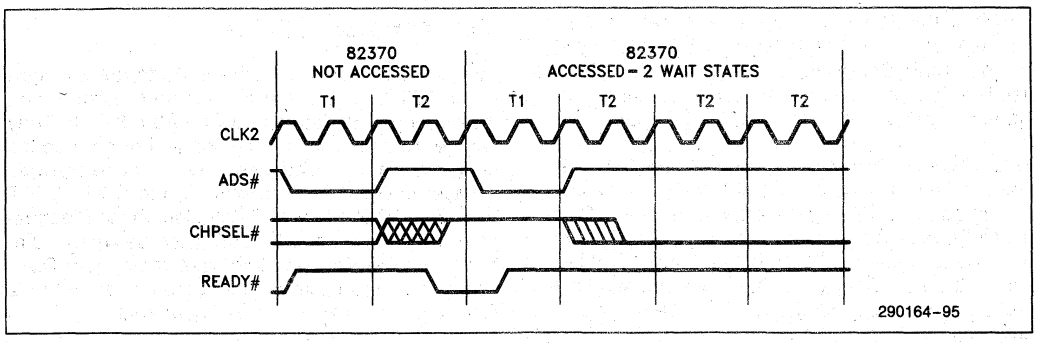


Figure 8-2. CHPSEL# Timing

cycle (either read or write), or in an interrupt acknowledge cycle in which the 82370 will drive the Data Bus. For a given bus cycle, CHPSEL# becomes active and valid in the first T2 (in a non-pipelined cycle) or in T1P (in a pipelined cycle). It will stay valid until the cycle is terminated by READY# driven active. As CHPSEL# becomes valid well before the 82370 drives the Data Bus, it can be used to control the transceivers that connect the local CPU bus to the system bus. The timing diagram of CHPSEL# is shown in Figure 8-2.

9.0 CPU RESET AND SHUTDOWN DETECT

The 82370 will activate the CPURST signal to reset the host processor when one of the following conditions occurs:

— 82370 RESET is active;

— 82370 detects a 80376 Shutdown cycle (this feature can be disabled);

— CPURST software command is issued to 80376.

Whenever the CPURST signal is activated, the 82370 will reset its own internal Slave-Bus state machine.

9.1 Hardware Reset

Following a hardware reset, the 82370 will assert its CPURST output to reset the host processor. This output will stay active for as long as the RESET input is active. During a hardware reset, the 82370 internal registers will be initialized as defined in the corresponding functional descriptions.

9.2 Software Reset

CPURST can be generated by writing the following bit pattern into 82370 register location 64H.

D7	. . .	D0
1	1 1 1 X X X	0

The Write operation into this port is considered as an 82370 access and the internal Wait State Generator will automatically determine the required number of wait states. The CPURST will be active following the completion of the Write cycle to this port. This signal will last for 62 CLK2 periods. The 82370 should not be accessed until the CPURST is deactivated.

This internal port is Write-Only and the 82370 will not respond to a Read operation to this location. Also, during a software reset command, the 82370 will reset its Slave-Bus state machine. However, its internal registers remain unchanged. This allows the operating system to distinguish a 'warm' reset by reading any 82370 internal register previously programmed for a non-default value. The Diagnostic registers can be used for this purpose (see Internal Control and Diagnostic Ports).

9.3 Shutdown Detect

The 82370 is constantly monitoring the Bus Cycle Definition signals (M/IO#, D/C#, W/R#) and is able to detect when the 80376 is in a Shutdown bus cycle. Upon detection of a processor shutdown, the 82370 will activate the CPURST output for 62 CLK2 periods to reset the host processor. This signal is generated after the Shutdown cycle is terminated by the READY# signal.

Although the 82370 Wait State Generator will not automatically respond to a Shutdown (or Halt) cycle, the Wait State Control inputs (WSC0, WSC1) can be used to determine the number of wait states in the same manner as other non-82370 bus cycles.

This Shutdown Detect feature can be enabled or disabled by writing a control bit in the Internal Control Port at address 61H (see Internal Control and Diagnostic Ports). This feature is disabled upon a hardware reset of the 82370. As in the case of Software Reset, the 82370 will reset its Slave-Bus state machine but will not change any of its internal register contents.

10.0 INTERNAL CONTROL AND DIAGNOSTIC PORTS

10.1 Internal Control Port

The format of the Internal Control Port of the 82370 is shown in Figure 10-1. This Control Port is used to enable/disable the Processor Shutdown Detect mechanism as well as controlling the Gate inputs of the Timer 2 and 3. Note that this is a Write-Only port. Therefore, the 82370 will not respond to a read operation to this port. Upon hardware reset, this port will be cleared; i.e., the Shutdown Detect feature and the Gate inputs of Timer 2 and 3 are disabled.

Port Address: 61H (Write only)

10.2 Diagnostic Ports

Two 8-bit read/write Diagnostic Ports are provided in the 82370. These are two storage registers and have no effect on the operation of the 82370. They can be used to store checkpoint data or error codes in the power-on sequence and in the diagnostic service routines. As mentioned in the CPU RESET AND SHUTDOWN DETECT section, these Diagnostic Ports can be used to distinguish between 'cold' and 'warm' reset. Upon hardware reset, both Diagnostic Ports are cleared. The address map of these Diagnostic Ports is shown in Figure 10-2.

Port		Address
Diagnostic Port 1	(Read/Write)	80H
Diagnostic Port 2	(Read/Write)	88H

Figure 10-2. Address Map of Diagnostic Ports

11.0 INTEL RESERVED I/O PORTS

There are nineteen I/O ports in the 82370 address space which are reserved for Intel future peripheral device use only. Their address locations are: 10H, 12H, 14H, 16H, 2AH, 3DH, 3EH, 45H, 46H, 76H, 77H, 7DH, 7EH, CCH, CDH, D0H, D2H, D4H, and D6H. These addresses should not be used in the system since the 82370 will respond to read/write operations to these locations and bus contention may occur if any peripheral is assigned to the same address location.

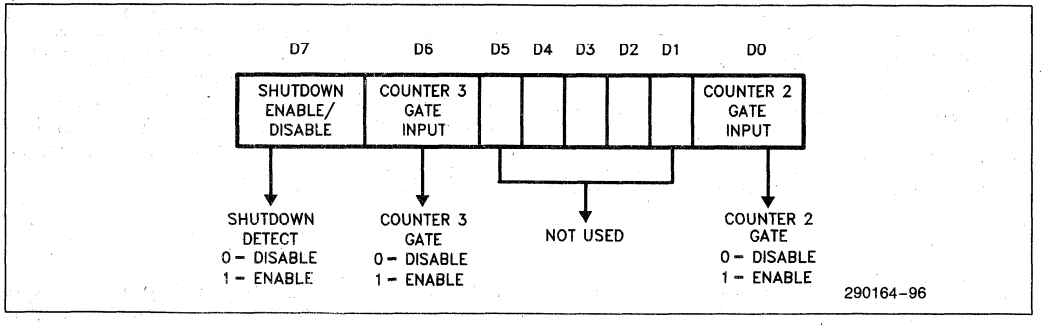


Figure 10-1. Internal Control Port

12.0 PACKAGE THERMAL SPECIFICATIONS

The intel 82370 Integrated System Peripheral is specified for operation when case temperature is within the range of 0°C to 78°C for the ceramic 132-pin PGA package, and 68°C for the 100-pin plastic package. The case temperature may be measured in any environment, to determine whether the 82370 is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

The ambient temperature is guaranteed as long as T_C is not violated. The ambient temperature can be calculated from the θ_{jc} and θ_{ja} from the following equations:

$T_J = T_C + P^*\theta_{jc}$

$T_A = T_j - P^*\theta_{ja}$

$T_C = T_a + P^*[\theta_{ja} - \theta_{jc}]$

Values for θ_{ja} and θ_{jc} are given in Table 12.1 for the 100-lead fine pitch. θ_{ja} is given at various airflows. Table 12.2 shows the maximum T_a allowable (without exceeding T_C) at various airflows. Note that T_a can be improved further by attaching "fins" or a "heat sink" to the package. P is calculated using the maximum ***hot*** I_{CC}.

Table 12.1 82370 Package Thermal Characteristics
Thermal Resistances (°C/Watt) θ_{jc} and θ_{ja}

Package	θ_{jc}	θ_{ja} Versus Airflow-ft³/min (m³/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
100L Fine Pitch	7	33	27	24	21	18	17
132L PGA	2	21	17	14	12	11	10

Table 12.2 82370 Maximum Allowable Ambient Temperature at Various Airflows

Package	θ_{jc}	T_a(c) Versus Airflow-ft³/min (m³/sec)					
		0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
100L Fine Pitch	7	63	74	79	85	91	92
132L PGA	2	74	83	88	93	97	99

100L PQFP Pkg:
$T_c = T_a + P^*(\theta_{ja} - \theta_{jc})$
$T_c = 63 + 1.21(33 - 7)$
$T_c = 63 + 1.21(26)$
$T_c = 63 + 31.46$
$T_c = 94\,°C$

132L PGA Pkg:
$T_c = T_a + P^*(\theta_{ja} - \theta_{jc})$
$T_c = 74 + 1.21(21 - 2)$
$T_c = 74 + 1.21(19)$
$T_c = 74 + 22.99$
$T_c = 96\,°C$

5

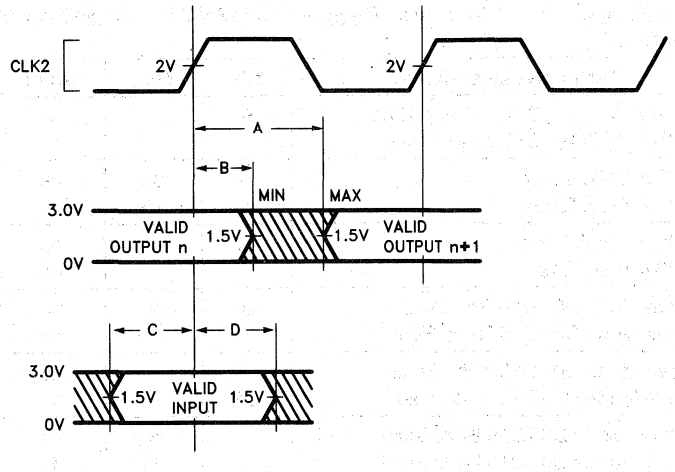
13.0 ELECTRICAL SPECIFICATIONS

82370 D.C. Specifications Functional Operating Range:

V_{CC} = 5.0V ±10%; T_{CASE} = 0°C to 96°C for 132-pin PGA, 0°C to 94°C for 100-pin plastic

Symbol	Parameter Description	Min	Max	Units	Notes
V_{IL}	Input Low Voltage	−0.3	0.8	V	(Note 1)
V_{IH}	Input High Voltage	2.0	V_{CC} + 0.3	V	
V_{ILC}	CLK2 Input Low Voltage	−0.3	0.8	V	(Note 1)
V_{IHC}	CLK2 Input High Voltage	V_{CC} − 0.8	V_{CC} + 0.3	V	
V_{OL}	Output Low Voltage I_{OL} = 4 mA: A_{1-23}, D_{0-15}, BHE#, BLE# I_{OL} = 5 mA: All Others		 0.45 0.45	 V V	
V_{OH}	Output High Voltage				
I_{OH} = −1 mA	A_{23}–A_1, D_{15}–D_0, BHE#, BLE#	2.4		V	(Note 5)
I_{OH} = −0.2 mA	A_{23}–A_1, D_{15}–D_0, BHE#, BLE#	V_{CC} − 0.5		V	(Note 5)
I_{OH} = −0.9 mA	All Others	2.4		V	(Note 5)
I_{OH} = −0.18 mA	All Others	V_{CC} − 0.5		V	(Note 5)
I_{LI}	Input Leakage Current All Inputs Except: IRQ11#–IRQ23# EOP#, TOUT2/IRQ3# DREQ4/IRQ9#		±15	μA	
I_{LI1}	Input Leakage Current Inputs: IRQ11#–IRQ23# EOP#, TOUT2/IRQ3 DREQ4/IRQ9	10	−300	μA	$0 < V_{IN} < V_{CC}$ (Note 3)
I_{LO}	Output Leakage Current		±15	μA	$0 < V_{IN} < V_{CC}$
I_{CC}	Supply Current (CLK2 = 32 MHz)		220	mA	(Note 4)
C_I	Input Capacitance		12	pF	(Note 2)
C_{CLK}	CLK2 Input Capacitance		20	pF	(Note 2)

NOTES:
1. Minimum value is not 100% tested.
2. f_C = 1 MHz; sampled only.
3. These pins have weak internal pullups. They sould not be left floating.
4. I_{CC} is specified with inputs driven to CMOS levels, and outputs driving CMOS loads. I_{CC} may be higher if inputs are driven to TTL levels, or if outputs are driving TTL loads.
5. Tested at the minimum operating frequency of the part.



LEGEND:
A—Maximum output delay specification
B—Minimum output delay specification
C—Minimum input setup specification
D—Minimum input hold specification

Figure 13-1. Drive Levels and Measurement Points for A.C. Specification

82370 A.C. Specifications These A.C. timings are tested at 1.5V thresholds, except as noted. Functional Operating Range: V_{CC} = 5.0V ±10%; T_{CASE} = 0°C to 96°C for 132-pin PGA, 0°C to 94°C for 100-pin plastic

Symbol	Parameter Description	Min	Max	Units	Notes
	Operating Frequency 1/(t1a × 2)	4	16	MHz	
t1	CLK2 Period	31	125	ns	
t2a	CLK2 High Time	9		ns	At 2.0V
t2b	CLK2 High Time	5		ns	At V_{CC} − 0.8V
t3a	CLK2 Low Time	9		ns	At 2.0V
t3b	CLK2 Low Time	7		ns	At 0.8V
t4	CLK2 Fall Time		7	ns	V_{CC} − 0.8V to 0.8V
t5	CLK2 Rise Time		7	ns	0.8V to V_{CC} − 0.8V
t6	A1–A23, BHE#, BLE# EDACK0–EDACK2 Valid Delay	4	36	ns	C_L = 120 pF
t7	A1–A23, BHE#, BLE# EDACK0–EDACK3 Float Delay	4	40	ns	(Note 1)
t8	A1–A23, BHE#, BLE# Setup Time	6		ns	
t9	A1–A23, BHE#, BLE# Hold Time	4		ns	
t10	W/R#, M/IO#, D/C# Valid Delay	4	33	ns	C_L = 75 pF
t11	W/R#, M/IO#, D/C# Float Delay	4	35	ns	(Note 1)

82370 A.C. Specifications These A.C. timings are tested at 1.5V thresholds, except as noted. Functional Operating Range: V_{CC} = 5.0V ±10%; T_{CASE} = 0°C to 96°C for 132-pin PGA, 0°C to 94°C for 100-pin plastic (Continued)

Symbol	Parameter Description	Min	Max	Units	Notes
t12	W/R#, M/IO#, D/C# Setup Time	6		ns	
t13	W/R#, M/IO#, D/C# Hold Time	4		ns	
t14	ADS# Valid Delay	6	33	ns	CL = 50 pF
t15	ADS# Float Delay	4	35	ns	(Note 1)
t16	ADS# Setup Time	21		ns	
t17	ADS# Hold Time	4		ns	
t18	Slave Mode D0-D15 Read Valid	3	46	ns	C_L = 120 pF
t19	Slave Mode D0-D15 Read Float	6	35	ns	(Note 1)
t20	Slave Mode D0-D15 Write Setup	31		ns	
t21	Slave Mode D0-D15 Write Hold	26		ns	
t22	Master Mode D0-D15 Write Valid	4	40	ns	C_L = 120 pF
t23	Master Mode D0-D15 Write Float	4	35	ns	(Note 1)
t24	Master Mode D0-D15 Read Setup	8		ns	
t25	Master Mode D0-D15 Read Hold	6		ns	
t26	READY# Setup Time	19		ns	
t27	READY# Hold Time	4		ns	
t28	WSC0-WSC1 Setup Time	6		ns	
t29	WSC0-WSC1 Hold Time	21		ns	
t30	RESET Setup Time	13		ns	
t31	RESET Hold Time	4		ns	
t32	READYO# Valid Delay	4	31	ns	C_L = 25 pF
t33	CPURST Valid Delay (Falling Edge Only)	2	18	ns	C_L = 50 pF
t34	HOLD Valid Delay	5	33	ns	C_L = 100 pF
t35	HLDA Setup Time	21		ns	
t36	HLDA Hold Time	6		ns	
t37a	EOP# Setup (Synchronous)	21		ns	
t38a	EOP# Hold (Synchronous)	6		ns	
t37b	EOP# Setup (Asynchronous)	11		ns	
t38b	EOP# Hold (Asynchronous)	11		ns	
t39	EOP# Valid Delay (Falling Edge Only)	5	38	ns	C_L = 100 pF
t40	EOP# Float Delay	5	40	ns	(Note 1)
t41a	DREQ Setup (Synchronous)	21		ns	
t42a	DREQ Hold (Synchronous)	4		ns	
t41b	DREQ Setup (Asynchronous)	11		ns	
t42b	DREQ Hold (Asynchronous)	11		ns	
t43	INT Valid Delay from IRQn		500	ns	
t44	NA# Setup Time	5		ns	
t45	NA# Hold Time	15		ns	

82370 A.C. Specifications These A.C. timings are tested at 1.5V thresholds, except as noted. Functional Operating Range: V_{CC} = 5.0V ±10%; T_{CASE} = 0°C to 96°C for 132-pin PGA, 0°C to 94°C for 100-pin plastic (Continued)

Symbol	Parameter Description	Min	Max	Units	Notes
t46	CLKIN Frequency	DC	10	MHz	
t47	CLKIN High Time	30		ns	2.0V
t48	CLKIN Low Time	50		ns	0.8V
t49	CLKIN Rise Time		10	ns	0.8V to 3.7V
t50	CLKIN Fall Time		10	ns	3.7V to 0.8V
	TOUT1#/REF# Valid Delay				
t51	from CLK2 (Refresh)	4	36	ns	C_L = 120 pF
t52	from CLKIN (Timer)	3	93	ns	C_L = 120 pF
t53	TOUT2# Valid Delay (from CLKIN, Falling Edge Only)	3	93	ns	C_L = 120 pf
t54	TOUT2# Float Delay	3	36	ns	(Note 1)
t55	TOUT3# Valid Delay (from CLKIN)	3	93	ns	C_L = 120 pF
t56	CHPSEL# Valid Delay	1	35	ns	C_L = 25 pF

NOTE:
1. Float condition occurs when the maximum output current becomes less than I_{LO} in magnitude. Float delay is not tested. For testing purposes, the float condition occurs when the dynamic output driven voltage changes with current loads.

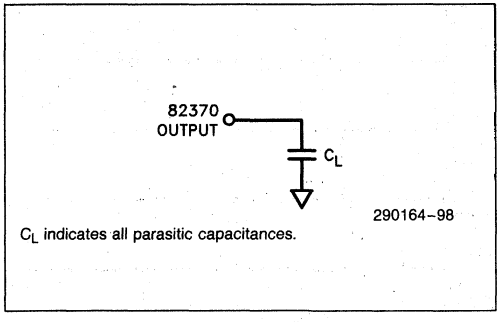


Figure 13-2. A.C. Test Load

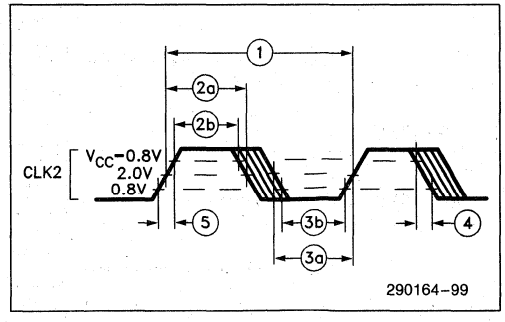


Figure 13-3

5

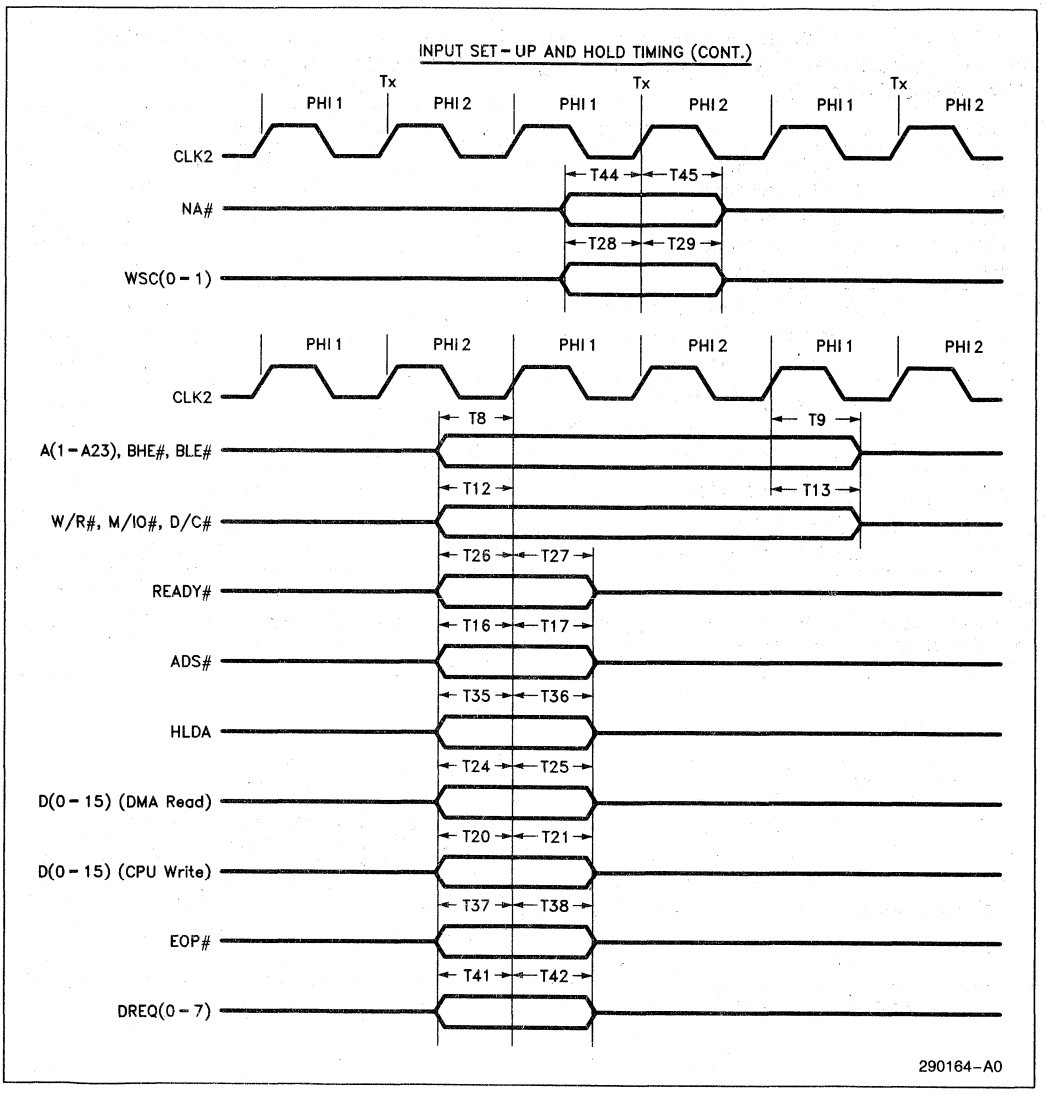


Figure 13-4. Input Setup and Hold Timing

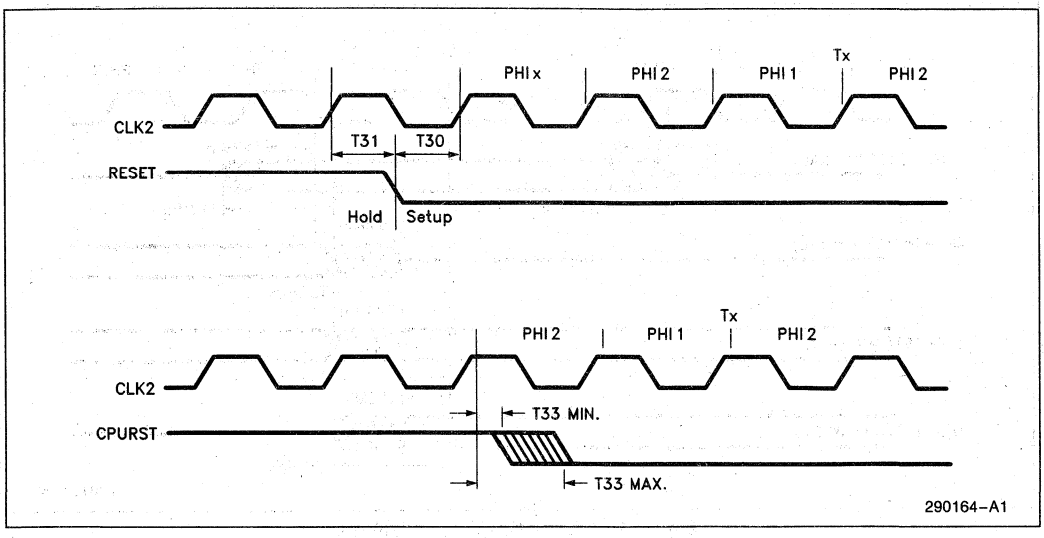


Figure 13-5. Reset Timing

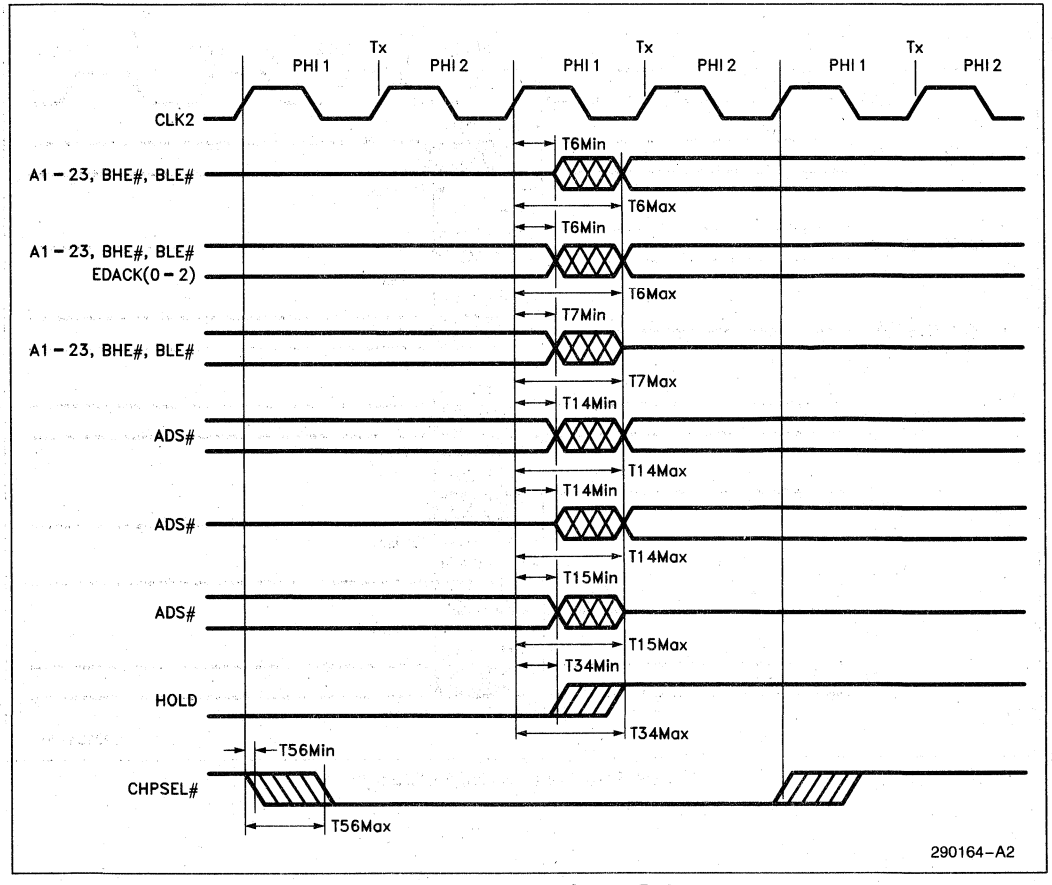


Figure 13-6. Address Output Delays

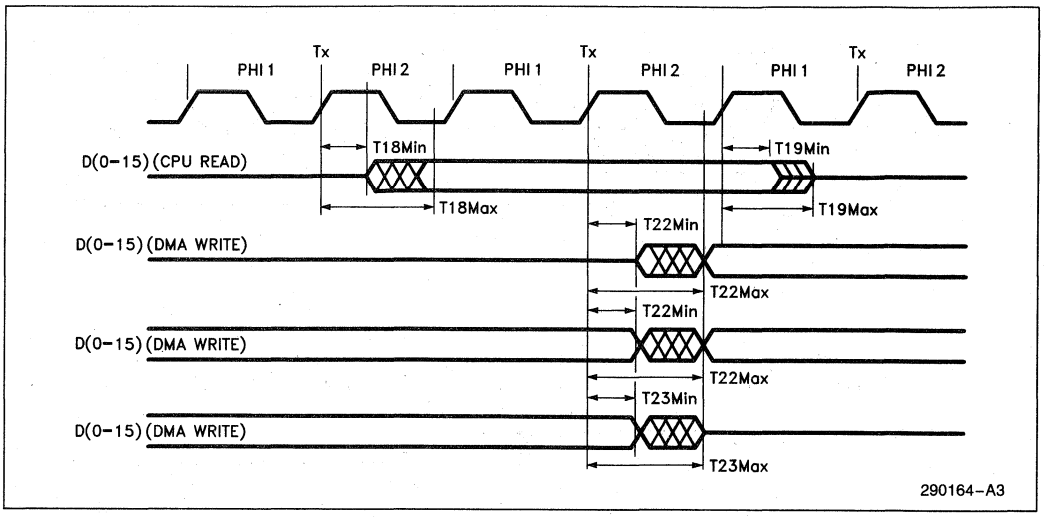


Figure 13-7. Data Bus Output Delays

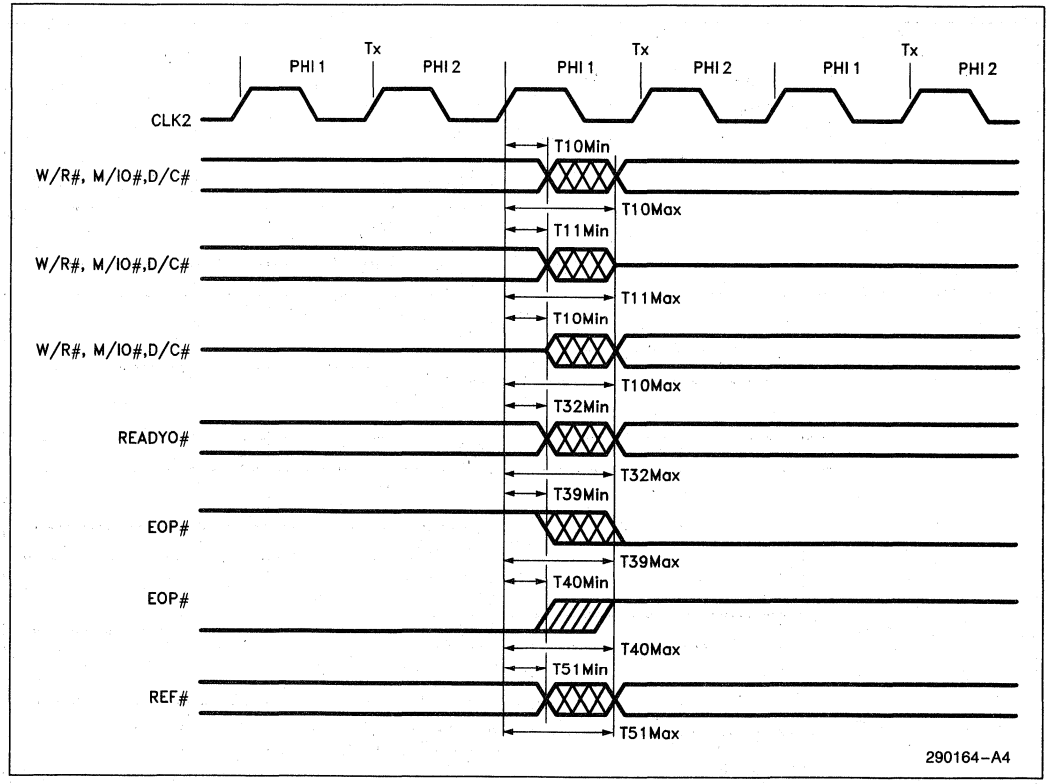


Figure 13-8. Control Output Delays

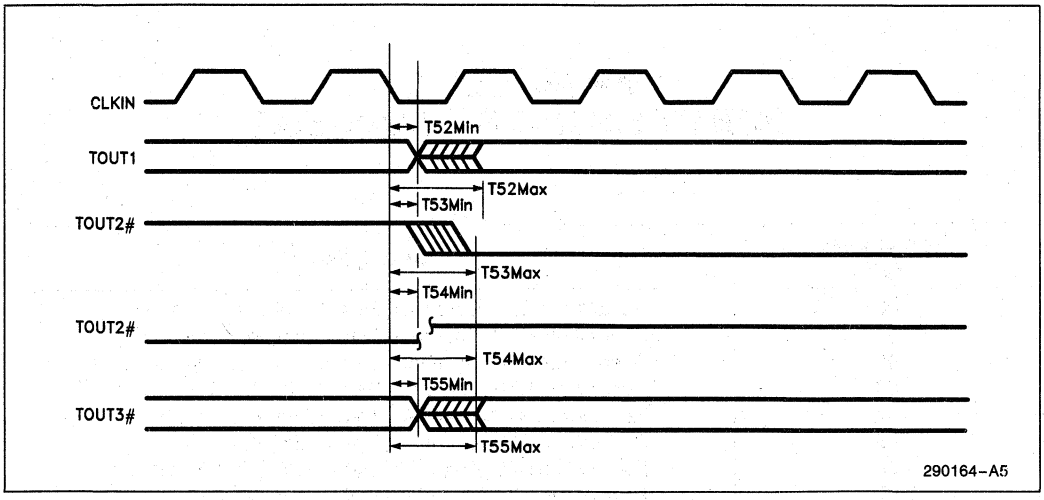


Figure 13-9. Timer Output Delays

APPENDIX A
PORTS LISTED BY ADDRESS

Port Address (HEX)	Description
00	Read/Write DMA Channel 0 Target Address, A0–A15
01	Read/Write DMA Channel 0 Byte Count, B0–B15
02	Read/Write DMA Channel 1 Target Address, A0–A15
03	Read/Write DMA Channel 1 Byte Count, B0–B15
04	Read/Write DMA Channel 2 Target Address, A0–A15
05	Read/Write DMA Channel 2 Byte Count, B0–B15
06	Read/Write DMA Channel 3 Target Address, A0–A15
07	Read/Write DMA Channel 3 Byte Count, B0–B15
08	Read/Write DMA Channel 0–3 Status/Command I Register
09	Read/Write DMA Channel 0–3 Software Request Register
0A	Write DMA Channel 0–3 Set-Reset Mask Register
0B	Write DMA Channel 0–3 Mode Register I
0C	Write Clear Byte-Pointer FF
0D	Write DMA Master-Clear
0E	Write DMA Channel 0–3 Clear Mask Register
0F	Read/Write DMA Channel 0–3 Mask Register
10	Intel Reserved
11	Read/Write DMA Channel 0 Byte Count, B16–B23
12	Intel Reserved
13	Read/Write DMA Channel 1 Byte Count, B16–B23
14	Intel Reserved
15	Read/Write DMA Channel 2 Byte Count, B16–B23
16	Intel Reserved
17	Read/Write DMA Channel 3 Byte Count, B16–B23
18	Write DMA Channel 0–3 Bus Size Register
19	Read/Write DMA Channel 0–3 Chaining Register
1A	Write DMA Channel 0–3 Command Register II
1B	Write DMA Channel 0–3 Mode Register II
1C	Read/Write Refresh Control Register
1E	Reset Software Request Interrupt
20	Write Bank B ICW1, OCW2 or OCW3 Read Bank B Poll, Interrupt Request or In-Service Status Register
21	Write Bank B ICW2, ICW3, ICW4 or OCW1 Read Bank B Interrupt Mask Register
22	Read Bank B ICW2
28	Read/Write IRQ8 Vector Register
29	Read/Write IRQ9 Vector Register
2A	Reserved

Port Address (HEX)	Description
2B	Read/Write IRQ11 Vector Register
2C	Read/Write IRQ12 Vector Register
2D	Read/Write IRQ13 Vector Register
2E	Read/Write IRQ14 Vector Register
2F	Read/Write IRQ15 Vector Register
30	Write Bank A ICW1, OCW2 or OCW3 Read Bank A Poll, Interrupt Request or In-Service Status Register
31	Write Bank A ICW2, ICW3, ICW4 or OCW1 Read Bank A Interrupt Mask Register
32	Read Bank A ICW2
38	Read/Write IRQ0 Vector Register
39	Read/Write IRQ1 Vector Register
3A	Read/Write IRQ1.5 Vector Register
3B	Read/Write IRQ3 Vector Register
3C	Read/Write IRQ4 Vector Register
3D	Reserved
3E	Reserved
3F	Read/Write IRQ7 Vector Register
40	Read/Write Counter 0 Register
41	Read/Write Counter 1 Register
42	Read/Write Counter 2 Register
43	Write Control Word Register I—Counter 0, 1, 2
44	Read/Write Counter 3 Register
45	Reserved
46	Reserved
47	Write Word Register II—Counter 3
61	Write Internal Control Port
64	Write CPU Reset Register (Data—1111XXX0H)
72	Read/Write Wait State Register 0
73	Read/Write Wait State Register 1
74	Read/Write Wait State Register 2
75	Read/Write Refresh Wait State Register
76	Reserved
77	Reserved
7D	Reserved
7E	Reserved
7F	Read/Write Relocation Register
80	Read/Write Internal Diagnostic Port 0
81	Read/Write DMA Channel 2 Target Address, A16–A23
82	Read/Write DMA Channel 3 Target Address, A16–A23
83	Read/Write DMA Channel 1 Target Address, A16–A23
87	Read/Write DMA Channel 0 Target Address, A16–A23
88	Read/Write Internal Diagnostic Port 1
89	Read/Write DMA Channel 6 Target Address, A16–A23
8A	Read/Write DMA Channel 7 Target Address, A16–A23
8B	Read/Write DMA Channel 5 Target Address, A16–A23
8F	Read/Write DMA Channel 4 Target Address, A16–A23

Port Address (HEX)	Description
90	Read/Write DMA Channel 0 Requester Address, A0–A15
91	Read/Write DMA Channel 0 Requester Address, A16–A23
92	Read/Write DMA Channel 1 Requester Address, A0–A15
93	Read/Write DMA Channel 1 Requester Address, A16–A23
94	Read/Write DMA Channel 2 Requester Address, A0–A15
95	Read/Write DMA Channel 2 Requester Address, A16–A23
96	Read/Write DMA Channel 3 Requester Address, A0–A15
97	Read/Write DMA Channel 3 Requester Address, A16–A23
98	Read/Write DMA Channel 4 Requester Address, A0–A15
99	Read/Write DMA Channel 4 Requester Address, A16–A23
9A	Read/Write DMA Channel 5 Requester Address, A0–A15
9B	Read/Write DMA Channel 5 Requester Address, A16–A23
9C	Read/Write DMA Channel 6 Requester Address, A0–A15
9D	Read/Write DMA Channel 6 Requester Address, A16–A23
9E	Read/Write DMA Channel 7 Requester Address, A0–A15
9F	Read/Write DMA Channel 7 Requester Address, A16–A23
A0	Write Bank C ICW1, OCW2 or OCW3 Read Bank C Poll, Interrupt Request or In-Service Status Register
A1	Write Bank C ICW2, ICW3, ICW4 or OCW1 Read Bank C Interrupt Mask Register
A2	Read Bank C ICW2
A8	Read/Write IRQ16 Vector Register
A9	Read/Write IRQ17 Vector Register
AA	Read/Write IRQ18 Vector Register
AB	Read/Write IRQ19 Vector Register
AC	Read/Write IRQ20 Vector Register
AD	Read/Write IRQ21 Vector Register
AE	Read/Write IRQ22 Vector Register
AF	Read/Write IRQ23 Vector Register
C0	Read/Write DMA Channel 4 Target Address, A0–A15
C1	Read/Write DMA Channel 4 Byte Count, B0–B15
C2	Read/Write DMA Channel 5 Target Address, A0–A15
C3	Read/Write DMA Channel 5 Byte Count, B0–B15
C4	Read/Write DMA Channel 6 Target Address, A0–A15
C5	Read/Write DMA Channel 6 Byte Count, B0–B15
C6	Read/Write DMA Channel 7 Target Address, A0–A15
C7	Read/Write DMA Channel 7 Byte Count, B0–B15
C8	Read DMA Channel 4–7 Status/Command I Register
C9	Read/Write DMA Channel 4–7 Software Request Register
CA	Write DMA Channel 4–7 Set-Reset Mask Register
CB	Write DMA Channel 4–7 Mode Register I
CC	Reserved
CD	Reserved
CE	Write DMA Channel 4–7 Clear Mask Register
CF	Read/Write DMA Channel 4–7 Mask Register
D0	Intel Reserved
D1	Read/Write DMA Channel 4 Byte Count, B16–B23
D2	Intel Reserved
D3	Read/Write DMA Channel 5 Byte Count, B16–B23

Port Address (HEX)	Description
D4	Intel Reserved
D5	Read/Write DMA Channel 6 Byte Count, B16–B23
D6	Intel Reserved
D7	Read/Write DMA Channel 7 Byte Count, B16–B23
D8	Write DMA Channel 4–7 Bus Size Register
D9	Read/Write DMA Channel 4–7 Chaining Register
DA	Write DMA Channel 4–7 Command Register II
DB	Write DMA Channel 4–7 Mode Register II

APPENDIX B
PORTS LISTED BY FUNCTION

Port Address (HEX)	Description
DMA CONTROLLER	
0D	Write DMA Master-Clear
0C	Write DMA Clear Byte-Pointer FF
08	Read/Write DMA Channel 0–3 Status/Command I Register
C8	Read/Write DMA Channel 4–7 Status/Command I Register
1A	Write DMA Channel 0–3 Command Register II
DA	Write DMA Channel 4–7 Command Register II
0B	Write DMA Channel 0–3 Mode Register I
CB	Write DMA Channel 4–7 Mode Register I
1B	Write DMA Channel 0–3 Mode Register II
DB	Write DMA Channel 4–7 Mode Register II
09	Read/Write DMA Channel 0–3 Software Request Register
C9	Read/Write DMA Channel 4–7 Software Request Register
1E	Reset Software Request Interrupt
0E	Write DMA Channel 0–3 Clear Mask Register
CE	Write DMA Channel 4–7 Clear Mask Register
0F	Read/Write DMA Channel 0–3 Mask Register
CF	Read/Write DMA Channel 4–7 Mask Register
0A	Write DMA Channel 0–3 Set-Reset Mask Register
CA	Write DMA Channel 4–7 Set-Reset Mask Register
18	Write DMA Channel 0–3 Bus Size Register
D8	Write DMA Channel 4–7 Bus Size Register
19	Read/Write DMA Channel 0–3 Chaining Register
D9	Read/Write DMA Channel 4–7 Chaining Register
00	Read/Write DMA Channel 0 Target Address, A0–A15
87	Read/Write DMA Channel 0 Target Address, A16–A23
01	Read/Write DMA Channel 0 Byte Count, B0–B15
11	Read/Write DMA Channel 0 Byte Count, B16–B23
90	Read/Write DMA Channel 0 Requester Address, A0–A15
91	Read/Write DMA Channel 0 Requester Address, A16–A23

Port Address (HEX)	Description
DMA CONTROLLER (Continued)	
02	Read/Write DMA Channel 1 Target Address, A0–A15
83	Read/Write DMA Channel 1 Target Address, A16–A23
03	Read/Write DMA Channel 1 Byte Count, B0–B15
13	Read/Write DMA Channel 1 Byte Count, B16–B23
92	Read/Write DMA Channel 1 Requester Address, A0–A15
93	Read/Write DMA Channel 1 Requester Address, A16–A23
04	Read/Write DMA Channel 2 Target Address, A0–A15
81	Read/Write DMA Channel 2 Target Address, A16–A23
05	Read/Write DMA Channel 2 Byte Count, B0–B15
15	Read/Write DMA Channel 2 Byte Count, B16–B23
94	Read/Write DMA Channel 2 Requester Address, A0–A15
95	Read/Write DMA Channel 2 Requester Address, A16–A23
06	Read/Write DMA Channel 3 Target Address, A0–A15
82	Read/Write DMA Channel 3 Target Address, A16–A23
07	Read/Write DMA Channel 3 Byte Count, B0–B15
17	Read/Write DMA Channel 3 Byte Count, B16–B23
96	Read/Write DMA Channel 3 Requester Address, A0–A15
97	Read/Write DMA Channel 3 Requester Address, A16–A23
C0	Read/Write DMA Channel 4 Target Address, A0–A15
8F	Read/Write DMA Channel 4 Target Address, A16–A23
C1	Read/Write DMA Channel 4 Byte Count, B0–B15
D1	Read/Write DMA Channel 4 Byte Count, B16–B23
98	Read/Write DMA Channel 4 Requester Address, A0–A15
99	Read/Write DMA Channel 4 Requester Address, A16–A23
C2	Read/Write DMA Channel 5 Target Address, A0–A15
8B	Read/Write DMA Channel 5 Target Address, A16–A23
C3	Read/Write DMA Channel 5 Byte Count, B0–B15
D3	Read/Write DMA Channel 5 Byte Count, B16–B23
9A	Read/Write DMA Channel 5 Requester Address, A0–A15
9B	Read/Write DMA Channel 5 Requester Address, A16–A23
C4	Read/Write DMA Channel 6 Target Address, A0–A15
89	Read/Write DMA Channel 6 Target Address, A16–A23
C5	Read/Write DMA Channel 6 Byte Count, B0–B15
D5	Read/Write DMA Channel 6 Byte Count, B16–B23
9C	Read/Write DMA Channel 6 Requester Address, A0–A15
9D	Read/Write DMA Channel 6 Requester Address, A16–A23
C6	Read/Write DMA Channel 7 Target Address, A0–A15
8A	Read/Write DMA Channel 7 Target Address, A16–A23
C7	Read/Write DMA Channel 7 Byte Count, B0–B15
D7	Read/Write DMA Channel 7 Byte Count, B16–B23
9E	Read/Write DMA Channel 7 Requester Address, A0–A15
9F	Read/Write DMA Channel 7 Requester Address, A16–A23

Port Address (HEX)	Description
INTERRUPT CONTROLLER	
20	Write Bank B ICW1, OCW2 or OCW3 Read Bank B Poll, Interrupt Request or In-Service Status Register
21	Write Bank B ICW2, ICW3, ICW4 or OCW1 Read Bank B Interrupt Mask Register
22	Read Bank B ICW2
28	Read/Write IRQ8 Vector Register
29	Read/Write IRQ9 Vector Register
2A	Reserved
2B	Read/Write IRQ11 Vector Register
2C	Read/Write IRQ12 Vector Register
2D	Read/Write IRQ13 Vector Register
2E	Read/Write IRQ14 Vector Register
2F	Read/Write IRQ15 Vector Register
A0	Write Bank C ICW1, OCW2 or OCW3 Read Bank C Poll, Interrupt Request or In-Service Status Register
A1	Write Bank C ICW2, ICW3, ICW4 or OCW1 Read Bank C Interrupt Mask Register
A2	Read Bank C ICW2
A8	Read/Write IRQ16 Vector Register
A9	Read/Write IRQ17 Vector Register
AA	Read/Write IRQ18 Vector Register
AB	Read/Write IRQ19 Vector Register
AC	Read/Write IRQ20 Vector Register
AD	Read/Write IRQ21 Vector Register
AE	Read/Write IRQ22 Vector Register
AF	Read/Write IRQ23 Vector Register
30	Write Bank A ICW1, OCW2 or OCW3 Read Bank A Poll, Interrupt Request or In-Service Status Register
31	Write Bank A ICW2, ICW3, ICW4 or OCW1 Read Bank A Interrupt Mask Register
32	Read Bank A ICW2
38	Read/Write IRQ0 Vector Register
39	Read/Write IRQ1 Vector Register
3A	Read/Write IRQ1.5 Vector Register
3B	Read/Write IRQ3 Vector Register
3C	Read/Write IRQ4 Vector Register
3D	Reserved
3E	Reserved
3F	Read/Write IRQ7 Vector Register

Port Address (HEX)	Description
PROGRAMMABLE INTERVAL TIMER	
40	Read/Write Counter 0 Register
41	Read/Write Counter 1 Register
42	Read/Write Counter 2 Register
43	Write Control Word Register I—Counter 0, 1, 2
44	Read/Write Counter 3 Register
47	Write Word Register II—Counter 3
CPU RESET	
64	Write CPU Reset Register (Data—1111XXX0H)
WAIT STATE GENERATOR	
72	Read/Write Wait State Register 0
73	Read/Write Wait State Register 1
74	Read/Write Wait State Register 2
75	Read/Write Refresh Wait State Register
DRAM REFRESH CONTROLLER	
1C	Read/Write Refresh Control Register
INTERNAL CONTROL AND DIAGNOSTIC PORTS	
61	Write Internal Control Port
80	Read/Write Internal Diagnostic Port 0
88	Read/Write Internal Diagnostic Port 1
RELOCATION REGISTER	
7F	Read/Write Relocation Register
INTEL RESERVED PORTS	
10	Reserved
12	Reserved
14	Reserved
16	Reserved
2A	Reserved
3D	Reserved
3E	Reserved
45	Reserved
46	Reserved
76	Reserved
77	Reserved
7D	Reserved
7E	Reserved
CC	Reserved
CD	Reserved
D0	Reserved
D2	Reserved
D4	Reserved
D6	Reserved

APPENDIX C
SYSTEM NOTES

1. BHE# IN MASTER MODE.

In Master Mode, BHE# will be activated during DMA to/from 8-bit devices residing at even locations when the remaining byte count is greater than 1.

For example, if an 8-bit device is located at 00000000 Hex and the number of bytes to be transferred is > 1, the first address/BHE# combination will be 00000000/0. In some systems this will cause the bus controller to perform two 8-bit accesses, the first to 0000000 Hex and the second to 00000001 Hex. However, the 82370's DMA will only read/write one byte. This may or may not cause a problem in the system depending on what is located at 00000001 Hex.

Solution:

There are two solutions if BH# active is unacceptable. Of the two, number 2 is the cleanest and most recommended.

1. If there is an 8-bit device that uses DMA located at an even address, do not use that address + 1. The limitation of this solution is that the user must have complete control over what addresses will be used in the end system.

2. Do not allow the Bus Controller to split cycles for the DMA.

82370 TIMER UNIT NOTES

The 82370 DMA Controller with Integrated System Peripherals is functionally inconsistent with the data sheet. This document explains the behavior of the 82370 Timer Unit and outlines subsequent limitations of the timer unit. This document also provides recommended workarounds.

1.0 WRITE CYCLES TO THE 82370 TIMER UNIT:

This errata applies only to SLAVE WRITE cycles to the 82370 timer unit. During these cycles, the data being written into the 82370 timer unit may be corrupted if asynchronous CLKIN is not inhibited during a certain "window" of the write cycle.

1.1 Description

Please refer to Figure C-2.

During write cycles to the 82370 timer unit, the 82370 translates the 80376 interface signals such as #ADS, #W/R, #M/IO, and #D/C into several internal signals that control the operation of the internal sub-blocks (e.g. Timer Unit).

The 82370 timer uint is controlled by such internal signals. These internal signals are generated and sampled with respect to two separate clock signals: CLK2 (the system clock) and CLKIN (the 82370 timer unit clock).

Since the CLKIN and CLK2 clock signals are used internally to generate control signals for the interface to the timer unit, some timing parameters must be met in order for the interface logic to function properly.

Those timing parameters are met by inhibiting the CLKIN signal for a specific window during Write Cycles to the 82370 Timer Unit.

The CLKIN signal must be inhibited using external logic, as the GATE function of the 82370 timer unit is not guaranteed to totally inhibit CLKIN.

1.2 Consequences

This CLKIN inhibits circuitry guarantees proper write cycles to the 82370 timer unit.

Without this solution, write cycles to the 82370 timer unit could place corrupted data into the timer unit registers. This, in turn, could yield inaccurate results and improper timer operation.

The proposed solution would involve a hardware modification for existing systems.

1.3 Solution

A timing waveform (Figure C-3) shows the specific window during which CLKIN must be inhibited. Please note that CLKIN must only be inhibited during the window shown in Figure C-3. This window is defined by two AC timing parameters:

t_a = 9 ns

t_b = 28 ns

The proposed solution provides a certain amount of system "guardband" to make sure that this window is avoided.

PAL equations for a suggested workaround are also included. Please refer to the comments in the PAL codes for stated assumptions of this particular workaround. A state diagram (Figure C-4) is provided to help clarify how this PAL is designed.

Figure C-5 shows how this PAL would fit into a system workaround. In order to show the effect of this workaround on the CLKIN signal, Figure C-6 shows how CLKIN is inhibited. Note that you must still meet the CLKIN AC timing parameters (e.g. t_{47} (min), t_{48} (min)) in order for the timer unit to function properly.

Please note that this workaround has not been tested. It is provided as a suggested solution. Actual solutions will vary from system to system.

1.4 Long Term Plans

Intel has no plans to fix this behavior in the 82370 timer unit.

```
module Timer_82370_Fix
flag '-r2', '-q2', '-fl', '-t4', '-wl,3,6,5,4,16,7,12,17,18,15,14'
title '82370 Timer Unit CLKIN
      INHIBIT signal PAL Solution '
Timer_Unit_Fix device 'P16R6';


"This PAL inhibits the CLKIN signal (that comes from an oscillator)
"during Slave Writes to the 82370 Timer unit.
"
"ASSUMPTION:  This PAL assumes that an external system address
"             decoder provides a signal to indicate that an 82370
"             Timer Unit access is taking place. This input
"             signal is called TMR in this PAL. This PAL also
"             assumes that this TMR signal occurs during a
"             specific T-State. Please see Figure 2 of this
"             document to see when this signal is expected to
"             be active by this PAL.
"
"
"NOTE:        This PAL does not support pipelined 82370 SLAVE
"             cycles.
"
"(c) Intel Corporation 1989. This PAL is provided as a proposed
"method of solving a certain 82370 Timer Unit problem. This PAL
"has not been tested or validated. Please validate this solution
"for your system and application.
"
"Input Pins"


CLK2          pin      1; "System Clock
RESET         pin      2; "Microprocessor RESET signal
TMR           pin      3; "Input from Address Decoder, indicating
                          "an access to the timer unit of the
                          "82370.
!RDY          pin      4; "End of Cycle indicator
!ADS          pin      5; "Address and control strobe
CLK           pin      6; "PHI2 clock
W_R           pin      7; "Write/Read Signal"
ncl           pin      8; "No Connect 0"
nc3           pin      9; "No Connect 1"
GNDa          pin     10; "Tied to ground, documentation only
GNDb          pin     11; "Output enable, documentation only
CLKIN_IN      pin     12; "Input-CLKIN directly from oscillator

"Output Pins"
Q_0           pin     18; "Internal signal only, fed back to
                          "PAL logic"
CLKIN_OUT     pin     17; "CLKIN signal fed to 82370 Timer Unit
INHIBIT       pin     16; "CLKIN Inhibit signal
S0            pin     15; "Unused State Indicator Pin
S1            pin     14; "Unused State Indicator Pin

"Declarations"
```

```
Valid_ADS = ADS & CLK     ; "#ADS sampled in PHI1 of 80376 T-State
Valid_RDY = RDY & CLK     ; "#RDY sampled in PHI1 of 80376 T-State
Timer_Acc = TMR & CLK     ; "Timer Unit Access, as provided by
                            "external Address Decoder"

State_Diagram [INHIBIT, S1, S0]

state 000:      if RESET then 000
                else if Valid_ADS & W_R then 001
                else 000;

state 001:      if RESET then 000
                else if Timer_Acc then 010
                else if !Timer_Acc then 000
                else 001;

state 010:      if RESET then 000
                else if CLK then 110
                else 010;

state 110:      if RESET then 000
                else if CLK then 111
                else 110;

state 111:      if RESET then 000
                else if CLK then 011
                else 111;

state 011:      if RESET then 000
                else if Valid_RDY then 000
                else 011;

state 100:      if RESET then 000
                else 000;

state 101:      if RESET then 000
                else 000;

EQUATIONS
Q_0 := CLKIN_IN; "Latched incoming clock. This signal is used
                  "internally to feed into the MUX-ing logic"


CLKIN_OUT := (INHIBIT & CLKIN_OUT & !RESET)
           +( !INHIBIT & Q_0 & !RESET);

                        "Equation for CLKIN_OUT. This
                        "feeds directly to the 82370 Timer Unit."

END
```

5

82370 Timer Unit CLKIN
INHIBIT signal PAL Solution
Equations for Module Timer__82370__Fix

Device Timer__Unit__Fix

—Reduced Equations:

```
!INHIBIT := (!CLK & !INHIBIT # CLK & S0 # RESET # !S1);

!S1 := (RESET
  # INHIBIT & !S1
  # CLK & !INHIBIT & !~RDY & S0 & S1
  # !CLK & !S1
  # !S1 & !TMR
  # !S0 & !S1);


!S0 := (RESET
  # INHIBIT & !S1
  # CLK & !INHIBIT & !~RDY & S1
  #!INHIBIT & !S0 & S1
  # !CLK & !S0
  # !INHIBIT & !S0 & S1
  # S0 & !S1
  # !S1 & !W__R
  # ~ADS & !S1);

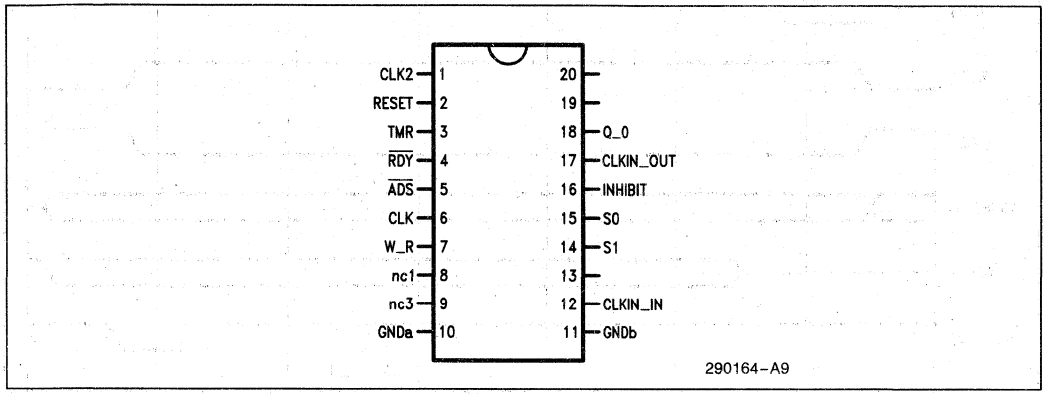

!Q__0 := (!CLKIN__IN);

!CLKIN__OUT := (RESET # !CLKIN__OUT & INHIBIT # !INHIBIT & !Q__0);
```

82370 Timer Unit CLKIN
INHIBIT signal PAL Solution
Chip diagram for Module Timer__82370__Fix

Device Timer__Unit__Fix

P16R6



end of module Timer__82370__Fix

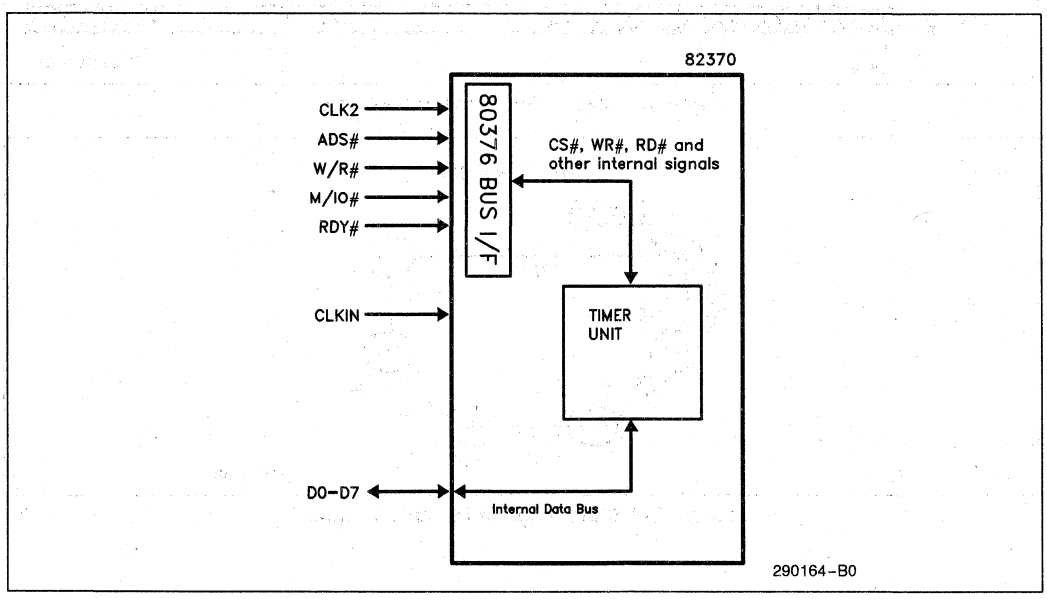


Figure C-2. Translation of 80376 Signals to Internal 82370 Timer Unit Signals

5

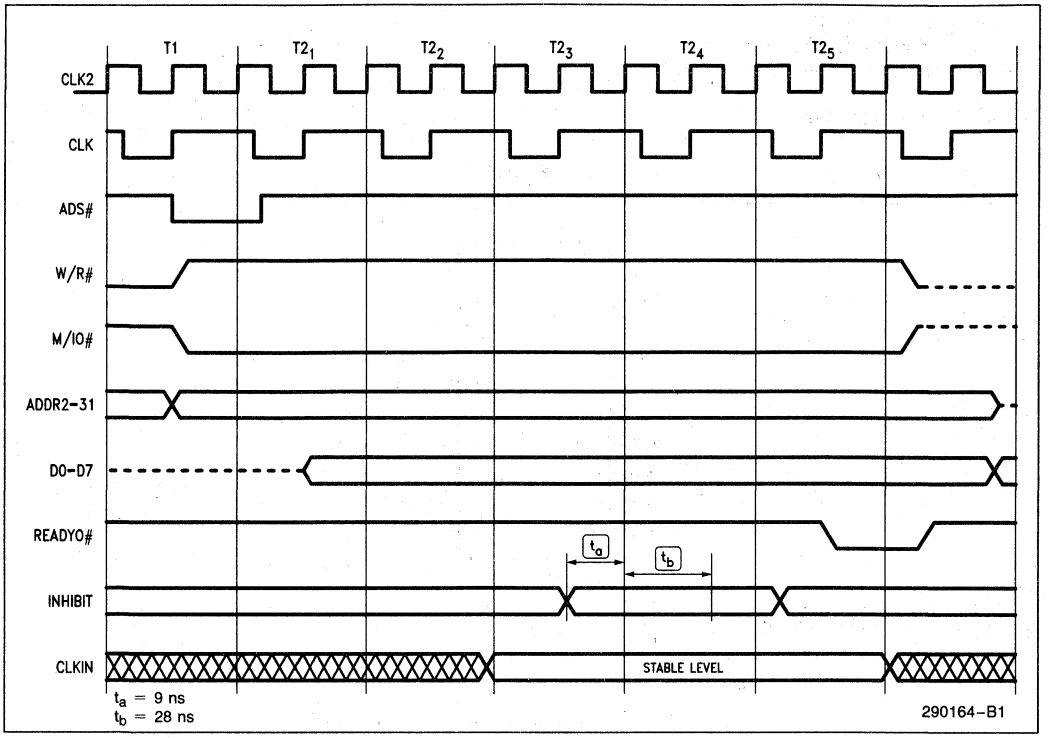


Figure C-3. 82370 Timer Unit Write Cycle

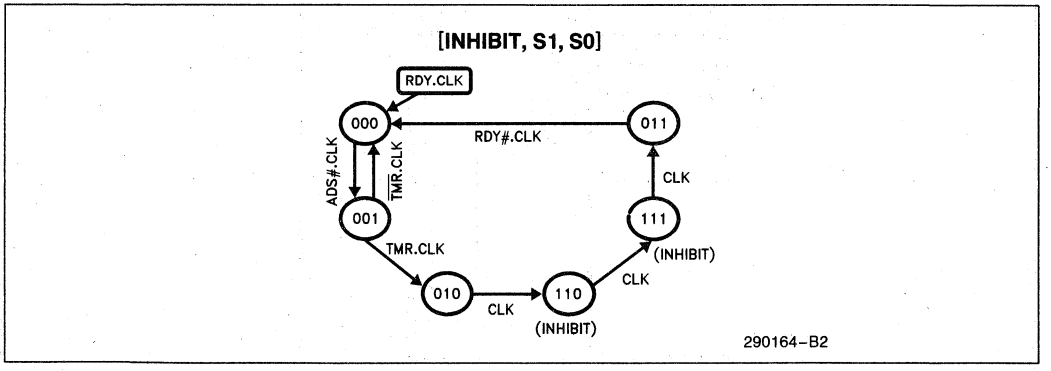
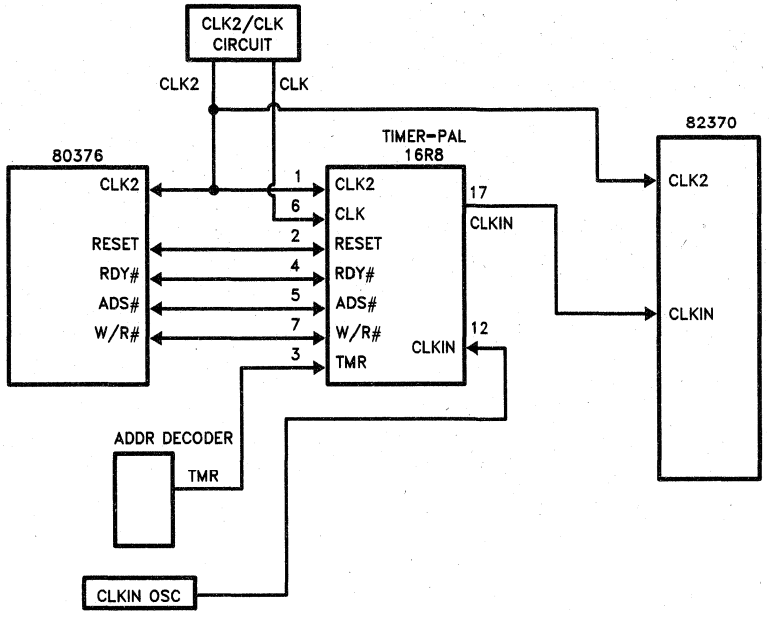


Figure C-4. State Diagram for Inhibit Signal



NOTE:
This solution does not support pipelined 82370 SLAVE Cycles.

Figure C-5. System with 82370 Timer Unit "INHIBIT" Circuitry

5

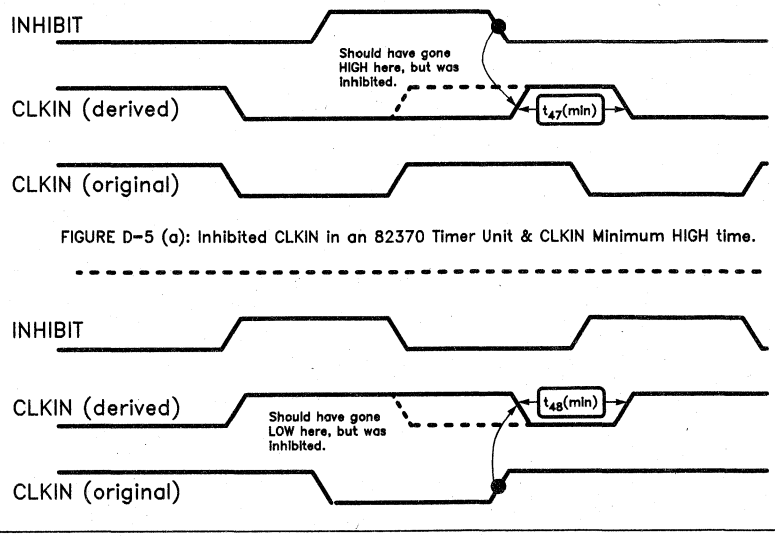


Figure C-6. Inhibited CLKIN in an 82370 Timer Unit and CLKIN Minimum LOW Time

i860™ Microprocessor Family 6

i860™ 64-BIT MICROPROCESSOR

- **Parallel Architecture that Supports Up to Three Operations per Clock**
  - **— One Integer or Control Instruction per Clock**
  - **— Up to Two Floating-Point Results per Clock**
- **High Performance Design**
  - **— 33.3/40/50** MHz Clock Rates**
  - **— 80 Peak Single Precision MFLOPs**
  - **— 60 Peak Double Precision MFLOPs**
  - **— 64-Bit External Data Bus**
  - **— 64-Bit Internal Instruction Cache Bus**
  - **— 128-Bit Internal Data Cache Bus**
- **High Level of Integration on One Chip**
  - **— 32-Bit Integer and Control Unit**
  - **— 32/64-Bit Pipelined Floating-Point Adder and Multiplier Units**
  - **— 64-Bit 3-D Graphics Unit**
  - **— Paging Unit with Translation Lookaside Buffer**
  - **— 4 Kbyte Instruction Cache**
  - **— 8 Kbyte Data Cache**
- **Compatible with Industry Standards**
  - **— ANSI/IEEE Standard 754-1985 for Binary Floating-Point Arithmetic**
  - **— 386™/i486™ Microprocessor Data Formats and Page Table Entries**
  - **— JEDEC 168-pin Ceramic Pin Grid Array Package (see *Packaging Outlines and Dimensions*, order #231369)**
- **Easy to Use**
  - **— On-Chip Debug Register**
  - **— Assembler, Linker, Simulator, Debugger, C and FORTRAN Compilers, FORTRAN Vectorizer, Scalar and Vector Math Libraries for both OS/2* and UNIX* Environments**

The Intel i860™ Microprocessor (order codes A80860-33 and A80860-40) delivers supercomputing performance in a single VLSI component. The 64-bit design of the i860 microprocessor balances integer, floating point, and graphics performance for applications such as engineering workstations, scientific computing, 3-D graphics workstations, and multiuser systems. Its parallel architecture achieves high throughput with RISC design techniques, pipelined processing units, wide data paths, large on-chip caches, million-transistor design, and fast one-micron CHMOS IV silicon technology.

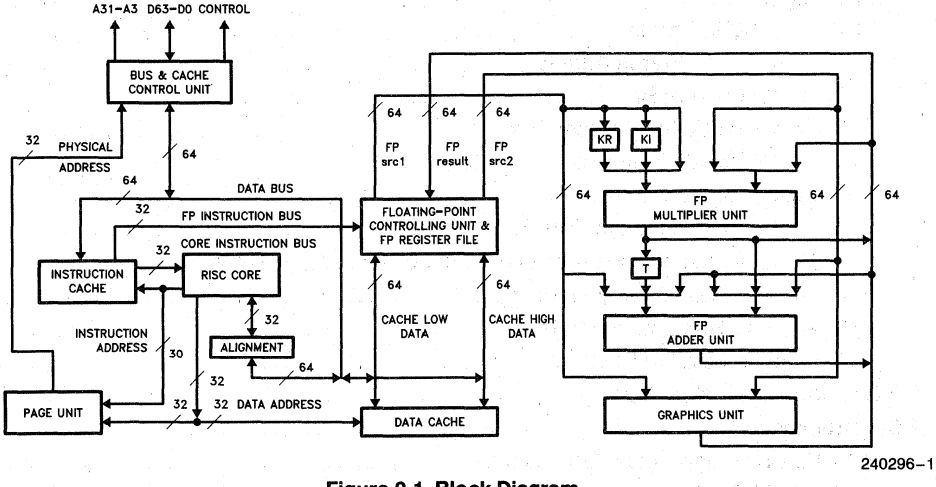


Figure 0.1. Block Diagram

Intel, intel, 386, i486, i860, Multibus II and Parallel System Bus are trademarks of Intel Corporation.
*UNIX is a registered trademark of AT&T. OS/2 is a trademark of International Business Machines Corporation.
**ADVANCE INFORMATION

November 1990
Order Number: 240296-004

TABLE OF CONTENTS

CONTENTS

PAGE

1.0 FUNCTIONAL DESCRIPTION 6-7

2.0 PROGRAMMING INTERFACE 6-7
2.1 Data Types 6-8
2.1.1 Integer 6-8
2.1.2 Ordinal 6-8
2.1.3 Single- and Double-Precision Real 6-8
2.1.4 Pixel 6-9
2.2 Register Set 6-9
2.2.1 Integer Register File 6-10
2.2.2 Floating-Point Register File 6-10
2.2.3 Processor Status Register 6-10
2.2.4 Extended Processor Status Register 6-13
2.2.5 Data Breakpoint Register 6-14
2.2.6 Directory Base Register 6-14
2.2.7 Fault Instruction Register 6-15
2.2.8 Floating-Point Status Register 6-15
2.2.9 KR, KI, T, and MERGE Registers 6-16
2.3 Addressing 6-17
2.4 Virtual Addressing 6-17
2.4.1 Page Forms 6-19
2.4.2 Virtual Address 6-19
2.4.3 Pages Tables 6-19
2.4.4 Page-Table Entries 6-20
2.4.4.1 Page Frame Address 6-20
2.4.4.2 Present Bit 6-20
2.4.4.3 Writable and User Bits 6-20
2.4.4.4 Write-Through Bit 6-21
2.4.4.5 Cache Disable Bit 6-21
2.4.4.6 Accessed and Dirty Bits 6-21
2.4.4.7 Combining Protection of Both Levels of Page Tables 6-21
2.4.5 Address Translation Algorithm 6-22
2.4.6 Address Translation Faults 6-23
2.4.7 Page Translation Cache 6-23
2.5 Caching and Cache Flushing 6-23
2.6 Instruction Set 6-24
2.6.1 Pipelined and Scalar Operations 6-24
2.6.1.1 Scalar Mode 6-24
2.6.1.2 Pipelining Status Information 6-24
2.6.1.3 Precision in the Pipelines 6-26
2.6.1.4 Transition between Scalar and Pipelined Operations 6-27

CONTENTS

PAGE
2.6.2 Dual-Instruction Mode ... 6-27
2.6.3 Dual-Operation Instruction ... 6-28
2.7 Addressing Modes ... 6-28
2.8 Traps and Interrupts ... 6-29
2.8.1 Trap Handler Invocation ... 6-29
2.8.2 Instruction Fault ... 6-30
2.8.3 Floating-Point Fault ... 6-30
2.8.3.1 Source Exception Faults ... 6-30
2.8.3.2 Result Exception Faults ... 6-30
2.8.4 Instruction Access Fault ... 6-31
2.8.5 Data Access Fault ... 6-31
2.8.6 Interrupt Trap ... 6-31
2.8.7 Reset Trap ... 6-31
2.9 Debugging ... 6-32

3.0 HARDWARE INTERFACE ... 6-32
3.1 Signal Description ... 6-32
3.1.1 Clock (CLK) ... 6-32
3.1.2 System Reset (RESET) ... 6-32
3.1.3 Bus Hold (HOLD) and Bus Hold Acknowledge (HLDA) ... 6-32
3.1.4 Bus Request (BREQ) ... 6-33
3.1.5 Interrupt/Code-Size (INT/CS8) ... 6-33
3.1.6 Address Pins (A31–A3) and Byte Enables (BE7#–BE0#) ... 6-34
3.1.7 Data Pins (D63–D0) ... 6-34
3.1.8 Bus Lock (LOCK#) ... 6-34
3.1.9 Write/Read Bus Cycle (W/R#) ... 6-35
3.1.10 Next Near (NENE#) ... 6-35
3.1.11 Next Address Request (NA#) ... 6-35
3.1.12 Transfer Acknowledge (READY#) ... 6-35
3.1.13 Address Status (ADS#) ... 6-35
3.1.14 Cache Enable (KEN#) ... 6-35
3.1.15 Page Table Bit (PTB) ... 6-36
3.1.16 Boundary Scan Shift Input (SHI) ... 6-36
3.1.17 Boundary Scan Enable (BSCN) ... 6-36
3.1.18 Shift Scan Path (SCAN) ... 6-36
3.1.19 Configuration (CC1–CC0) ... 6-36
3.1.20 System Power (V_{CC}) and Ground (V_{SS}) ... 6-36
3.2 Initialization ... 6-36
3.3 Testability ... 6-37
3.3.1 Normal Mode ... 6-38
3.3.2 Shift Mode ... 6-38

CONTENTS

PAGE

4.0 BUS OPERATION 6-38
4.1 Pipelining 6-38
4.2 Bus State Machine 6-39
4.3 Bus Cycles 6-41
4.3.1 Nonpipelined Read Cycles 6-41
4.3.2 Nonpipelined Write Cycles 6-42
4.3.3 Pipelined Read and Write Cycles 6-44
4.3.4 Locked Cycles 6-46
4.3.5 HOLD and BREQ Arbitration Cycles 6-46
4.4 Bus States during RESET 6-47

5.0 MECHANICAL DATA 6-48

6.0 PACKAGE THERMAL SPECIFICATIONS 6-53

7.0 ELECTRICAL DATA 6-55
7.1 Absolute Maximum Ratings 6-55
7.2 D.C. Characteristics 6-55
7.3 A.C. Characteristics 6-56

8.0 INSTRUCTION SET 6-59
8.1 Instruction Definitions in Alphabetical Order 6-60
8.2 Instruction Format and Encoding 6-67
8.2.1 REG-Format Instructions 6-67
8.2.2 CTRL-Format Instructions 6-70
8.2.3 Floating-Point Instructions 6-71
8.3 Instruction Timings 6-73
8.4 Instruction Characteristics 6-76

FIGURES

PAGE

Figure 0.1 Block Diagram 6-1
Figure 2.1 Real Number Formats 6-8
Figure 2.2 Pixel Format Example 6-9
Figure 2.3 Registers and Data Paths 6-11
Figure 2.4 Processor Status Register 6-12
Figure 2.5 Extended Processor Status Register 6-12
Figure 2.6 Directory Base Register 6-13
Figure 2.7 Floating-Point Status Register 6-15
Figure 2.8 Little and Big Endian Data Access 6-18
Figure 2.9 Format of a Virtual Address 6-19
Figure 2.10 Address Translation 6-19
Figure 2.11 Format of a Page Table Entry 6-20
Figure 2.12 Pipelined Instruction Execution 6-26
Figure 2.13 Dual-Instruction Mode Transitions 6-27
Figure 2.14 Dual-Operation Data Paths 6-28
Figure 3.1 Order of Boundary Scan Chain 6-38
Figure 4.1 Bus State Machine 6-40
Figure 4.2 Fastest Read Cycles 6-41
Figure 4.3 Fastest Write Cycles 6-42
Figure 4.4 Fastest Read/Write Cycles 6-43
Figure 4.5 Pipelined Read Followed by Pipelined Write 6-43
Figure 4.6 Pipelined Write Followed by Pipelined Read 6-44
Figure 4.7 Pipelining Driven by NA# 6-45
Figure 4.8 NA# Active with No Internal Bus Request 6-45
Figure 4.9 Locked Cycles 6-46
Figure 4.10 HOLD, HLDA, and BREQ 6-47
Figure 4.11 Reset Activities 6-47
Figure 5.1 Pin Configuration—View from Top Side 6-48
Figure 5.2 Pin Configuration—View from Pin Side 6-49
Figure 5.3 168-Lead Ceramic PGA Package Dimensions 6-53
Figure 6.1 I_{CC} vs Case Temperature 6-54
Figure 7.1 CLK, Input, and Output Timings 6-57
Figure 7.2 Typical Output Delay vs Load Capacitance under Worst-Case Conditions 6-58
Figure 7.3 Typical Slew Time vs Load Capacitance under Worst-Case Conditions 6-58
Figure 7.4 Typical I_{CC} vs Frequency 6-58
Figure 8.1 REG-Format Variations 6-68
Figure 8.2 Core Escape Instruction Format 6-69
Figure 8.3 CTRL Instruction Format 6-70
Figure 8.4 Floating-Point Instruction Encoding 6-71

TABLES

PAGE

Table 2.1 Pixel Formats 6-9
Table 2.2 Values of PS 6-13
Table 2.3 Values of RB 6-15
Table 2.4 Values of RC 6-15
Table 2.5 Values of RM 6-16
Table 2.6 Combining Directory and Page Protection 6-22
Table 2.7 Instruction Set 6-25
Table 2.8 Types of Traps 6-29
Table 2.9 Register and Cache Values after Reset 6-32
Table 3.1 Pin Summary 6-33
Table 3.2 Indentifying Instruction Fetches 6-35
Table 3.3 Cacheability based on KEN# and CD OR'ed WT 6-36
Table 3.4 Output Pin Status during RESET 6-37
Table 3.5 Test Mode Selection 6-37
Table 3.6 Test Mode Latches 6-37
Table 5.1 Pin Cross Reference by Location 6-50
Table 5.2 Pin Cross Reference by Pin Name 6-51
Table 5.3 Ceramic PGA Package Dimension Symbols 6-52
Table 6.1 θ_{CA} at Various Airflows and θ_{JC} 6-54
Table 6.2 Maximum T_A at Various Airflows 6-55
Table 7.1 D.C. Characteristics 6-55
Table 7.2 A.C. Characteristics 6-56
Table 8.1 Precision Specification 6-59
Table 8.2 FADDP MERGE Update 6-67
Table 8.3 Register Encoding 6-67
Table 8.4 REG-Format Opcodes 6-69
Table 8.5 Core Escape Opcodes 6-70
Table 8.6 CTRL-Format Opcodes 6-70
Table 8.7 Floating-Point Opcodes 6-71
Table 8.8 DPC Encoding 6-72
Table 8.9 Instruction Characteristics 6-77

1.0 FUNCTIONAL DESCRIPTION

As shown by the block diagram on the front page, the i860 microprocessor consists of 9 units:

1. Core Execution Unit
2. Floating-Point Control Unit
3. Floating-Point Adder Unit
4. Floating-Point Multiplier Unit
5. Graphics Unit
6. Paging Unit
7. Instruction Cache
8. Data Cache
9. Bus and Cache Control Unit

The core execution unit controls overall operation of the i860 microprocessor. The core unit executes load, store, integer, bit, and control-transfer operations, and fetches instructions for the floating-point unit as well. A set of 32 x 32-bit general-purpose registers are provided for the manipulation of integer data. Load and store instructions move 8-, 16-, and 32-bit data to and from these registers. Its full set of integer, logical, and control-transfer instructions give the core unit the ability to execute complete systems software and applications programs. A trap mechanism provides rapid response to exceptions and external interrupts. Debugging is supported by the ability to trap on data or instruction reference.

The floating-point hardware is connected to a separate set of floating-point registers, which can be accessed as 16 x 64-bit registers, or 32 x 32-bit registers. Special load and store instructions can also access these same registers as 8 x 128-bit registers. All floating-point instructions use these registers as their source and destination operands.

The floating-point control unit controls both the floating-point adder and the floating-point multiplier, issuing instructions, handling all source and result exceptions, and updating status bits in the floating-point status register. The adder and multiplier can operate in parallel, producing up to two results per clock. The floating-point data types, floating-point instructions, and exception handling all support the IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985).

The floating-point adder performs addition, subtraction, comparison, and conversions on 64- and 32-bit floating-point values. An adder instruction executes in three clocks; however, in pipelined mode, a new result is generated every clock.

The floating-point multiplier performs floating-point and integer multiply and floating-point reciprocal operations on 64- and 32-bit floating-point values. A multiplier instruction executes in three to four clocks; however, in pipelined mode, a new result can be generated every clock for single-precision and every other clock for double precision.

The graphics unit has special integer logic that supports three-dimensional drawing in a graphics frame buffer, with color intensity shading and hidden surface elimination via the Z-buffer algorithm. The graphics unit recognizes the pixel as an 8-, 16-, or 32-bit data type. It can compute individual red, blue, and green color intensity values within a pixel; but it does so with parallel operations that take advantage of the 64-bit internal word size and 64-bit external bus. The graphics features of the i860 microprocessor assume that the surface of a solid object is drawn with polygon patches whose shapes approximate the original object. The color intensities of the vertices of the polygon and their distances from the viewer are known, but the distances and intensities of the other points must be calculated by interpolation. The graphics instructions of the i860 microprocessor directly aid such interpolation.

The paging unit implements protected, paged, virtual memory via a 64-entry, four-way set-associative memory called the TLB (Translation Lookaside Buffer). The paging unit uses the TLB to perform the translation of logical address to physical address, and to check for access violations. The access protection scheme employs two levels of privilege: user and supervisor.

The instruction cache is a two-way set-associative memory of four Kbytes, with 32-byte blocks. It transfers up to 64 bits per clock (400 Mbyte/sec at 50 MHz).

The data cache is a two-way set-associative memory of eight Kbytes, with 32-byte blocks. It transfers up to 128 bits per clock (800 Mbyte/sec at 50 MHz). The i860 microprocessor normally uses writeback caching, i.e. memory writes update the cache (if applicable) without necessarily updating memory immediately; however, caching can be inhibited by software where necessary.

The bus and cache control unit performs data and instruction accesses for the core unit. It receives cycle requests and specifications from the core unit, performs the data-cache or instuction-cache miss processing, controls TLB translation, and provides the interface to the external bus. Its pipelined structure supports up to three outstanding bus cycles.

2.0 PROGRAMMING INTERFACE

The programmer-visible aspects of the architecture of the i860 microprocessor include data types, registers, instructions, and traps.

2.1 Data Types

The i860 microprocessor provides operations for integer and floating-point data. Integer operations are performed on 32-bit operands with some support also for 64-bit operands. Load and store instructions can reference 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit operands. Floating-point operations are performed on IEEE-standard 32- and 64-bit formats. Graphics oriented instructions operate on arrays of 8-, 16-, or 32-bit pixels.

2.1.1 INTEGER

An integer is a 32-bit signed value in standard two's complement form. A 32-bit integer can represent a value in the range $-2{,}147{,}483{,}648$ (-2^{31}) to $2{,}147{,}483{,}647$ $(+2^{31}-1)$. Arithmetic operations on 8- and 16-bit integers can be performed by sign-extending the 8- or 16-bit values to 32 bits, then using the 32-bit operations.

There are also add and subtract instructions that operate on 64-bit long integers.

Load and store instructions may also reference (in addition to the 32- and 64-bit formats previously mentioned) 8- and 16-bit items in memory. When an 8- or 16-bit item is loaded into a register, it is converted to an integer by sign-extending the value to 32 bits. When an 8- or 16-bit item is stored from a register, the corresponding number of low-order bits of the register are used.

2.1.2 ORDINAL

Arithmetic operations are available for 32-bit ordinals. An ordinal is an unsigned integer. An ordinal can represent values in the range 0 to $4{,}294{,}967{,}295$ $(+2^{32}-1)$.

Also, there are add and subtract instructions that operate on 64-bit ordinals.

2.1.3 SINGLE- AND DOUBLE-PRECISION REAL

Figure 2.1 shows the real number formats. A single-precision real (also called "single real") data type is a 32-bit binary floating-point number. Bit 31 is the sign bit; bits 30..23 are the exponent; and bits 22..0 are the fraction. In accordance with ANSI/IEEE standard 754, the value of a single-precision real is defined as follows:

1. If $e = 0$ and $f \neq 0$ or $e = 255$ then generate a floating-point source-exception trap when encountered in a floating-point operation.
2. If $0 < e < 255$, then the value is $(-1)^s \times 1.f \times 2^{e-127}$.
3. If $e = 0$ and $f = 0$, then the value is signed zero.

A double-precision real (also called "double real") data type is a 64-bit binary floating-point number. Bit 63 is the sign bit; bits 62..52 are the exponent; and bits 51..0 are the fraction. In accordance with ANSI/IEEE standard 754, the value of a double-precision real is defined as follows:

1. If $e = 0$ and $f \neq 0$ or $e = 2047$, then generate a floating-point source-exception trap when encountered in a floating-point operation.
2. If $0 < e < 2047$, then the value is $(-1)^s \times 1.f \times 2^{e-1023}$.

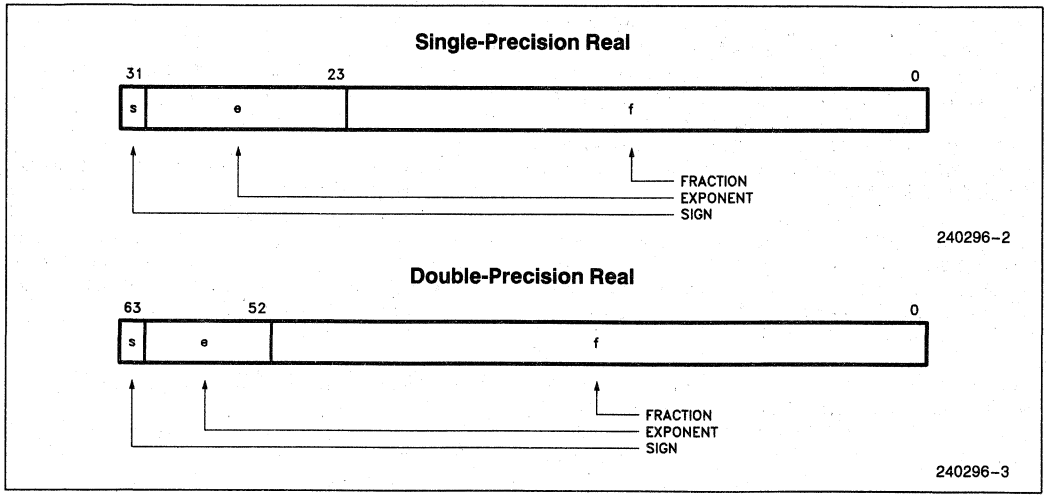


Figure 2.1. Real Number Formats

3. If e = 0 and f = 0, then the value is signed zero.

The special values infinity, NaN ("Not a Number"), indefinite, and denormal generate a trap when encountered. The trap handler implements IEEE-standard results.

A double real value occupies an even/odd pair of floating-point registers. Bits 31..0 are stored in the even-numbered floating-point register; bits 63..32 are stored in the next higher odd-numbered floating-point register.

2.1.4 PIXEL

A pixel may be 8, 16, or 32 bits long depending on color and intensity resolution requirements. Regardless of the pixel size, the i860 microprocessor always operates on 64 bits worth of pixels at a time. The pixel data type is used by two kinds of instructions:

- The selective pixel-store instruction that helps implement hidden surface elimination.
- The pixel add instruction that helps implement 3-D color intensity shading.

To perform color intensity shading efficiently in a variety of applications, the i860 microprocessor defines three pixel formats according to Table 2.1.

Figure 2.2 illustrates one way of assigning meaning to the fields of pixels. These assignments are for illustration purposes only. The i860 microprocessor defines only the field sizes, not the specific use of each field. Other ways of using the fields of pixels are possible.

Table 2.1. Pixel Formats

Pixel Size (in bits)	Bits of Color 1 Intensity	Bits of Color 2 Intensity	Bits of Color 3 Intensity	Bits of Other Attribute (Texture)
8	N (≤ 8) bits of intensity*			8 − N
16	6	6	4	
32	8	8	8	8

The intensity attribute fields may be assigned to colors in any order convenient to the application.

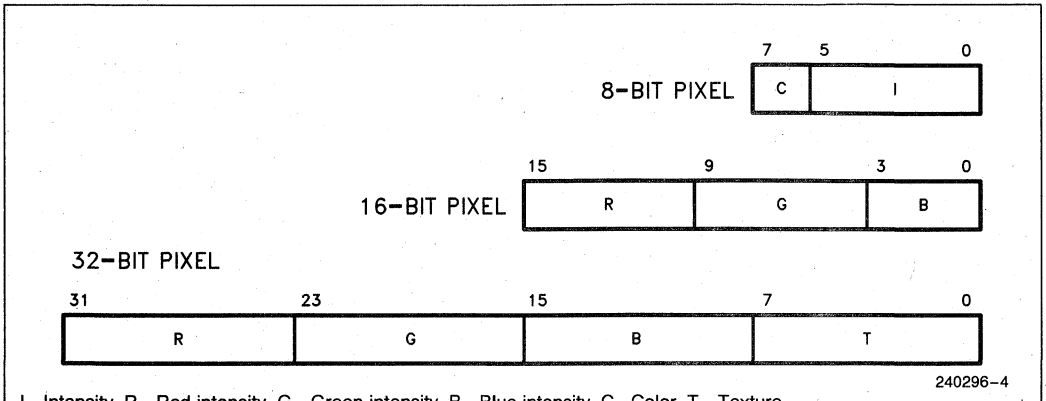
*With 8-bit pixels, up to 8 bits can be used for intensity; the remaining bits can be used for any other attribute, such as color. The intensity bits must be the low-order bits of the pixel.

2.2 Register Set

As Figure 2.3 shows, the i860 microprocessor has the following registers:

- An integer register file
- A floating-point register file
- Six control registers (**psr, epsr, db, dirbase, fir,** and **fsr)**
- Four special-purpose registers (KR, KI, T, and MERGE)

The control registers are accessible only by load and store control-register instructions; the integer and floating-point registers are accessed by arithmetic operations and load and store instructions. The special-purpose registers KR, KI, T, and MERGE are used by a few specific instructions.



I—Intensity, R—Red intensity, G—Green intensity, B—Blue intensity, C—Color, T—Texture
These assignments of specific meanings to the fields of pixels are for illustration purposes only. Only the field sizes are defined, not the specific use of each field.

Figure 2.2. Pixel Format Example

2.2.1 INTEGER REGISTER FILE

There are 32 integer registers, each 32 bits wide, referred to as **r0** through **r31**, which are used for address computation and scalar integer computations. Register **r0** always returns zero when read, independently of what is stored in it.

2.2.2 FLOATING-POINT REGISTER FILE

There are 32 floating-point registers, each 32-bits wide, referred to as **f0** through **f31**, which are used for floating-point computations. Registers **f0** and **f1** always return zero when read, independently of what is stored in them. The floating-point registers are also used by a set of graphics operations, primarily for 3D graphics computations.

When accessing 64-bit floating-point or integer values, the i860 microprocessor uses an even/odd pair of registers. When accessing 128-bit values, it uses an aligned set of four registers (**f0**, **f4**, **f8**, . . . , **f28**). The instruction must designate the lowest register number of the set of registers containing 64- or 128-bit values. Misaligned register numbers produce undefined results. The register with the lowest number contains the least significant part of the value. For 128-bit values, the register pair with the lower numbers contain the least significant 64 bits while the register pair with the higher numbers contain the most significant 64 bits.

The 128-bit load and store instructions, along with the 128-bit data path between the floating-point registers and the data cache help to sustain the extraordinarily high rate of computation.

2.2.3 PROCESSOR STATUS REGISTER

The processor status register (**psr**) contains miscellaneous state information for the current process. Figure 2.4 shows the format of the **psr**.

- BR (Break Read) and BW (Break Write) enable a data access trap when the operand address matches the address in the **db** register and a read or write (respectively) occurs.
- Various instructions set CC (Condition Code) according to tests they perform. The branch-on-condition-code instructions test its value. The **bla** instruction sets and tests LCC (Loop Condition Code).
- IM (Interrupt Mode) enables external interrupts if set; disables interrupts if clear.
- U (User Mode) is set when the i860 microprocessor is executing in user mode; it is clear when the i860 microprocessor is executing in supervisor mode. In user mode, writes to some control registers are inhibited. This bit also controls the memory protection mechanism. See section 2.4.4.3 for a description of memory protection in user and supervisor modes.

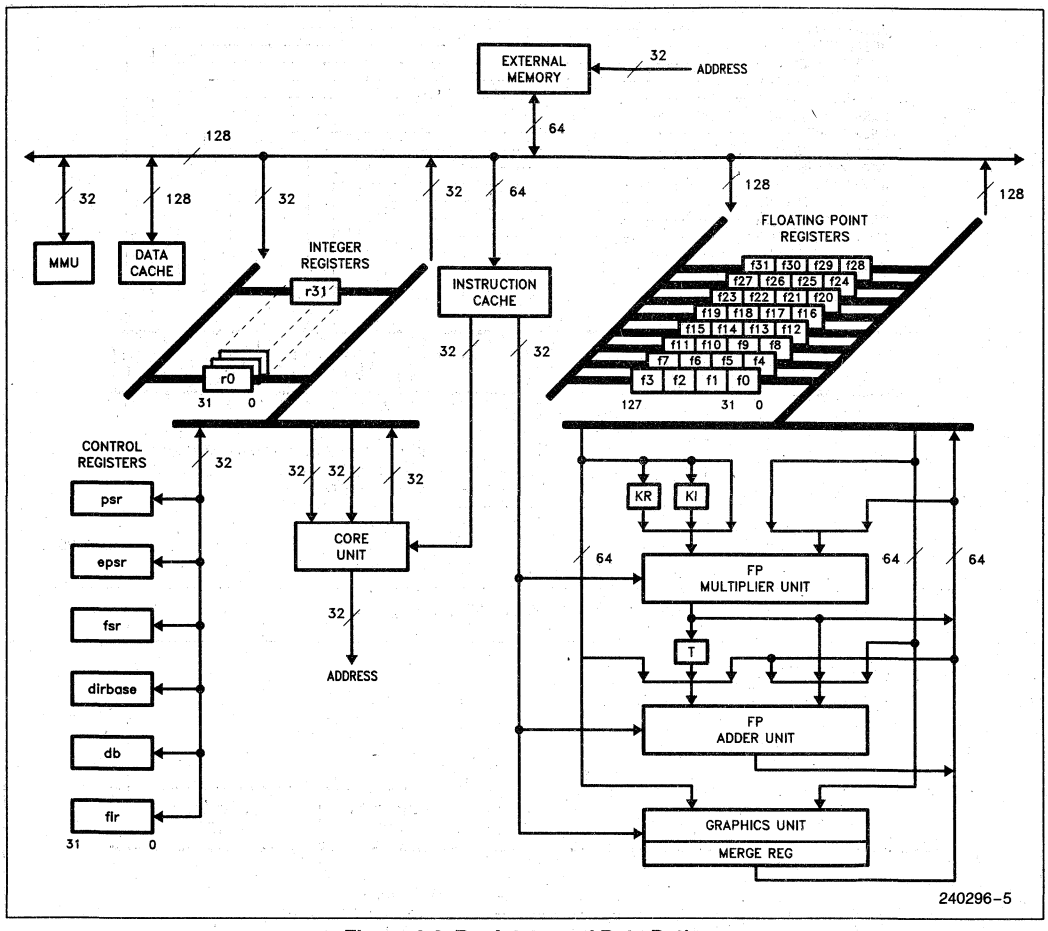
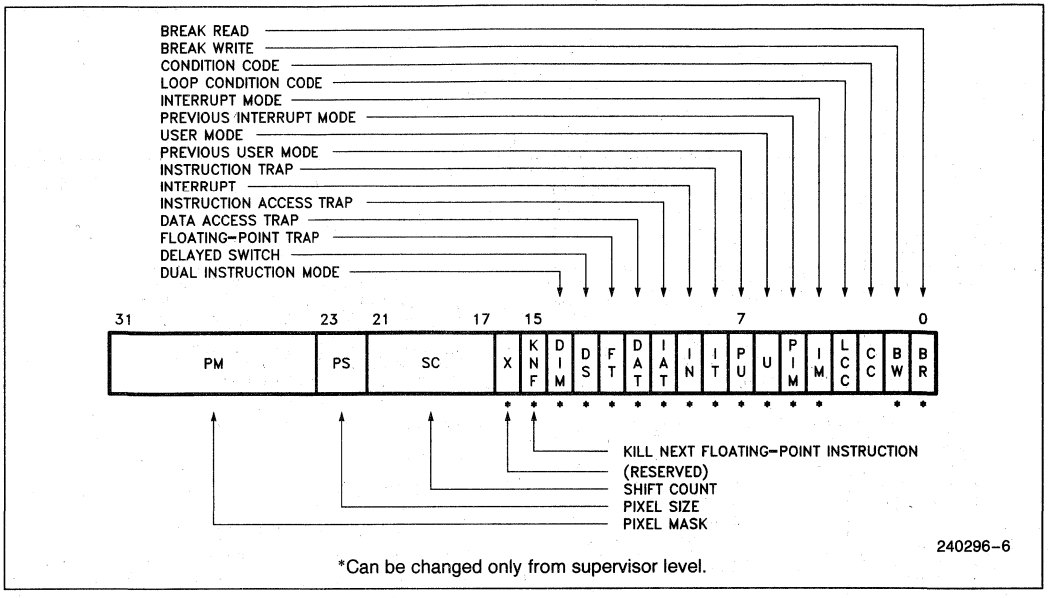


Figure 2.3. Registers and Data Paths

6



*Can be changed only from supervisor level.

Figure 2.4 Processor Status Register

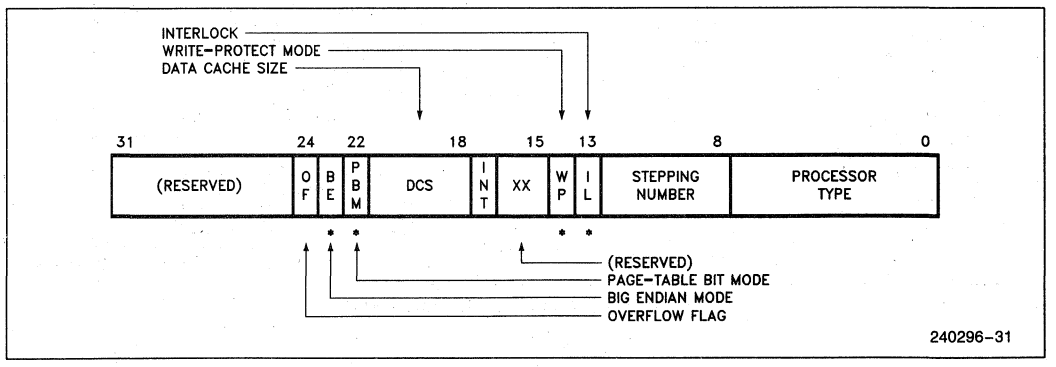


Figure 2.5 Extended Processor Status Register

- PIM (Previous Interrupt Mode) and PU (Previous User Mode) save the corresponding status bits (IM and U) on a trap, because those status bits are changed when a trap occurs. They are restored into their corresponding status bits when returning from a trap handler with a branch indirect instruction when a trap flag is set in the **psr**.
- FT (Floating-Point Trap), DAT (Data Access Trap), IAT (Instruction Access Trap), IN (Interrupt), and IT (Instruction Trap) are trap flags. They are set when the corresponding trap condition occurs. The trap handler examines these bits to determine which condition or conditions have caused the trap.
- DS (Delayed Switch) is set if a trap occurs during the instruction before dual-instruction mode is entered or exited. If DS is set and DIM (Dual Instruction Mode) is clear, the i860 microprocessor switches to dual-instruction mode one instruction after returning from the trap handler. If DS and DIM are both set, the i860 microprocessor switches to single-instruction mode one instruction after returning from the trap handler.
- When a trap occurs, the i860 microprocessor sets DIM if it is executing in dual-instruction mode; it clears DIM if it is executing in single-instruction mode. If DIM is set after returning from a trap handler, the i860 microprocessor resumes execution in dual-instruction mode.

- When KNF (Kill Next Floating-Point Instruction) is set, the next floating-point instruction is suppressed (except that its dual-instruction mode bit is interpreted). A trap handler sets KNF if the trapped floating-point instruction should not be reexecuted.
- SC (Shift Count) stores the shift count used by the last right-shift instruction. It controls the number of shifts executed by the double-shift instruction.
- PS (Pixel Size) and PM (Pixel Mask) are used by the pixel-store instruction and by the graphics instructions. The values of PS control pixel size as defined by Table 2.2. The bits in PM correspond to pixels to be updated by the pixel-store instruction **pst.d**. The low-order bit of PM corresponds to the low-order pixel of the 64-bit source operand of **pst.d**. The number of low-order bits of PM that are actually used is the number of pixels that fit into 64-bits, which depends upon PS. If a bit of PM is set, then **pst.d** stores the corresponding pixel. Refer also to the **pst.d** instruction in section 8.

Table 2.2. Values of PS

Value	Pixel Size in bits	Pixel Size in bytes
00	8	1
01	16	2
10	32	4
11	(undefined)	(undefined)

2.2.4 EXTENDED PROCESSOR STATUS REGISTER

The extended processor status register (**epsr**) contains additional state information for the current process beyond that stored in the **psr**. Figure 2.5 shows the format of the **epsr**.

- The processor type is one for the i860 microprocessor.
- The stepping number has a unique value that distinguishes among different revisions of the processor.
- IL (Interlock) is set if a trap occurs after a **lock** instruction but before the load or store following the subsequent **unlock** instruction. IL indicates to the trap handler that a locked sequence has been interrupted. When the trap handler finds IL set, it should scan backwards for the **lock** instruction and restart at that point. The absence of a **lock** instruction within 30–33 instructions of the trap indicates a programming error.
- WP (write protect) controls the semantics of the W bit of page table entries. A clear W bit in either the directory or the page table entry causes writes to be trapped. When WP is clear, writes are trapped in user mode, but not in supervisor mode. When WP is set, writes are trapped in both user and supervisor modes. After the value of the WP bit is changed, the TLB must be invalidated by setting the ITI bit of the **dirbase** register, before any stores are performed.
- INT (Interrupt) is the value of the INT input pin.
- DCS (Data Cache Size) is a read-only field that tells the size of the on-chip data cache. The number of bytes actually available is 2^{12+DCS}; therefore, a value of zero indicates 4 Kbytes, one indicates 8 Kbytes, etc.

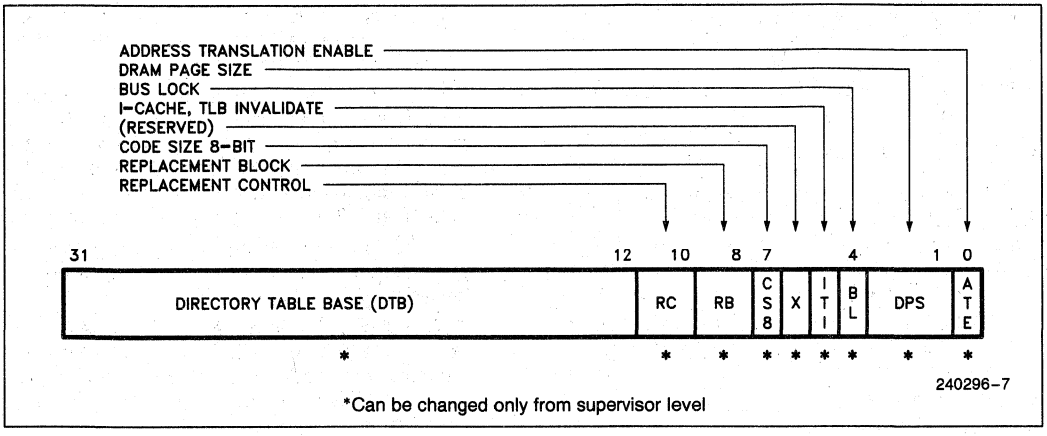


Figure 2.6. Directory Base Register

- PBM (Page-Table Bit Mode) determines which bit of page-table entries is output on the PTB pin. When PBM is clear, the PTB signal reflects bit CD of the page-table entry used for the current cycle. When PBM is set, the PTB signal reflects bit WT of the page-table entry used for the current cycle.
- BE (Big Endian) controls the ordering of bytes within a data item in memory. Normally (i.e. when BE is clear) the i860 microprocessor operates in little endian mode, in which the addressed byte is the low-order byte. When BE is set (big endian mode), the low-order three bits of all load and store addresses are complemented, then masked to the appropriate boundary for alignment. This causes the addressed byte to be the most significant byte. Section 2.3 discusses little and big endian addressing.
- OF (Overflow Flag) is set by **adds**, **addu**, **subs**, and **subu** when integer overflow occurs. For **adds** and **subs**, OF is set if the carry from bit 31 is different than the carry from bit 30. For **addu**, OF is set if there is a carry from bit 31. For **subu**, OF is set if there is no carry from bit 31. Under all other conditions, it is cleared by these instructions. OF controls the function of the **intovr** instruction.

2.2.5 DATA BREAKPOINT REGISTER

The data breakpoint register (**db**) is used to generate a trap when the i860 microprocessor makes a data-operand access to the address stored in this register. The trap is enabled by BR and BW in **psr**. The **db** register can only be changed from supervisor level. When comparing, a number of low order bits of the address are ignored, depending on the size of the operand. For example, a 16-bit access ignores the low-order bit of the address when comparing to **db**; a 32-bit access ignores the low-order two bits. This ensures that any access that overlaps the address contained in the register will generate a trap. The DAT occurs before the data is accessed and prevents the load or store from completing.

2.2.6 DIRECTORY BASE REGISTER

The directory base register **dirbase** (shown in Figure 2.6) controls address translation, caching, and bus options. The **dirbase** register can only be changed from supervisor level. The BL bit is changed from user level with the **lock** and **unlock** instructions.

- ATE (Address Translation Enable), when set, enables the virtual-address translation algorithm. The data cache must be flushed before changing the ATE bit.
- DPS (DRAM Page Size) controls how many bits to ignore when comparing the current bus-cycle address with the previous bus-cycle address to generate the NENE# signal. This feature allows for higher speeds when using static column or page-mode DRAMs and consecutive reads and writes access the row. The comparison ignores the low-order 12 + DPS bits. A value of zero is appropriate for one bank of 256K $\times$ n RAMs, 1 for 1M $\times$ n RAMS, etc. For interleaved memory, increase DPS by one for each power of interleaving—add one for 2-way, and two for 4-way, etc.
- When BL (Bus Lock) is set, external bus accesses are locked. The LOCK# signal is asserted the next bus cycle whose internal bus request is generated after BL is set. It remains set on every subsequent bus cycle as long as BL remains set. The LOCK# signal is deasserted on the next load or store instruction after BL is cleared. Traps immediately clear BL. The **lock** and **unlock** instructions control the BL bit. The result of modifying BL with the **st.c** instruction is not defined.
- ITI (I-Cache, TLB Invalidate), when set in the value that is loaded into **dirbase**, causes all entries in the instruction cache and address-translation cache (TLB) to be invalidated. The ITI bit does not remain set in **dirbase**. ITI always appears as zero when reading **dirbase**. Section 2.5 discusses flushing the data cache before invalidating the TLB.
- When CS8 (Code Size 8-Bit) is set, instruction cache misses are processed as 8-bit bus cycles. When this bit is clear, instruction cache misses are processed as 64-bit bus cycles. This bit can not be set by software; hardware sets this bit at initialization time. It can be cleared by software (one time only) to allow the system to execute out of 64-bit memory after bootstrapping from 8-bit EPROM. A nondelayed branch to code in 64-bit memory should directly follow the **st.c** (store control register) instruction that clears CS8, in order to make the transition from 8-bit to 64-bit memory occur at the correct time. The branch must be aligned on a 64-bit boundary.
- RB (Replacement Block) identifies the cache block to be replaced by cache replacement algorithms. The high-order bit of RB is ignored by the instruction and data caches. RB conditions the cache flush instruction **flush**, which is discussed in Section 8. Table 2.3 explains the values of RB.
- RC (Replacement Control) controls cache replacement algorithms. Table 2.4 explains the significance of the values of RC.
- DTB (Directory Table Base) contains the high-order 20 bits of the physical address of the page directory when address translation is enabled (i.e. ATE = 1). The low-order 12 bits of the address are zeros.

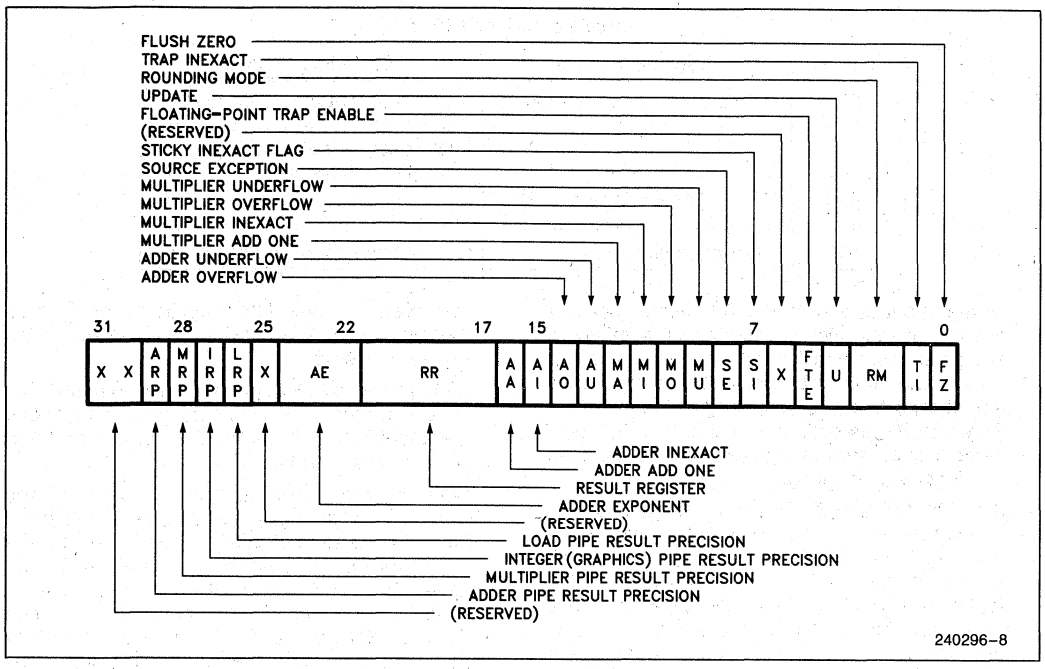


Figure 2.7. Floating-Point Status Register

Table 2.3. Values of RB

Value	Replace TLB Block	Replace Instruction and Data Cache Block
0 0	0	0
0 1	1	1
1 0	2	0
1 1	3	1

Table 2.4. Values of RC

Value	Meaning
00	Selects the normal replacement algorithm where any block in the set may be replaced on cache misses in all caches.
01	Instruction, data, and TLB cache misses replace the block selected by RB. The instruction and data caches ignore the high-order bit of RB. This mode is used for instruction cache and TLB testing.
10	Data cache misses replace the block selected by the low-order bit of RB. Instruction and TLB caches use random replacement.
11	Disables data cache replacement. Instruction and TLB caches use random replacement.

2.2.7 FAULT INSTRUCTION REGISTER

When a trap occurs, this register contains the address of the trapping instruction (not necessarily the instruction that created the conditions that required the trap). The **fir** is a read-only register. In single-instruction mode, using a **ld.c** instruction to read the **fir** anytime except the first time after a trap saves in *idest* the address of the **ld.c** instruction; in dual-instruction mode, the address of its floating-point companion (address of the **ld.c** − 4) is saved.

2.2.8 FLOATING-POINT STATUS REGISTER

The floating-point status register (**fsr**) contains the floating-point trap and rounding-mode status for the current process. Figure 2.7 shows its format. The **fsr** is writable in user level.

- If FZ (Flush Zero) is clear and underflow occurs, a result-exception trap is generated. When FZ is set and underflow occurs, the result is set to zero, and no trap due to underflow occurs.
- If TI (Trap Inexact) is clear, inexact results do not cause a trap. If TI is set, inexact results cause a trap. The sticky inexact flag (SI) is set whenever an inexact result is produced, regardless of the setting of TI.
- RM (Rounding Mode) specifies one of the four rounding modes defined by the IEEE standard. Given a true result *b* that cannot be represented

Table 2.5. Values of RM

Value	Rounding Mode	Rounding Action
00	Round to nearest or even	Closer to *b* of *a* or *c*; if equally close, select even number (the one whose least significant bit is zero).
01	Round down (toward $-\infty$)	*a*
10	Round up (toward $+\infty$	*c*
11	Chop (toward zero)	Smaller in magnitude of *a* or *c*.

by the target data type, the i860 microprocessor determines the two representable numbers *a* and *c* that most closely bracket *b* in value ($a < b < c$). The i860 microprocessor then rounds (changes) *b* to *a* or *c* according to the mode selected by RM as defined in Table 2.5. Rounding introduces an error in the result that is less than one least-significant bit.

- The U-bit (Update Bit), if set in the value that is loaded into **fsr** by a **st.c** instruction, enables updating of the result-status bits (AE, AA, AI, AO, AU, MA, MI, MO, and MU) in the first-stage of the floating-point adder and multiplier pipelines. If this bit is clear, the result-status bits are unaffected by a **st.c** instruction; **st.c** ignores the corresponding bits in the value that is being loaded. A **st.c** always updates **fsr** bits 21..17 and 8..0 directly. The U-bit does not remain set; it always appears as zero when read.
- The FTE (Floating-Point Trap Enable) bit, if clear, disables all floating-point traps (invalid input operand, overflow, underflow, and inexact result).
- SI (Sticky Inexact) is set when the last stage result of either the multiplier or adder is inexact (i.e. when either AI or MI is set). SI is "sticky" in the sense that it remains set until reset by software. AI and MI, on the other hand, can by changed by the subsequent floating-point instruction.
- SE (Source Exception) is set when one of the source operands of a floating-point operation is invalid; it is cleared when all the input operands are valid. Invalid input operands include denormals, infinities, and all NaNs (both quiet and signaling).
- When read from the **fsr,** the result-status bits MA, MI, MO, and MU (Multiplier Add-One, Inexact, Overflow, and Underflow, respectively) describe the last stage result of the multiplier.

  When read from the **fsr**, the result-status bits AA, AI, AO, AU, and AE (Adder Add-One, Inexact, Overflow, Underflow, and Exponent, respectively) describe the last stage result of the adder. The high-order three bits of the 11-bit exponent of the adder result are stored in the AE field.

  The Adder Add One and Multiplier Add One bits indicate that the absolute value of the result fraction grew by one least-significant bit due to rounding. AA and MA are not influenced by the sign of the result.

  After a floating-point operation in a given unit (adder or multiplier), the result-status bits of that unit are undefined until the point at which result exceptions are reported.

  When written to the **fsr** with the U-bit set, the result-status bits are placed into the first stage of the adder and multiplier pipelines. When the processor executes pipelined operations, it propagates the result-status bits of a particular unit (multiplier or adder) one stage for each pipelined floating-point operation for that unit. When they reach the last stage, they replace the normal result-status bits in the **fsr.** When the U-bit is not set, result-status bits in the word being written to the **fsr** are ignored.

  In a floating-point dual-operation instruction (e.g. add-and-multiply or subtract-and-multiply), both the multiplier and the adder may set exception bits. The result-status bits for a particular unit remain set until the next operation that uses that unit.
- RR (Result Register) specifies which floating-point register (**f0–f31**) was the destination register when a result-exception trap occurs due to a scalar operation.
- LRP (Load Pipe Result Precision), IRP (Integer (Graphics) Pipe Result Precision), MRP (Multiplier Pipe Result Precision), and ARP (Adder Pipe Result Precision) aid in restoring pipeline state after a trap or process switch. Each defines the precision of the last stage result in the corresponding pipeline. One of these bits is set when the result in the last stage of the corresponding pipeline is double precision; it is cleared if the result is single precision. These bits cannot be changed by software.

2.2.9 KR, KI, T, AND MERGE REGISTERS

The KR, KI, and T registers are special-purpose registers used by the dual-operation floating-point instructions **pfam, pfmam, pfsm,** and **pfmsm,**

which initiate both an adder (A-unit) operation and a multiplier (M-unit) operation. The KR, KI, and T registers can store values from one dual-operation instruction and supply them as inputs to subsequent dual-operation instructions. (Refer to Figure 2.14.)

The MERGE register is used only by the graphics instructions. The purpose of the MERGE register is to accumulate (or merge) the results of multiple-addition operations that use as operands the color-intensity values from pixels or distance values from a Z-buffer. The accumulated results can then be stored in one 64-bit operation.

Two multiple-addition instructions and an OR instruction use the MERGE register. The addition instructions are designed to add interpolation values to each color-intensity field in an array of pixels or to each distance value in a Z-buffer.

Refer to the instruction descriptions in section 8 for more information about these registers.

2.3 Addressing

Memory is addressed in byte units with a paged virtual-address space of 2^{32} bytes. Data and instructions can be located anywhere in this address space. Address arithmetic is performed using 32-bit input values and produces 32-bit results. The low-order 32 bits of the result are used in case of overflow.

Normally, multibyte data values are stored in memory in little endian format, i.e., with the least significant byte at the lowest memory address. As an option, the ordering can be dynamically selected by software in supervisor mode. The i860 microprocessor also offers big endian mode, in which the most significant byte of a data item is at the lowest address. Figure 2.8 shows the difference between the two storage modes. Big endian and little endian data areas should not be mixed within a 64-bit data word. Illustrations of data structures in this data sheet show data stored in little endian mode, i.e., the low-order byte is at the lowest memory address.

Code accesses are always done with little endian addressing. This implies that code will appear differently than documented here when accessed as big endian data. Intel recommends that disassemblers running in a big endian system, convert instructions which have been read as data back to little endian form and present them in the format documented here.

Page directories and page tables are also accessed in little endian mode, regardless of the value of the BE bit.

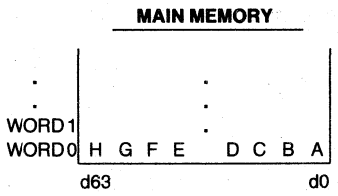
Alignment requirements are as follows (any violation results in a data-access trap):

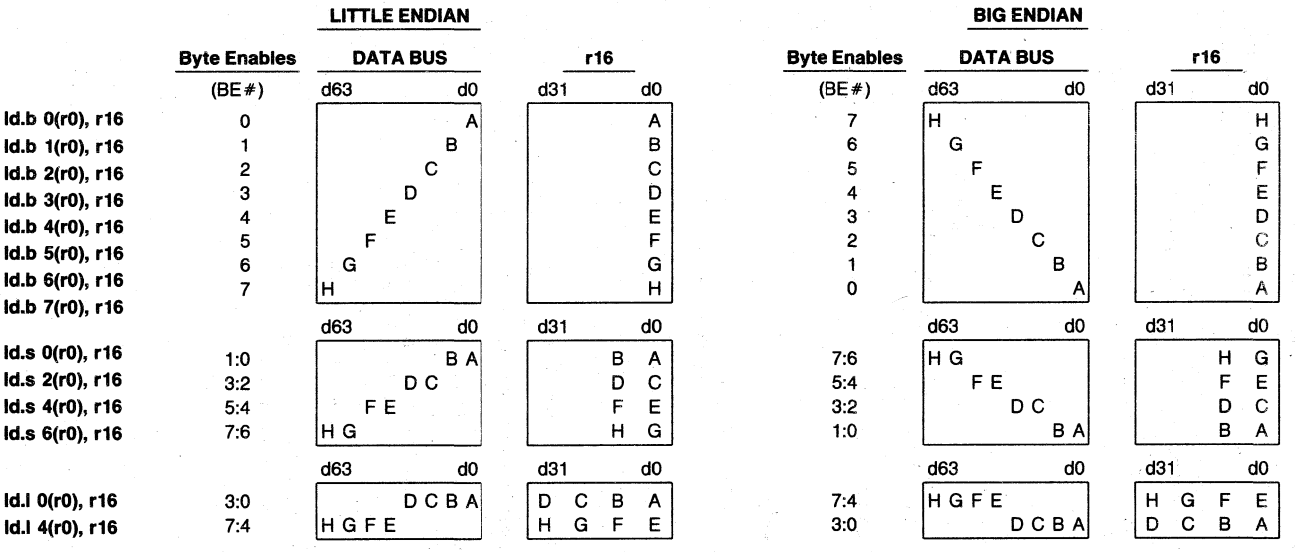
- 128-bit values are aligned on 16-byte boundaries when referenced in memory (i.e. the four least significant address bits must be zero).
- 64-bit values are aligned on 8-byte boundaries when referenced in memory (i.e. the three least significant address bits must be zero).
- 32-bit values are aligned on 4-byte boundaries when referenced in memory (i.e. the two least significant address bits must be zero).
- 16-bit values are aligned on 2-byte boundaries when referenced in memory (i.e. the least significant address bit must be zero).

2.4 Virtual Addressing

When address translation is enabled, the i860 microprocessor maps instruction and data virtual addresses into physical addresses before referencing memory. This address transformation is compatible with that of the 386 microprocessor and implements the basic features needed for page-oriented virtual-memory systems and page-level protection.

The address translation is optional. Address translation is in effect only when the ATE bit of **dirbase** is set. This bit is typically set by the operating system during software initialization. The ATE bit must be set if the operating system is to implement page-oriented protection or page-oriented virtual memory.





NOTE:
64- and 128-bit big endian accesses are treated the same as little endian accesses.

Figure 2.8 Little and Big Endian Accesses

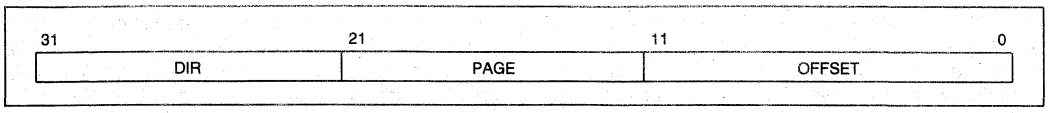


Figure 2.9. Format of a Virtual Address

Address translation is disabled when the processor is reset. It is enabled when a store to **dirbase** sets the ATE bit. It is disabled again when a store clears the ATE bit.

2.4.1 PAGE FRAME

A **page frame** is a 4-Kbyte unit of contiguous addresses of physical main memory. Page frames begin on 4-Kbyte boundaries and are fixed in size. A **page** is the collection of data that occupies a page frame when that data is present in main memory. The data may also occupy some location in secondary storage when there is not sufficient space in main memory.

2.4.2 VIRTUAL ADDRESS

A virtual address refers indirectly to a physical address by specifying a page table, a page within that table, and an offset within that page. Figure 2.9 shows the format of a virtual address.

Figure 2.10 shows how the i860 microprocessor converts the DIR, PAGE, and OFFSET fields of a virtual address into the physical address by consulting two levels of page tables. The addressing mechanism uses the DIR field as an index into a page directory, uses the PAGE field as an index into the page table determined by the page directory, and uses the OFFSET field to address a byte within the page determined by the page table.

2.4.3 PAGE TABLES

A page table is simply an array of 32-bit page specifiers. A page table is itself a page, and therefore contains 4 Kbytes of memory or at most 1K 32-bit entries.

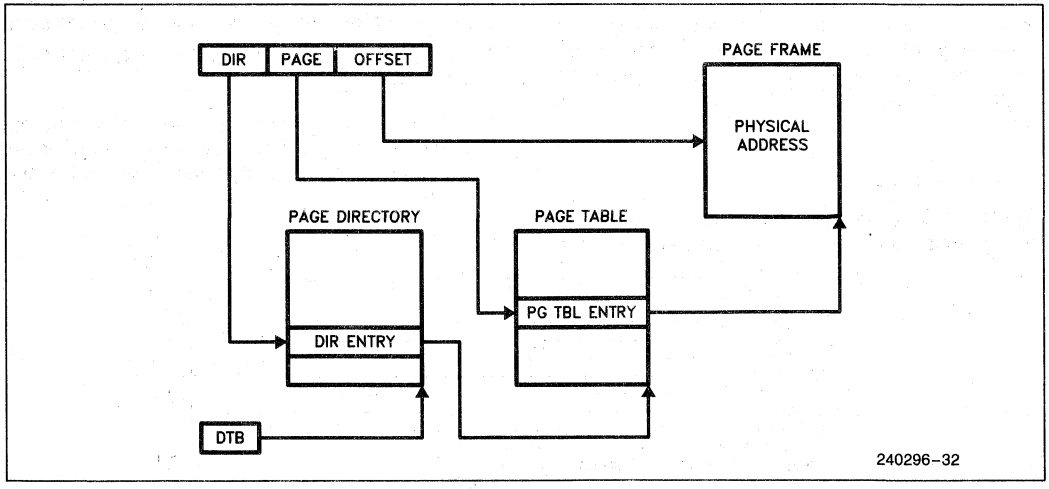


Figure 2.10. Address Translation

Two levels of tables are used to address a page of memory. At the higher level is a page directory. The page directory addresses up to 1K page tables of the second level. A page table of the second level addresses up to 1K pages. All the tables addressed by one page directory, therefore, can address 1M pages (2^{20}). Because each page contains 4 Kbytes (2^{12} bytes), the tables of one page directory can span the entire physical address space of the i860 microprocessor ($2^{20} \times 2^{12} = 2^{32}$).

The physical address of the current page directory is stored in DTB field of the **dirbase** register. Memory management software has the option of using one page directory for all processes, one page directory for each process, or some combination of the two.

2.4.4 PAGE-TABLE ENTRIES

Page-table entries (PTEs) in either level of page tables have the same format. Figure 2.11 illustrates this format.

2.4.4.1 Page Frame Address

The page frame address specifies the physical starting address of a page. Because pages are located on 4K boundaries, the low-order 12 bits are always zero. In a page directory, the page frame address is the address of a page table. In a second-level page table, the page frame address is the address of the page frame that contains the desired memory operand.

2.4.4.2 Present Bit

The P (present) bit indicates whether a page table entry can be used in address translation. P = 1 indicates that the entry can be used. When P = 0 in either level of page tables, the entry is not valid for address translation, and the rest of the entry is available for software use; none of the other bits in the entry is tested by the hardware. If P = 0 in either level of page tables when an attempt is made to use a page-table entry for address translation, the processor signals either a data-access fault or an instruction-access fault. In software systems that support paged virtual memory, the trap handler can bring the required page into physical memory.

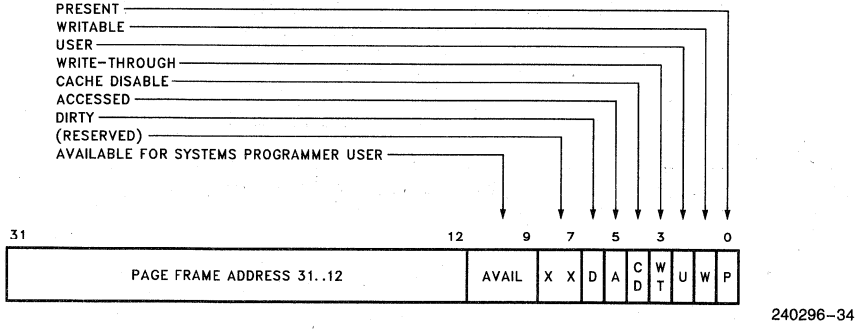
Note that there is no P bit for the page directory itself. The page directory may be not-present while the associated process is suspended, but the operating system must ensure that the page directory indicated by the **dirbase** image associated with the process is present in physical memory before the process is dispatched.

2.4.4.3 Writable and User Bits

The W (writable) and U (user) bits are used for page-level protection, which the i860 microprocessor performs at the same time as address translation. The concept of privilege for pages is implemented by assigning each page to one of two levels:

1. Supervisor level (U = 0)—for the operating system and other systems software and related data.
2. User level (U = 1)—for applications procedures and data.

The U bit of the **psr** indicates whether the i860 microprocessor is executing at user or supervisor level. The i860 microprocessor maintains the U bit of **psr** as follows:



NOTE:
X indicates Intel reserved. Do not use.

Figure 2.11. Format of a Page Table Entry

- The i860 microprocessor clears the **psr** U bit to indicate supervisor level when a trap occurs (including when the **trap** instruction causes the trap). The prior value of U is copied into PU.
- The i860 microprocessor copies the **psr** PU bit into the U bit when an indirect branch is executed and one of the trap bits is set. If PU was one, the i860 microprocessor enters user level.

With the U bit of **psr** and the W and U bits of the page table entries, the i860 microprocessor implements the following protection rules:

- When at user level, a read or write of a supervisor-level page causes a trap.
- When at user level, a write to a page whose W bit is clear causes a trap.
- When at user level, **st.c** to certain control registers is ignored.

When the i860 microprocessor is executing at supervisor level, all pages are addressable, but, when it is executing at user level, only pages that belong to the user-level are addressable.

When the i860 microprocessor is executing at supervisor level, all pages are readable. Whether a page is writable depends upon the write-protection mode controlled by WP of **epsr**:

WP = 0	All pages are writable.
WP = 1	A write to a page whose W bit is clear causes a trap.

When the i860 microprocessor is executing at user level, only pages that belong to user level and are marked writable are actually writable; pages that belong to supervisor level are neither readable nor writable from user level.

2.4.4.4 Write-Through Bit

The i860 microprocessor does not implement a write-through caching policy for the on-chip data cache; however, the WT (write-through) bit in the second-level page-table entry does determine internal caching policy. If WT is set in a PTE, on-chip caching of data from the corresponding page is inhibited. The i860 CPU may place pages having WT = 1 into the instruction cache. Future implementations of the i860 architecture may adhere to a write-through data caching policy. Therefore, they may cache pages having the WT bit of the PTE set. If WT is clear, the normal write-back policy is applied to data from the page in the on-chip caches. The WT bit of page directory entries is not referenced by the processor, but is **reserved**.

The WT bit is independent of the CD bit; therefore, data may be placed in a second-level coherent cache, but kept out of the on-chip caches.

2.4.4.5 Cache Disable Bit

If the CD (cache disable) bit in the second-level page-table entry is set, data from the associated page is not placed in instruction or data caches. Clearing CD permits the cache hardware to place data from the associated page into caches. The CD bit of page directory entries is not referenced by the processor, but is **reserved.**

To control external caches, the i860 microprocessor outputs on its PTB pin either the CD or WT bit. The PBM bit of **epsr** determines which bit is output.

2.4.4.6 Accessed and Dirty Bits

The A (accessed) and D (dirty) bits provide data about page usage in both levels of the page tables.

The i860 microprocessor sets the corresponding accessed bits in both levels of page tables before a read or write operation to a page. The processor tests the dirty bit in the second-level page table before a write to an address covered by that page table entry, and, under certain conditions, causes traps. The trap handler then has the opportunity to maintain appropriate values in the dirty bits. The dirty bit in directory entries is not tested by the i860 microprocessor. The precise algorithm for using these bits is specified in Section 2.4.5.

An operating system that supports paged virtual memory can use these bits to determine what pages to eliminate from physical memory when the demand for memory exceeds the physical memory available. The D and A bits in the PTE (page-table entry) are normally initialized to zero by the operating system. The processor sets the A bit when a page is accessed either by a read or write operation. When a data- or instruction-access fault occurs, the trap handler sets the D bit if an allowable write is being performed, then re-executes the instruction.

The operating system is responsible for coordinating its updates to the accessed and dirty bits with updates by the CPU and by other processors that may share the page tables. The i860 microprocessor automatically asserts the LOCK# signal while setting the A bit. If an A-bit of a PTE is found not set during a locked sequence (created by the **lock** instruction), a trap will occur and the processor will not update the A-bit.

2.4.4.7 Combining Protection of Both Levels of Page Tables

For any one page, the protection attributes of its page directory entry may differ from those of its page table entry. The i860 microprocessor computes the effective protection attributes for a page

by examining the protection attributes in both the directory and the page table. Table 2.6 shows the effective protection provided by the possible combinations of protection attributes.

2.4.5 ADDRESS TRANSLATION ALGORITHM

The algorithm below defines the translation of each virtual address to a physical address. Let DIR, PAGE, and OFFSET be the fields of the virtual address; let PFA1 and PFA2 be the page frame address fields of the first and second level page tables respectively; DTB is the page directory table base address stored in the **dirbase** register.

1. Read the PTE (page table entry) at the physical address formed by DTB:DIR:00.
2. If P in the PTE is zero, generate a data- or instruction-access fault.
3. If W in the PTE is zero, the operation is a write, and either the U-bit of the PSR is set or WP = 1, generate a data or instruction access fault.
4. If the U-bit in the PTE is zero and the U-bit in the **psr** is set, generate a data or instruction access fault.
5. If A in the PTE is zero, and if the TLB miss occurred while the bus was locked, generate a data or instruction access fault. (The trap allows software to set A to one and restart the sequence. This avoids ambiguity in determining what address corresponds to a locked semaphore for external bus hardware use.)
6. If A in the PTE is zero, and if the TLB miss occurred while the bus was not locked, assert LOCK#. Re-fetch and check the PTE, set A, and store the PTE. Deassert LOCK# during the store.
7. Locate the PTE at the physical address formed by PFA1:PAGE:00.
8. Perform the P, W, U, and A checks as in steps 2 through 6 with the second-level PTE.
9. If D in the PTE is clear and the operation is a write, generate a data or instruction access fault.
10. Form the physical address as PFA2:OFFSET.

The i860 microprocessor looks only in external memory for Page Directories and Page Tables, in the translation process. The data cache is not searched. Therefore, any code which modifies Page Directories or Page Tables must keep them out of the cache. The tables should be kept in non-cacheable memory, or flushed from the cache.

Table 2.6. Combining Directory and Page Protections

Page Directory Entry		Page Table Entry		Combined Protection		
				User Access	Supervisor Access	
U-bit	W-bit	U-bit	W-bit	WP = X	WP = 0	WP = 1
0	0	0	0	N	R/W	R
0	0	0	1	N	R/W	R
0	0	1	0	N	R/W	R
0	0	1	1	N	R/W	R
0	1	0	0	N	R/W	R
0	1	0	1	N	R/W	R/W
0	1	1	0	N	R/W	R
0	1	1	1	N	R/W	R/W
1	0	0	0	N	R/W	R
1	0	0	1	N	R/W	R
1	0	1	0	R	R/W	R
1	0	1	1	R	R/W	R
1	1	0	0	N	R/W	R
1	1	0	1	N	R/W	R/W
1	1	1	0	R	R/W	R
1	1	1	1	R/W	R/W	R/W

NOTES:
N = No access allowed
R = Read access only
R/W = Both reads and writes allowed
X = Don't care

The i860 microprocessor expects Page Directories and Page Tables to be in little endian format. The operating system must maintain these tables in little endian format by either setting BE = 0 when manipulating the tables or by complementing bit 2 of the address when loading or storing entries.

2.4.6 ADDRESS TRANSLATION FAULTS

The address translation fault is one instance of the data-access fault. The instruction causing the fault can be re-executed upon returning from the trap handler.

2.4.7 PAGE TRANSLATION CACHE

For greatest efficiency in address translation, the i860 microprocessor stores the most recently used page-table data in an on-chip cache called the TLB (translation lookaside buffer). Only if the necessary paging information is not in the cache must both levels of page tables be referenced.

2.5 Caching and Cache Flushing

The i860 microprocessor has the ability to cache instruction, data, and address-translation information in on-chip caches. Caching uses virtual-address tags. The effects of mapping two different virtual addresses in the same address space to the same physical address are undefined.

Instruction, data, and address-translation caching on the i860 microprocessor are not transparent. Because the data cache uses a write-back protocol, writes do not immediately update memory, and writes to memory by other bus devices do not update the cache. Changes to page tables do not automatically update the TLB, and changes to instructions do not automatically update the instruction cache. Under certain circumstances, such as I/O references, self-modifying code, page-table updates, or shared data in a multiprocessing system, it is necessary to bypass or to flush the caches. The i860 microprocessor provides the following methods for doing this:

- **Bypassing Instruction and Data Caches.** If deasserted during cache-miss processing, the KEN# pin disables instruction and data caching of the referenced data. If the CD bit of the associated second-level PTE is set, caching of data and instructions is disabled. The i860 CPU may place pages having WT = 1 into the instruction cache. Future implementations of the i860 architecture may adhere to a write-through data cache policy. Thus, they may cache pages having the WT bit of the PTE set. The value of the CD bit or the WT bit is output on the PTB pin for use by external caches.
- **Invalidating Instruction and Address-Translation Caches.** Storing to the **dirbase** register with the ITI bit set invalidates the contents of the instruction and address-translation caches. This bit should be set when modifying a page table, when modifying a page containing instructions, or when changing the DTB field of **dirbase** or the WP bit of the **epsr**. Note that in order to make the instruction or address-translation caches consistent with the data cache, the data cache must be flushed *before* invalidating the other caches.

  NOTE:
  The mapping of the page containing the currently executing instruction and the next six instructions should not be different in the new page tables when **st.c dirbase** changes DTB or activates ITI. The six instructions following the **st.c** should be **nop**s and should lie in the same page as the **st.c**.
- **Flushing the Data Cache.** The data cache is flushed by a software routine using the **flush** instruction. The data cache must be flushed prior to invalidating the instruction or address-translation caches (as controlled by the ITI bit of **dirbase**) or enabling or disabling address translation (via the ATE bit). The data cache does not need flushing if the program is modifying only the P, U, W, A, or D bits of a PTE (as long as the Page Frame Address is not changed and the PTE itself was not in the data cache.) The i860 CPU does not check these protection bits on cache line writeback. Thus, a trap handler can service a DAT for D-bit-zero by setting D = 1 and then ITI = 1. In the case of setting the P or A bits active, there is no need to invalidate or flush any caches because the processor does not load entries into the TLB that have P = 0 or A = 0. The i860 microprocessor searches only external memory for Page Directories and Page Tables in the translation process. The data cache is not searched. Therefore, Page Tables and Directories should be kept in non-cacheable memory, or flushed from the cache by any code which accesses them.

2.6 Instruction Set

Table 2.7 shows the complete set of instructions grouped by function within processing unit. Refer to Section 8 for an algorithmic definition of each instruction.

The architecture of the i860 microprocessor uses parallelism to increase the rate at which operations may be introduced into the unit. Parallelism in the i860 microprocessor is **not** transparent; rather, programmers have complete control over parallelism and therefore can achieve maximum performance for a variety of computational problems.

2.6.1 PIPELINED AND SCALAR OPERATIONS

One type of parallelism used within the floating-point unit is "pipelining". The pipelined architecture treats each operation as a series of more primitive operations (called "stages") that can be executed in parallel. Consider just the floating-point adder unit as an example. Let **A** represent the operation of the adder. Let the stages be represented by $\mathbf{A_1}$, $\mathbf{A_2}$, and $\mathbf{A_3}$. The stages are designed such that $\mathbf{A_{i+1}}$ for one adder instruction can execute in parallel with $\mathbf{A_i}$ for the next adder instruction. Furthermore, each $\mathbf{A_i}$ can be executed in just one clock. The pipelining within the multiplier and graphics units can be described similarly, except that the number of stages may be different.

Figure 2.12 illustrates three-stage pipelining as found in the floating-point adder (also in the floating-point multiplier when single-precision input operands are employed). The columns of the figure represent the three stages of the pipeline. Each stage holds intermediate results and also (when introduced into first stage by software) holds status information pertaining to those results. The figure assumes that the instruction stream consists of a series of consecutive floating-point instructions, all of one type (i.e. all adder instructions or all single-precision multiplier instructions). The instructions are represented as **i**, **i+1**, etc. The rows of the figure represent the states of the unit at successive clock cycles. Each time a pipelined operation is performed, the result of the last stage of the pipeline is stored in the destination register *fdest*, the pipeline is advanced one stage, and the input operands *fsrc1* and *fsrc2* are transferred to the first stage of the pipeline.

In the i860 microprocessor, the number of pipeline stages ranges from one to three. A pipelined operation with a three-stage pipeline stores the result of the third prior operation. A pipelined operation with a two-stage pipeline stores the result of the second prior operation. A pipelined operation with a one-stage pipeline stores the result of the prior operation.

There are four floating-point pipelines: one for the multiplier, one for the adder, one for the graphics unit, and one for floating-point loads. The adder pipeline has three stages. The number of stages in the multiplier pipeline depends on the precision of the source operands in the pipeline. Single precision has three stages and double precision has two stages. The graphics unit has one stage for all precisions. The load pipeline has three stages for all precisions.

Changing the FZ (flush zero), RM (rounding mode), or RR (result register) bits of **fsr** while there are results in either the multiplier or adder pipeline produces effects that are not defined.

2.6.1.1 Scalar Mode

In addition to the pipelined execution mode, the i860 microprocessor also can execute floating-point instructions in "scalar" mode. Most floating-point instructions have both pipelined and scalar variants, distinguished by a bit in the instruction encoding. In scalar mode, the floating-point unit does not start a new operation until the previous floating-point operation is completed. The scalar operation passes through all stages of its pipeline before a new operation is introduced, and the result is stored automatically. Scalar mode is used when the next operation depends on results from the previous few floating-point operations (or when the compiler or programmer does not want to deal with pipelining).

2.6.1.2 Pipelining Status Information

Result status information in the **fsr** consists of the AA, AI, AO, AU, and AE bits, in the case of the adder, and the MA, MI, MO, and MU bits, in the case of the multiplier. This information arrives at the **fsr** via the pipeline in one of two ways:

Table 2.7. Instruction Set

Core Unit	
Mnemonic	**Description**
Load and Store Instructions	
ld.x	Load integer
st.x	Store integer
fld.y	F-P load
pfld.z	Pipelined F-P load
fst.y	F-P store
pst.d	Pixel store
Register to Register Moves	
ixfr	Transfer integer to F-P register
Integer Arithmetic Instructions	
addu	Add unsigned
adds	Add signed
subu	Subtract unsigned
subs	Subtract signed
Shift Instructions	
shl	Shift left
shr	Shift right
shra	Shift right arithmetic
shrd	Shift right double
Logical Instructions	
and	Logical AND
andh	Logical AND high
andnot	Logical AND NOT
andnoth	Logical AND NOT high
or	Logical OR
orh	Logical OR high
xor	Logical exclusive OR
xorh	Logical exclusive OR high
Control-Transfer Instructions	
trap	Software trap
intovr	Software trap on integer overflow
br	Branch direct
bri	Branch indirect
bc	Branch on CC
bc.t	Branch on CC taken
bnc	Branch on not CC
bnc.t	Branch on not CC taken
bte	Branch if equal
btne	Branch if not equal
bla	Branch on LCC and add
call	Subroutine call
calli	Indirect subroutine call
System Control Instructions	
flush	Cache flush
ld.c	Load from control register
st.c	Store to control register
lock	Begin interlocked sequence
unlock	End interlocked sequence

Floating-Point Unit	
Mnemonic	**Description**
Register to Register Moves	
fxfr	Transfer F-P to integer register
F-P Multiplier Instruction	
fmul.p	F-P multiply
pfmul.p	Pipelined F-P multiply
pfmul3.dd	3-Stage pipelined F-P multiply
fmlow.p	F-P multiply low
frcp.p	F-P reciprocal
frsqr.p	F-P reciprocal square root
F-P Adder Instructions	
fadd.p	F-P add
pfadd.p	Pipelined F-P add
famov.r	F-P adder move
pfamov.r	Pipelined F-P adder move
fsub.p	F-P subtract
pfsub.p	Pipelined F-P subtract
pfgt.p	Pipelined F-P greater-than compare
pfeq.p	Pipelined F-P equal compare
fix.p	F-P to integer conversion
pfix.p	Pipelined F-P to integer conversion
ftrunc.p	F-P to integer truncation
pftrunc.p	Pipelined F-P to integer truncation
Dual-Operation Instructions	
pfam.p	Pipelined F-P add and multiply
pfsm.p	Pipelined F-P subtract and multiply
pfmam.p	Pipelined F-P multiply with add
pfmsm.p	Pipelined F-P multiply with subtract
Long Integer Instructions	
fisub.z	Long-integer subtract
pfisub.z	Pipelined long-integer subtract
fiadd.z	Long-integer add
pfiadd.z	Pipelined long-integer add
Graphics Instructions	
fzchks	16-bit Z-buffer check
pfzchks	Pipelined 16-bit Z-buffer check
fzchkl	32-bit Z-buffer check
pfzchkl	Pipelined 32-bit Z-buffer check
faddp	Add with pixel merge
pfaddp	Pipelined add with pixel merge
faddz	Add with Z merge
pfaddz	Pipelined add with Z merge
form	OR with MERGE register
pform	Pipelined OR with MERGE register

Assembler Pseudo-Operations	
Mnemonic	**Description**
mov	Integer register-register move
fmov.r	F-P reg-reg move
pfmov.r	Pipelined F-P reg-reg move
nop	Core no-operation
fnop	F-P no-operation
pfle.p	Pipelined F-P less-than or equal

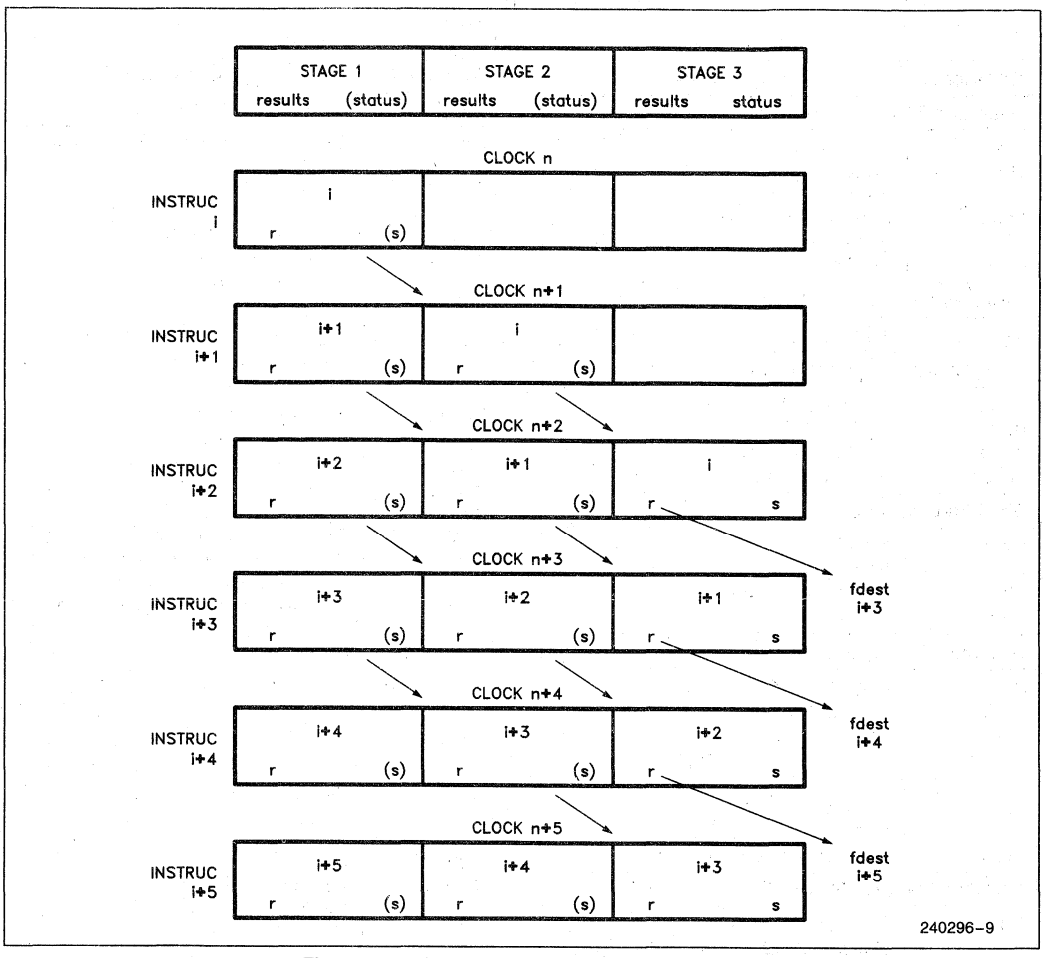


Figure 2.12. Pipelined Instruction Execution

1. It is calculated by the last stage of the pipeline. This is the normal case.
2. It is propagated from the first stage of the pipeline. This method is used when restoring the state of the pipeline after a preemption. When a store instruction updates the **fsr** and the value of the U bit in the word being written into the **fsr** is set, the store updates the result status bits in the first stage of both the adder and multiplier pipelines. When software changes the result-status bits of the first stage of a particular unit (multiplier or adder), the updated result-status bits are propagated one stage for each pipelined floating-point operation for that unit. In this case, each stage of the adder and multiplier pipelines holds its own copy of the relevant bits of the **fsr**. When they reach the last stage, they override the normal result-status bits computed from the last stage result.

At the next floating-point instruction (or at certain core instructions), after the result reaches the last stage, the i860 microprocessor traps if any of the status bits of the **fsr** indicate exceptions. Note that the instruction that creates the exceptional condition is not the instruction at which the trap occurs.

2.6.1.3 Precision in the Pipelines

In pipelined mode, when a floating-point operation is initiated, the result of an earlier pipelined floating-point operation is returned. The result precision of the current instruction applies to the operation being initiated. The precision of the value stored in *fdest* is that which was specified by the instruction that initiated that operation.

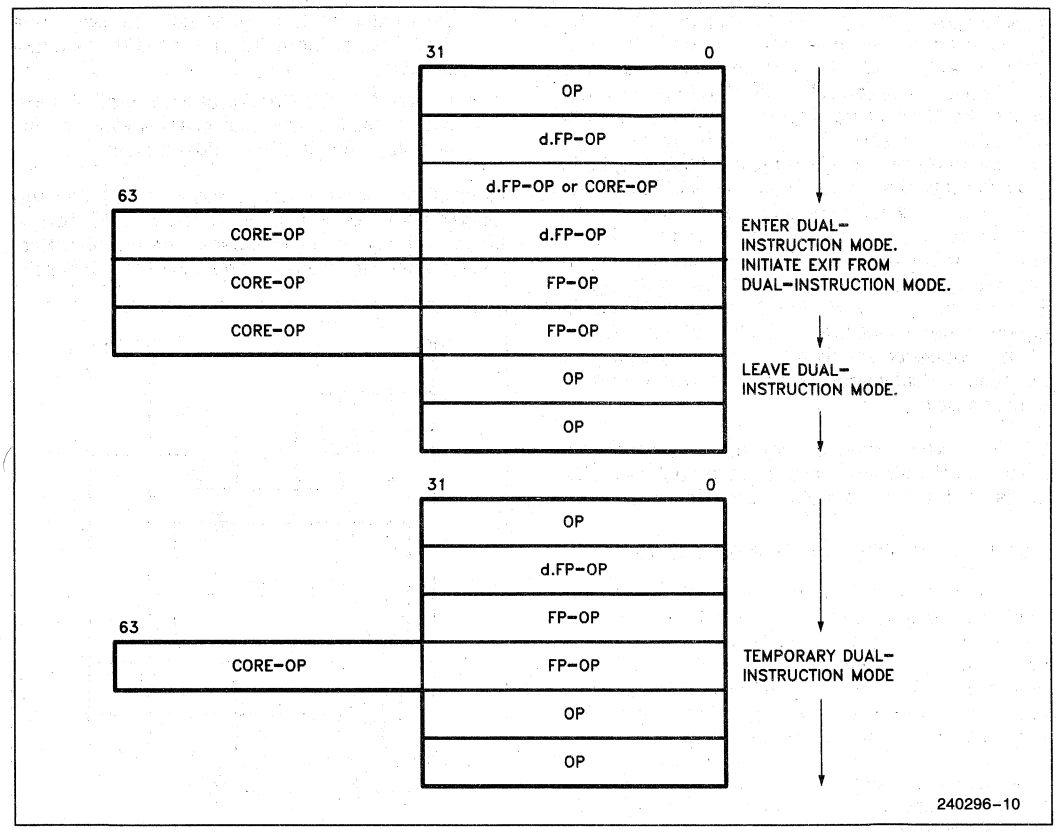


Figure 2.13. Dual-Instruction Mode Transitions

If *fdest* is the same as *fsrc1* or *fsrc2*, the value being stored in *fdest* is used as the input operand. In this case, the precision of *fdest* must be the same as the source precision.

The multiplier pipeline has two stages when the source operand is double-precision and three stages when the precision of the source operand is single. This means that a pipelined multiplier operation stores the result of the second previous multiplier operation for double-precision inputs and third previous for single-precision inputs (except when changing precisions).

2.6.1.4 Transition between Scalar and Pipelined Operations

When a scalar operation is executed, it passes through all stages of the pipeline; therefore, any unstored results in the affected pipeline are lost. To avoid losing information, the last pipelined operations before a scalar operation should be dummy pipelined operations that unload unstored results from the affected pipeline.

After a scalar operation, the values of all pipeline stages of the affected unit (except the last) are undefined. No spurious result-exception traps result when the undefined values are subsequently stored by pipelined operations; however, the values should not be referenced as source operands.

For best performance a scalar operation should not immediately precede a pipelined operation whose *fdest* is nonzero.

2.6.2 DUAL-INSTRUCTION MODE

Another form of parallelism results from the fact that the i860 microprocessor can execute both a floating-point and a core instruction simultaneously. Such parallel execution is called dual-instruction mode. When executing in dual-instruction mode, the instruction sequence consists of 64-bit aligned instructions with a floating-point instruction in the lower 32 bits and a core instruction in the upper 32 bits. Table 2.7 identifies which instructions are executed by the core unit and which by the floating-point unit.

Programmers specify dual-instruction mode either by including in the mnemonic of a floating-point instruction a **d.** prefix or by using the Assembler directives **.dual** . . . **.enddual**. Both of the specifications cause the D-bit of floating-point instructions to be set. If the i860 microprocessor is executing in single-instruction mode and encounters a floating-point instruction with the D-bit set, one more 32-bit instruction is executed before dual-mode execution begins. If the i860 microprocessor is executing in dual-instruction mode and a floating-point instruction is encountered with a clear D-bit, then one more pair of instructions is executed before resuming single-instruction mode. Figure 2.13 illustrates two variations of this sequence of events: one for extended sequences of dual-instructions and one for a single instruction pair.

When a 64-bit dual-instruction pair sequentially follows a delayed branch instruction in dual-instruction mode, both 32-bit instructions are executed.

2.6.3 DUAL-OPERATION INSTRUCTIONS

Special dual-operation floating-point instructions (add-and-multiply, subtract-and-multiply) use both the multiplier and adder units within the floating-point unit in parallel to efficiently execute such common tasks as evaluating systems of linear equations, performing the Fast Fourier Transform (FFT), and performing graphics transformations.

The instructions **pfam** *fsrc1*, *fsrc2*, *fdest* (add and multiply), **pfsm** *fsrc1*, *fsrc2*, *fdest* (subtract and multiply), **pfmam** *fscr1*, *fsrc2*, *fdest* (multiply and add), and **pfmsm** *fsrc1*, *fsrc2*, *fdest* (multiply and subtract) initiate both an adder operation and a multiplier operation. Six operands are required, but the instruction format specifies only three operands; therefore, there are special provisions for specifying the operands. These special provisions consist of:

- Three special registers (KR, KI, and T), that can store values from one dual-operation instruction and supply them as inputs to subsequent dual-operation instructions.
  1. The constant registers KR and KI can store the value of *fsrc1* and subsequently supply that value to the multiplier pipeline in place of *fsrc1*.
  2. The transfer register T can store the last stage result of the multiplier pipeline and subsequently supply that value to the adder pipeline in place of *fsrc1*.
- A four-bit data-path control field in the opcode (DPC) that specifies the operands and loading of the special registers.
  1. Operand-1 of the multiplier can be KR, KI, or *fsrc1*.
  2. Operand-2 of the multiplier can be *fsrc2* or the last stage result of the adder pipeline.
  3. Operand-1 of the adder can be *fsrc1*, the T-register, or the last stage result of the adder pipeline.
  4. Operand-2 of the adder can be *fsrc2*, the last stage result of the multiplier pipeline, or the last stage result of the adder pipeline.

Figure 2.14 shows all the possible data paths surrounding the adder and multiplier. A DPC field in these instructions select different data paths. Section 8 shows the various encodings of the DPC field.

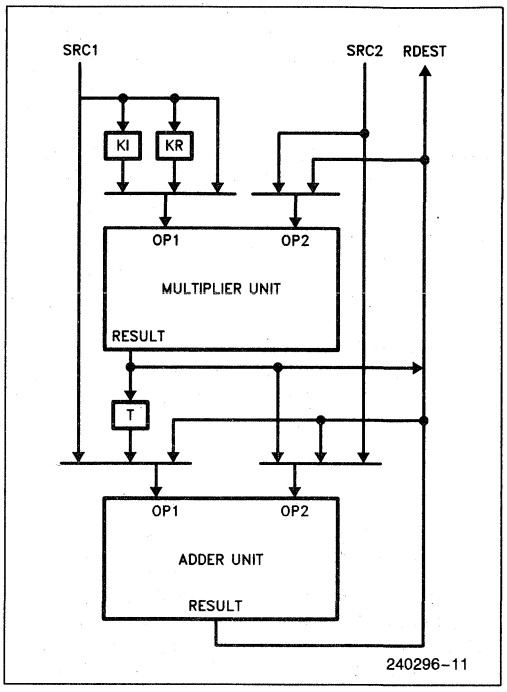


Figure 2.14. Dual-Operation Data Paths

Note that the mnemonics **pfam.p**, **pfsm.p**, **pfmam.p**, and **pfmsm.p** are never used as such in the assembly language; these mnemonics are used here to designate classes of related instructions. Each value of DPC has a unique mnemonic associated with it.

2.7 Addressing Modes

Data access is limited to load and store instructions. Memory addresses are computed from two fields of load and store instructions: *isrc1* and *isrc2*.

1. *isrc1* either contains the identifier of a 32-bit integer register or contains an immediate 16-bit address offset.
2. *isrc2* always specifies a register.

Table 2.8. Types of Traps

Type	Indication		Caused by	
	PSR,EPSR	FSR	Condition	Instruction
Instruction Fault	IT OF IL		Software traps Missing **unlock**	**trap, intovr** Any
Floating Point Fault	FT	SE AO, MO AU, MU AI, MI	Floating-point source exception Floating-point result exception overflow underflow inexact result	Any M- or A-unit except **fmlow** Any M- or A-unit except **fmlow, pfgt,** and **pfeq**. Reported on any F-P instruction plus **pst, fst,** and sometimes **fld, pfld, ixfr**
Instruction Access Fault	IAT		Address translation exception during instruction fetch	Any
Data Access Fault	DAT*		Load/store address translation exception Misaligned operand address Operand address matches **db** register	Any load/store Any load/store Any load/store
Interrupt	IN		External interrupt	
Reset	No trap bits set		Hardware RESET signal	

NOTES:
*These cases can be distinguished by examining the operand addresses.
The IL bit of the **epsr** must be checked by the trap handler to tell if the bus is currently in a locked sequence.

Because either *isrc1* or *isrc2* may be null (zero), a variety of useful addressing modes result:

offset + *register* — Useful for accessing fields within a record, where *register* points to the beginning of the record. Useful for accessing items in a stack frame, where *register* is **r3**, the register used for pointing to the beginning of the stack frame.

register + *register* — Useful for two-dimensional arrays or for array access within the stack frame.

register — Useful as the end result of any arbitrary address calculation.

offset — Absolute address into the first or last 32K of the logical address space.

In addition, the floating-point load and store instructions may select autoincrement addressing. In this mode *isrc2* is replaced by the sum of *isrc1* and *isrc2* after performing the load or store. This mode makes stepping through arrays more efficient, because it eliminates one address-calculation instruction.

2.8 Traps and Interrupts

Traps are caused by exceptional conditions detected in programs or by external interrupts. Traps cause interruption of normal program flow to execute a special program known as a trap handler. Traps are divided into the types shown in Table 2.8. Interrupts and traps start execution in single instruction mode at virtual address 0xFFFFFF00 in supervisor level (U = 0).

2.8.1 TRAP HANDLER INVOCATION

This section applies to traps other than reset. When a trap occurs, execution of the current instruction is aborted. The instruction is restartable. The processor takes the following steps while transferring control to the trap handler:

1. Copies U (user mode) of the **psr** into PU (previous U).
2. Copies IM (interrupt mode) into PIM (previous IM).
3. Sets U to zero (supervisor mode).
4. Sets IM to zero (interrupts disabled).
5. If the processor is in dual instruction mode, it sets DIM; otherwise it clears DIM.
6. If the processor is in single-instruction mode and the next instruction will be executed in dual-instruction mode or if the processor is in dual-instruction mode and the next instruction will be executed in single-instruction mode, DS is set; otherwise, it is cleared.
7. The appropriate trap type bits in **psr** are set (IT, IN, IAT, DAT, FT). Several bits may be set if the corresponding trap conditions occur simultaneously.

8. An address is placed in the fault instruction register (**fir**) to help locate the trapped instruction. In single-instruction mode, the address in **fir** is the address of the trapped instruction itself. In dual-instruction mode, the address in **fir** is that of the floating-point half of the dual instruction. If an instruction or data access fault occurred, the associated core instruction is the high-order half of the dual instruction (**fir** + 4). In dual-instruction mode, when a data access fault occurs in the absence of other trap conditions, the floating-point half of the dual instruction will already have been executed.

The processor begins executing the trap handler by transferring execution to virtual address 0xFFFFFF00. The trap handler begins execution in single-instruction mode. The trap handler must examine the trap-type bits in **psr** (IT, IN, IAT, DAT, FT) to determine the cause or causes of the trap.

2.8.2 INSTRUCTION FAULT

This fault is caused by any of the following conditions. In all cases the processor sets the IT bit before entering the trap handler.

1. By the **trap** instruction. When **trap** is executed in dual-instruction mode, the floating-point companion of the **trap** instruction is not executed before the trap is taken.
2. By the **intovr** instruction. The trap occurs only if OF in **epsr** is set when **intovr** is executed. The trap handler should clear OF before returning. When **intovr** causes a trap in dual-instruction mode, the floating-point companion of the **intovr** instruction is completely executed before the trap is taken.
3. By violation of lock/unlock protocol, explained below. (Note that **trap** and **intovr** should not be used within a locked sequence; otherwise, it would be difficult to distinguish between this and the prior cases.)

   The lock protocol requires the following sequence of activities:

   1. **lock**
   2. Any load or store instruction that misses the cache
   3. **unlock**
   4. Any load or store instruction (regardless of whether it misses the cache)

There may be other instructions between any of these steps. The bus is locked after step 2, and remains locked until step 4. Step 4 must follow step 1 by 30 instructions or less, otherwise the instruction trap occurs. In case of a trap, IL is also set. If the load or store instruction in step 2 hits the cache, the sequence is legal, but the bus is not locked.

2.8.3 FLOATING-POINT FAULT

The floating-point fault is reported on floating-point instructions, **pst**, **fst**, and sometimes **fld**, **pfld**, **ixfr**. The floating-point faults of the i860 microprocessor support the floating-point exceptions defined by the IEEE standard as well as some other useful classes of exceptions. The i860 microprocessor divides these into two classes: source exceptions and result exceptions. The numerics library supplied by Intel provides the IEEE standard default handling for all these exceptions.

2.8.3.1 Source Exception Faults

When used as inputs to the multiplier or adder, all exceptional operands, including infinities, denormalized numbers and NaNs, cause a floating-point fault and set SE in the **fsr**. Source exceptions are reported on the instruction that initiates the operation. For pipelined operations, the pipeline is not advanced.

The SE value is undefined for faults on **fld**, **pfld**, **fst**, **pst**, and **ixfr** instructions when in single-instruction mode or when in dual-instruction mode and the companion instruction is not a multiplier or adder operation.

2.8.3.2 Result Exception Faults

The class of result exceptions includes any of the following conditions:

- **Overflow.** The absolute value of the rounded true result would exceed the largest positive finite number in the destination format.
- **Underflow** (when FZ is clear). The absolute value of the rounded true result would be smaller than the smallest positive finite number in the destination format.
- **Inexact result** (when TI is set). The result is not exactly representable in the destination format. For example, the fraction 1/3 cannot be precisely represented in binary form. This exception occurs frequently and indicates that some (generally acceptable) accuracy has been lost.

The point at which a result exception is reported depends upon whether pipelined operations are being used:

- **Scalar (nonpipelined) operations.** Result exceptions are reported on the next floating-point, **fst.x**, or **pst.x** (and sometimes **fld**, **pfld**, **ixfr**) instruction after the scalar operation. When a trap occurs, the last stage of the affected unit contains the result of the scalar operation.
- **Pipelined operations.** Result exceptions are reported when the result is in the last stage and the next floating-point, **fst.x** or **pst.x** (and sometimes **fld**, **pfld**, **ixfr**) instruction is executed. When a trap occurs, the pipeline is not advanced, and the last stage results (that caused the trap) remain unchanged.

When no trap occurs (either because FTE is clear or because no exception occurred), the pipeline is advanced normally by the new floating-point operation.

The result-status bits of the affected unit are undefined until the point that result exceptions are reported. At this point, the last stage result-status bits (bits 29..22 and 16..9 of the **fsr**) reflect the values in the last stages of both the adder and multiplier. For example, if the last stage result in the multiplier has overflowed and a pipelined floating-point **pfadd** is started, a trap occurs and MO is set.

For scalar operations, the RR bits of **fsr** specify the register in which the result was stored. RR is updated when the scalar instruction is initiated. The trap, however, occurs on a subsequent instruction. Programmers must prevent intervening stores to **fsr** from modifying the RR bits. Prevention may take one of the following forms:

- Before any store to **fsr** when a result exception may be pending, execute a dummy floating-point operation to trigger the result-exception trap.
- Always read from **fsr** before storing to it, and mask updates so that the RR bits are not changed.

For pipelined operations, RR is cleared and the result is in the last stage of the pipeline of the appropriate unit. The trap handler must flush the pipeline, saving the results and the status bits.

In either pipelined or scalar mode, the trap handler must then compute the trapping result. In either case, the result has the same fraction as the true result and has an exponent which is the low-order bits of the true result. The trap handler can inspect the result, compute the result appropriate for that instruction (a NaN or an infinity, for example), and store the correct result. The result is either stored in the register specified by RR (if nonzero) or (if RR = 0) the trap handler must reload the pipeline with the saved results and status bits.

Result exceptions may be reported for both the adder and multiplier units at the same time. In this case, the trap handler should fix up the last stage of both pipelines.

2.8.4 INSTRUCTION ACCESS FAULT

This trap occurs during address translation for instruction fetches in any of these cases:

- The address fetched is in a page whose P (present) bit in the page table is clear (not present).
- The address fetched is in a supervisor mode page, but the processor is in user mode.
- The address fetched is in a page whose PTE has A = 0, and the access occurs during a locked sequence (i.e., between **lock** and **unlock**).

Note that several instructions are fetched at one time, either due to instruction prefetching or to instruction caching. Therefore, a trap handler can change from supervisor to user mode and continue to execute instructions fetched from a supervisor page. An instruction access trap occurs only when the next group of instructions is fetched from a supervisor page (up to eight instructions later). If, in the meantime, the handler branches to a user page, no instruction access trap occurs. No protection violation results, because the processor does not permit data accesses to supervisor pages while running in user mode.

2.8.5 DATA ACCESS FAULT

This trap results from an abnormal condition detected during data operand fetch or store. Such an exception can be due only to one of the following causes:

- An attempt is being made to write to a page whose D (Dirty) bit is clear.
- A memory operand is misaligned (is not located at an address that is a multiple of the length of the data).
- The address stored in the **db** register is equal to one of the addresses spanned by the operand.
- The operand is in a not-present page.
- An attempt is being made from user level to write to a read-only page or to access a supervisor-level page.
- The operand was in a page whose PTE had A = 0, and the access occurred during a locked sequence. (i.e., between **lock** and **unlock**.)
- Write protection (determined by **epsr** bit WP = 1) is violated in supervisor mode.

2.8.6 INTERRUPT TRAP

An interrupt is an event that is signaled from an external source. If the processor is executing with interrupts enabled (IM set in the **psr**), the processor sets the interrupt bit IN in the **psr**, and generates an interrupt trap. Vectored interrupts are implemented by interrupt controllers and software.

2.8.7 RESET TRAP

When the i860 microprocessor is reset, execution begins in single-instruction mode at physical address 0xFFFFFF00. This is the same address as for other traps. The reset trap can be distinguished from other traps by the fact that no trap bits are set. The instruction cache is flushed. The bits DPS, BL, and ATE in **dirbase** are cleared. CS8 is initialized by the value at the INT pin at the end of reset. The read-only fields of the **espr** are set to identify the processor, while the IL, WP, and PBM bits are cleared. The

bits U, IM, BR, and BW in **psr** are cleared, as are the trap bits FT, DAT, IAT, IN, and IT. All other bits of **psr** and all other register contents are **undefined.**

Refer to Table 2.9 for a summary of these initial settings.

Table 2.9. Register and Cache Values after Reset

Registers	Initial Value
Integer Registers	*Undefined*
Floating-Point Registers	*Undefined*
psr	U, IM, BR, BW, FT, DAT, IAT, IN, IT = 0; others are *undefined*
epsr	IL, WP, PBM, BE = 0; Processor Type, Stepping Number, DCS are read only; others are *undefined*
db	*Undefined*
dirbase	DPS, BL, ATE = 0; others are *undefined*
fir	*Undefined*
fsr	*Undefined*
KR, KI, T, MERGE	*Undefined*
Caches	**Initial Value**
Instruction Cache	Flushed
Data Cache	*Undefined*
TLB	Flushed

The software must ensure that the data cache is flushed and control registers are properly initialized before performing operations that depend on the values of the cache or registers. The data cache has no "validity" bits, so memory accesses before the flush may result in false data cache hits.

Reset code must initialize the floating-point pipeline state to zero with floating-point traps disabled to ensure that no spurious floating-point traps are generated.

After a RESET the i860 microprocessor starts execution at supervisor level (U=0). Before branching to the first user-level instruction, the RESET trap handler or subsequent initialization code has to set PU and a trap bit so that an indirect branch instruction will copy PU to U, thereby changing to user level.

2.9 Debugging

The i860 microprocessor supports debugging with both data and instruction breakpoints. The features of the i860 architecture that support debugging include:

- **db** (data breakpoint register) which permits specification of a data addresses that the i860 microprocessor will monitor.
- BR (break read) and BW (break write) bits of the **psr**, which enable trapping of either reads or writes (respectively) to the address in **db.**
- DAT (data access trap) bit of the **psr**, which allows the trap handler to determine when a data breakpoint was the cause of the trap.
- **trap** instruction that can be used to set breakpoints in code. Any number of code breakpoints can be set. The values of the *isrc1* and *isrc2* fields help identify which breakpoint has occurred.
- IT (instruction trap) bit of the **psr**, which allows the trap handler to determine when a **trap** instruction was the cause of the trap.

3.0 HARDWARE INTERFACE

In the following description of hardware interface, the # symbol at the end of a signal name indicates that the active or asserted state occurs when the signal is at a low voltage. When no # is present after the signal name, the signal is asserted when at the high voltage level.

3.1 Signal Description

Table 3.1 identifies functional groupings of the pins, lists every pin by its identifier, gives a brief description of its function, and lists some of its characteristics. All output pins are tristate, except HLDA and BREQ. All inputs are synchronous, except HOLD and INT.

3.1.1 CLOCK (CLK)

The CLK input determines execution rate and timing of the i860 microprocessor. Timing of other signals is specified relative to the rising edge of this signal. The i860 microprocessor can utilize a clock rate of 33.3 MHz or 40 MHz. The internal operating frequency is the same as the external clock.

3.1.2 SYSTEM RESET (RESET)

Asserting RESET for at least 16 CLK periods causes initialization of the i860 microprocessor. Refer to section 3.2 "Initialization" for more details related to RESET.

3.1.3 BUS HOLD (HOLD) AND BUS HOLD ACKNOWLEDGE (HLDA)

These pins are used for i860 microprocessor bus arbitration. At some clock after the HOLD signal is asserted, the i860 microprocessor releases control

Table 3.1. Pin Summary

Pin Name	Function	Active State	Input/ Output
Execution Control Pins			
CLK	CLocK		I
RESET	System reset	High	I
HOLD	Bus hold	High	I
HLDA	Bus hold acknowledge	High	O
BREQ	Bus request	High	O
INT/CS8	Interrupt, code-size	High	I
Bus Interface Pins			
A31–A3	Address bus	High	O
BE7#–BE0#	Byte Enables	Low	O
D63–D0	Data bus	High	I/O
LOCK#	Bus lock	Low	O
W/R#	Write/Read bus cycle	High/Low	O
NENE#	NExt NEar	Low	O
NA#	Next Address request	Low	I
READY#	Transfer Acknowledge	Low	I
ADS#	ADdress Status	Low	O
Cache Interface Pins			
KEN#	Cache ENable	Low	I
PTB	Page Table Bit	High	O
Testability Pins			
SHI	Boundary Scan Shift Input	High	I
BSCN	Boundary Scan Enable	High	I
SCAN	Shift Scan Path	High	I
Intel-Reserved Configuration Pins			
CC1–CC0	Configuration	High	I
Power and Ground Pins			
V_{CC}	System power		
V_{SS}	System ground		

A # after a pin name indicates that the signal is active when at the low voltage level.

of the local bus and puts all bus interface outputs (except BREQ and HLDA) into a floating state, then asserts HLDA—all during the same clock period. It maintains this state until HOLD is deasserted. Instruction execution stops only if required instructions or data cannot be read from the on-chip instruction and data caches.

The time required to acknowledge a hold request is one clock plus the number of clocks needed to finish any outstanding bus cycles. HOLD is recognized even while RESET or LOCK# are asserted.

When leaving a bus hold, the i860 microprocessor deactivates HLDA and, in the same clock period, initiates a pending bus cycle, if any.

Hold is an asynchronous input.

3.1.4 BUS REQUEST (BREQ)

This signal is asserted when the i860 microprocessor has a pending memory request, even when HLDA is asserted. This allows an external bus arbiter to implement an "on demand only" policy for granting the bus to the i860 microprocessor. BREQ is asserted the clock after the i860 microprocessor realizes an internal request for the bus. In normal operation, BREQ goes low the clock after ADS# goes low for the final pending bus cycle. (Refer to Figure 4.10 for timing information.) During data or instuction cache fills, however, BREQ may be deasserted for one or more clocks, due to cache and TLB logic.

3.1.5 INTERRUPT/CODE-SIZE (INT/CS8)

This input allows interruption of the current instruction stream. If interrupts are enabled (IM set in **psr**) when INT is asserted, the i860 microprocessor fetches the next instruction from address

0xFFFFFF00. To assure that an interrupt is recognized, INT should remain asserted until the software acknowledges the interrupt (by writing, for example, to a memory-mapped port of an interrupt controller). When the bus is not locked, the maximum time between the assertion of INT and the execution of the first instruction of the trap handler is ten clocks, plus the time for four sets of four pipelined read cycles and two sets of four pipelined writes (instruction- and data-cache misses and write-back cycles to update memory), plus the time for twenty nonpipelined read cycles (six TLB misses, with eight refetches when the A-bit is zero), plus the time for eight nonpipelined writes (updates to the A-bit).

If the bus is locked from a **lock** instruction, the INT pin is ignored and the INT bit of **epsr** is always zero. The **lock** instruction can only assert LOCK# for 30–33 instructions before trapping.

If INT is asserted during the clock before the falling edge of RESET, the eight-bit code-size mode is selected. For more about this mode, refer to section 3.2 "Initialization".

INT is an asynchronous input.

3.1.6 ADDRESS PINS (A31–A3) AND BYTE ENABLES (BE7#–BE0#)

The 29-bit address bus (A31–A3) identifies addresses to a 64-bit location. Separate byte-enable signals (BE7#–BE0#) identify which bytes should be accessed within the 64-bit location. In all noncacheable read cycles (KEN# deasserted), the byte enables match the length and address of the requested data. Cacheable read cycles (KEN# asserted), however, result in four 64-bit memory cycles to fill an entire 32-byte cache line. The BE*n*# pins activated are those that represent the operand of the load instruction that caused the line fill, and these same BE*n*# pins remain activated for all four cycles of the line fill. All 64 bits must be returned for each cycle without regard for the BE*n*# signals. In all write cycles (noncacheable writes as well as cache line write-backs) the BE*n*# signals indicate the bytes that must be written.

Instruction fetches (W/R# is low) are distinguished from data accesses by the unique combinations of BE7#–BE0# defined in Table 3.2. For an eight-bit code fetch in eight-bit code-size (CS8) mode, BE2#–BE0# are redefined to be A2–A0 of the address. In this case BE7#–BE3# form the code shown in Table 3.2 that identifies an instruction fetch. The A2 in the table does not represent a physical pin, just a conceptual internal address line value. The "x"under A2 for CS8 mode means "not applicable", or "don't care". All other combinations of byte enables indicate data accesses.

The address and byte-enable pins are driven until either NA# or READY# is asserted.

3.1.7 DATA PINS (D63–D0)

The bus interface has 64 bidirectional data pins (D63–D0) to transfer data in eight- to 64-bit quantities. Pins D7–D0 transfer the least significant byte; pins D63–D56 transfer the most significant byte.

In read bus cycles, all 64 bits of the data bus are latched, even in CS8-mode instruction fetches when only the low-order eight bits are used.

In write bus cycles, the point at which data is driven onto the bus depends on the type of the preceding cycle. If there was no preceding cycle (i.e. the bus was idle), data is driven with the address. If the preceding cycle was a write, data is driven as soon as READY# is returned from the previous cycle. If the preceding cycle was a read, data is driven one clock after READY# is returned from the previous cycle, thereby allowing time for the bus to be turned around. Data continues to be driven until READY# for the current cycle is returned.

3.1.8 BUS LOCK (LOCK#)

This signal is used to provide atomic (indivisible) read-modify-write sequences in multiprocessor systems. A multiprocessor bus arbiter must permit only one processor a locked access to the address which is on the bus when LOCK# first activates. The system must maintain the lock of that location until LOCK# deactivates.

The i860 microprocessor coordinates the external LOCK# signal with the software-controlled BL bit of the **dirbase** register. Programmers do not have to be concerned about the fact that bus activity is not always synchronous with instruction execution. LOCK# is asserted with ADS# for the address operand of the first load or store instruction executed after the BL bit is set by the **lock** instruction. Pending bus cycles are locked according to the value of the BL bit when the instruction was executed. Even if the BL bit is changed between the time that an instruction generates an internal bus request and the time that the cycle appears on the bus, the i860 microprocessor still asserts LOCK# for that bus cycle.

If ADS# is active when LOCK# deactivates, then that request should complete before the hardware relinquishes the lock. If ADS# is not active, the locking of the location can immediately end when LOCK# deactivates. Of course the simplest arbitration hardware can just lock the entire bus against all other accesses during LOCK# assertion.

Table 3.2. Identifying Instruction Fetches

Code Fetch	A2	BE7#	BE6#	BE5#	BE4#	BE3#	BE2#	BE1#	BE0#
Normal (Non-CS8)	0	1	1	1	1	1	0	1	0
Normal (Non-CS8)	1	1	0	1	0	1	1	1	1
CS8 Mode	x	1	0	1	0	1	Low-order address bits		

When the BL bit is deasserted with the **unlock** instruction, LOCK# is deasserted with the next load or store but after any pending bus cycles. Between locked sequences, at least one cycle of no LOCK# is guaranteed by the behavior of the **unlock** instruction. LOCK# deassertion may occur independently of ADS# for the case of a trap or a cache hit after **unlock**.

The i860 microprocessor also asserts LOCK# during TLB miss processing for updates of the accessed bit in page-table entries. The maximum time that LOCK# can be asserted in this case is five clocks plus the time required to perform a read-modify-write sequence. Instruction fetches do not alter the LOCK# pin.

Between **lock** and **unlock** instructions, the INT pin is ignored and the INT bit of **epsr** is zero when read by **ld.c epsr**. The time that interrupts are disabled is limited by the lock protocol outlined in Section 2.8.2.

3.1.9 WRITE/READ BUS CYCLE (W/R#)

This pin specifies whether a bus cycle is a read (LOW) or write (HIGH) cycle. It is driven until either NA# or READY# is asserted.

3.1.10 NEXT NEAR (NENE#)

This signal allows higher-speed reads and writes in the case of consecutive reads and writes that access static column or page-mode DRAMs. The i860 microprocessor asserts NENE# when the current address is in the same DRAM page as the previous bus cycle. The i860 microprocessor determines the DRAM page size by inspecting the DPS field in the **dirbase** register. The page size can range from 2^9 to 2^{16} 64-bit words, supporting DRAM sizes from 256K × 1, 256K × 4, and up. NENE# is never asserted on the next bus cycle after HLDA is deasserted.

3.1.11 NEXT ADDRESS REQUEST (NA#)

NA# makes address pipelining possible. The system asserts NA# for at least one clock to indicate that it is ready to accept the next address from the i860 microprocessor. NA# may be asserted before the current cycle ends. (If the system does not implement pipelining, NA# does not have to be activated.) The i860 microprocessor samples NA# every clock, starting one clock after the prior activation of ADS#. When NA# is active, the i860 microprocessor is free to drive address and bus-cycle definition for the next pending bus cycle. The i860 microprocessor remembers that NA# was asserted when no internal request is pending; therefore, NA# can be deactivated after the next rising edge of the CLK signal. Up to three bus cycles can be outstanding simultaneously.

3.1.12 TRANSFER ACKNOWLEDGE (READY#)

The system must assert the READY# signal during read cycles when valid data is on the data pins and during write cycles when the system has accepted data from the data pins. READY# must be asserted for at least one clock. Sampling of READY# begins in the clock after an ADS# or in the second clock after a prior READY#.

3.1.13 ADDRESS STATUS (ADS#)

The i860 microprocessor asserts ADS# during the first clock of each bus cycle to identify the clock period during which it begins to assert outputs on the address bus. This signal is held active for one clock.

3.1.14 CACHE ENABLE (KEN#)

The i860 microprocessor samples KEN# to determine whether the data being read for the current cache-miss cycle is to be cached. This pin is internally NORed with the CD and WT bits to control cacheability on a page by page basis (refer to Table 3.3).

If the address is one that is permitted to be in the cache, KEN# must be continuously asserted during the sampling period starting from the second rising clock edge after ADS# is asserted, through the clock NA# or READY# is asserted. The entire 64 bits of the data bus will be used for the read, regardless of the state of the byte-enable pins. Three additional 64-bit bus cycles will be generated to fill the rest of the 32-byte cache block.

If KEN# is found deasserted at any clock from the clock after ADS# through the clock of the **first** NA# or READY#, the data being read will not be cached and two scenarios can occur: 1) if the cycle is due to data-cache miss, no subsequent cache-fill cycles will be generated; 2) if the cycle is due to an instruction-cache miss, additional cycle(s) will be generated until the address reaches a 32-byte boundary. To avoid caching a line, external hardware must deassert KEN# during or before the first NA# or READY#.

3.1.15 PAGE TABLE BIT (PTB)

Depending on the setting of the PBM (page-table bit mode) bit of the **epsr**, the PTB reflects the value of either the CD (cache disable) bit or the WT (write through) bit of the page-table entry used for the current cycle. When paging is disabled, PTB remains inactive.

Table 3.3. Cacheability based on KEN# and CD OR WT

CD OR WT	KEN#	Meaning
0	0	Cacheable access
0	1	Noncacheable access
1	0	Noncacheable page
1	1	Noncacheable page

3.1.16 BOUNDARY SCAN SHIFT INPUT (SHI)

This pin is used with the testability features. Refer to section 3.3.

3.1.17 BOUNDARY SCAN ENABLE (BSCN)

This pin is used with the testability features. Refer to section 3.3.

3.1.18 SHIFT SCAN PATH (SCAN)

This pin is used with the testability features. Refer to section 3.3.

3.1.19 CONFIGURATION (CC1-CC0)

These two pins are reserved by Intel. Strap both pins LOW.

3.1.20 SYSTEM POWER (V_{CC}) AND GROUND (V_{SS})

The i860 microprocessor has 48 pins for power and ground. All pins must be connected to the appropriate low-inductance power and ground signals in the system.

3.2 Initialization

Initialization of the i860 microprocessor is caused by assertion of the RESET signal for at least 16 clocks. Table 3.4 shows the status of output pins during the time that RESET is asserted. Note that HOLD requests are honored during RESET and that the status of output pins depends on whether a HOLD request is being acknowledged.

Table 3.4. Output Pin Status during Reset

Pin Name	Pin Value	
	HOLD Not Acknowledged	HOLD Acknowledged
ADS#, LOCK#	HIGH	Tri-State OFF
W/R#, PTB	LOW	Tri-State OFF
BREQ	LOW	LOW
HLDA	LOW	HIGH
D63–D0	Tri-State OFF	Tri-State OFF
A31–A3, BE7#–BE0#, NENE#	Undefined	Tri-State OFF

After a reset, the i860 microprocessor begins executing at physical address 0xFFFFFF00. The program-visible state of the i860 microprocessor after reset is detailed in section 2.8.7.

Eight-bit code-size mode is selected when INT/CS8 is asserted during the clock before the falling edge of RESET. While in eight-bit code-size mode, instruction cache misses are byte reads (transferred on D7-D0 of the data bus) instead of eight-byte reads. This allows the i860 microprocessor to be bootstrapped from an eight-bit EPROM. For these code reads, byte enables BE2#–BE0# are redefined to be the low order three bits of the address, so that a complete byte address is available. These reads update the instruction cache if KEN# is asserted (refer to section 3.1.14) and are not pipelined even if NA# is asserted. While in this mode, instructions must reside in an eight-bit wide memory, while data must reside in a separate 64-bit wide memory. After the code has been loaded into 64-bit memory, initialization code can initiate 64-bit code fetches by clearing the CS8 bit of the **dirbase** register (refer to section 2). Once eight-bit code-size mode is disabled by software, it cannot be reenabled except by resetting the i860 microprocessor.

3.3 Testability

The i860 microprocessor has a *boundary scan mode* that may be used in component- or board-level testing to test the signal traces leading to and from the i860 microprocessor. Boundary scan mode provides a simple serial interface that makes it possible to test all signal traces with only a few probes. Probes need be connected only to CLK, BSCN, SCAN, SHI, BREQ, RESET, and HOLD.

The pins BSCN and SCAN control the boundary scan mode (refer to Table 3.5). When BSCN is asserted, the i860 microprocessor enters boundary scan mode on the next rising clock edge. Boundary scan mode can be activated even while RESET is active. When BSCN is deasserted while in boundary scan mode, the i860 microprocessor leaves boundary scan mode on the next rising clock edge. After leaving boundary scan mode, the internal state is undefined; therefore, RESET should be asserted.

Table 3.5. Test Mode Selection

BSCN	SCAN	Testability Mode
LO	LO	No testability mode selected
LO	HI	(Reserved for Intel)
HI	LO	Boundary scan mode, normal
HI	HI	Boundary scan mode, shift SHI as input; BREQ as output

For testing purposes, each signal pin has associated with it an internal latch. Table 3.6 indentifies these latches by name and classifies them as input, output, or control. The input and output latches carry the name of the corresponding pins.

Table 3.6. Test Mode Latches

Input Latch	Output Latch	Associated Control Latch
SHI		
BSCN		
SCAN		
RESET		
D0–D63	D0–D63	DATAt
CC1-CC0		
	A31–A3	ADDRt
	NENE#	NENEt
	PTB#	PTBt
	W/R#	W/Rt
	ADS#	ADSt
	HLDA	
	LOCK#	LOCKt
READY#		
KEN#		
NA#		
INT/CS8		
HOLD		
	BE7#–BE0#	BEt
	BREQ	

Within boundary scan mode the i860 microprocessor operates in one of two submodes: normal mode or shift mode, depending on the value of the SCAN input. A typical test sequence is . . .

1. Enter shift mode to assign values to the latches that correspond with the pins.
2. Enter normal mode. In normal mode the i860 microprocessor transfers the latched values to the output pins and latches the values that are being driven onto the input pins.
3. Reenter shift mode to read the new values of the input pins.

3.3.1 NORMAL MODE

When SCAN is deasserted, the normal mode is selected. For each input pin (RESET, HOLD, INT/CS8, NA#, READY#, KEN#, SHI, BSCN, SCAN, CC1, and CC0), the corresponding latch is loaded with the value that is being driven onto the pin.

The tristate output pins (A31–A3, BE7#–BE0#, W/R#, NENE#, ADS#, LOCK#, and PTB) are enabled by the control latches ADDRt (for A31–A3), BEt, W/Rt, NENEt, ADSt, LOCKt, and PTBt. If a control latch is set, the corresponding output latches drive their output pins; otherwise the pins are not driven.

The I/O pins (D63–D0) are enabled by the control latch DATAt, which is similar to the other control latches. In addition, when DATAt is not set, the data pins are treated as input pins and their values are latched.

3.3.2 SHIFT MODE

When SCAN is asserted, the shift mode is selected. In shift mode, the pins are organized into a *boundary scan chain*. The scan chain is configured as a shift register that is shifted on the rising edge of CLK. The SHI pin is connected to the input of one end of the boundary scan chain. The value of the most significant bit of the scan chain is output on the BREQ pin. To avoid glitches while the values are being shifted along the chain, the tester should assert both the RESET and HOLD pins. Then all tristate outputs are disabled. The order of the pins within the chain is shown in Figure 3.1.

A tester causes entry into this mode for one of two purposes:

1. To assign values to output latches to be driven onto output pins upon subsequent entry into normal mode.
2. To read the values of input pins previously latched in normal mode.

4.0 BUS OPERATION

A bus cycle begins when ADS# is activated and ends when READY# is sampled active. READY# is sampled one clock after assertion of ADS# and thereafter until it becomes active. New cycles can start as often as every other clock until three cycles are outstanding. A bus cycle is considered outstanding as long as READY# has not been asserted to terminate that cycle. After READY# becomes active, it is not sampled again for the following (outstanding) cycle until the second clock after the one during which it became active. READY# is assumed to be inactive when it is not sampled.

With regard to how a bus cycle is generated by the i860 microprocessor, there are two types of cycles: pipelined and nonpipelined. Both types of cycles can be either read or write cycles. A pipelined cycle is one that starts while one or two other bus cycles are outstanding. A nonpipelined cycle is one that starts when no other bus cycles are outstanding.

4.1 Pipelining

A **m-n** read or write cycle is a cycle with a total cycle time of **m** clocks and a cycle-to-cycle time of **n** clocks (**m** ≥ **n**). Total cycle time extends from the clock in which ADS# is activated to the clock in which READY# becomes active, whereas cycle-to-cycle time extends from the time that READY# is sampled active for the previous cycle to the time that it is sampled active again for the current cycle. When **m** = **n**, a nonpipelined cycle is implied; **m** > **n** implies a pipelined cycle.

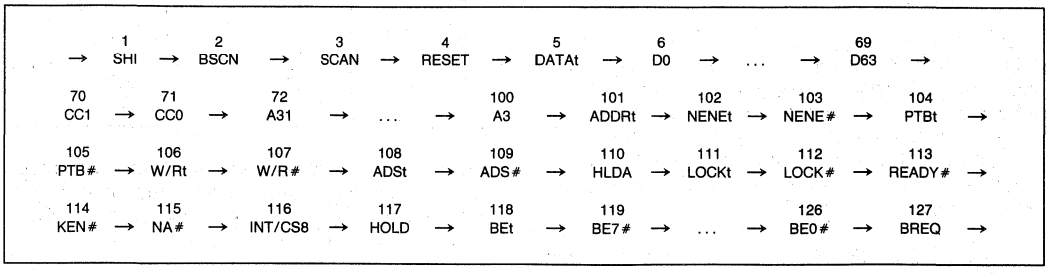


Figure 3.1. Order of Boundary Scan Chain

Pipelining may occur for the next bus cycle any time the current bus cycle requires more than two clock periods to finish (**m** > **2**). If a bus request is pending, the next cycle will be initiated when NA# is sampled active, even if the current cycle has not terminated. In this case, pipelining occurs. NA# is not recognized unitl after ADS# has become inactive.

To allow high transfer rates in large memory systems, two-level pipelining is supported (i.e., there may be up to three cycles in progress at one time). Pipelining enables a new word of data to be transferred every two clocks, even though the total cycle time may be up to six clocks.

4.2 Bus State Machine

The operation of the bus is described in terms of a bus state machine using a state transition diagram. Figure 4.1 illustrates the i860 microprocessor bus state machine. A bus cycle is composed of two or more states. Each bus state lasts for one CLK period.

The i860 microprocessor supports up to two levels of address pipelining. Once it has started the first bus cycle, it can generate up to two more cycles as long as READY# remains inactive. To start a new bus cycle while other cycles are still outstanding, NA# must be active for at least one clock cycle starting with the clock after the previous ADS#. NA# is latched internally.

States T_j and T_{jk}, for **j** = {1,2,3} and **k** = {1,2}, are used to describe the state of the i860 microprocessor Bus State Machine. Index **j** indicates the number of outstanding bus cycles while index **k** distinguishes the intermediate states for the **j**-th outstanding cycle.

Therefore there can be up to three outstanding cycles, and there are two possible intermediate states for each level of pipelining. T_{j1} is the next state after T_j, as long as **j** cycles are outstanding. T_{j2} is entered when NA# is active but the i860 microprocessor is not ready to start a new cycle.

Five conditions have to be met to start a new cycle while one or more cycles are already pending:

1. READY# inactive
2. NA# having been active
3. An internal request pending (BREQ active)
4. HOLD not active
5. Fewer than three cycles outstanding

Note that BREQ is asserted on the clock after the i860 microprocessor realizes an internal request for the bus.

Upon hardware RESET, the bus control logic enters the idle state T_I and awaits an internal request for a bus cycle. If a bus cycle is requested while there is no hold request from the system, a bus cycle begins, advancing to state T_1. On the next cycle, the state machine automatically advances to state T_{11}. If READY# is active in state T_{11}, the bus control logic returns either to T_I, if no new cycle is started, or to T_1, if a new cycle request is pending internally. In fact, if an internal bus request is pending each time READY# is active, the state machine continues to cycle between T_{11} and T_1.

However, if READY# is not active but the next address request is pending (as indicated by an active NA#), the state machine advances either to state T_2 (if an internal bus request is pending, signifying that two bus cycles are now outstanding), or to state T_{12} (if no bus internal request is pending, signifying NA# has been found active). Transitions from state T_{12} are similar to those from T_{11}.

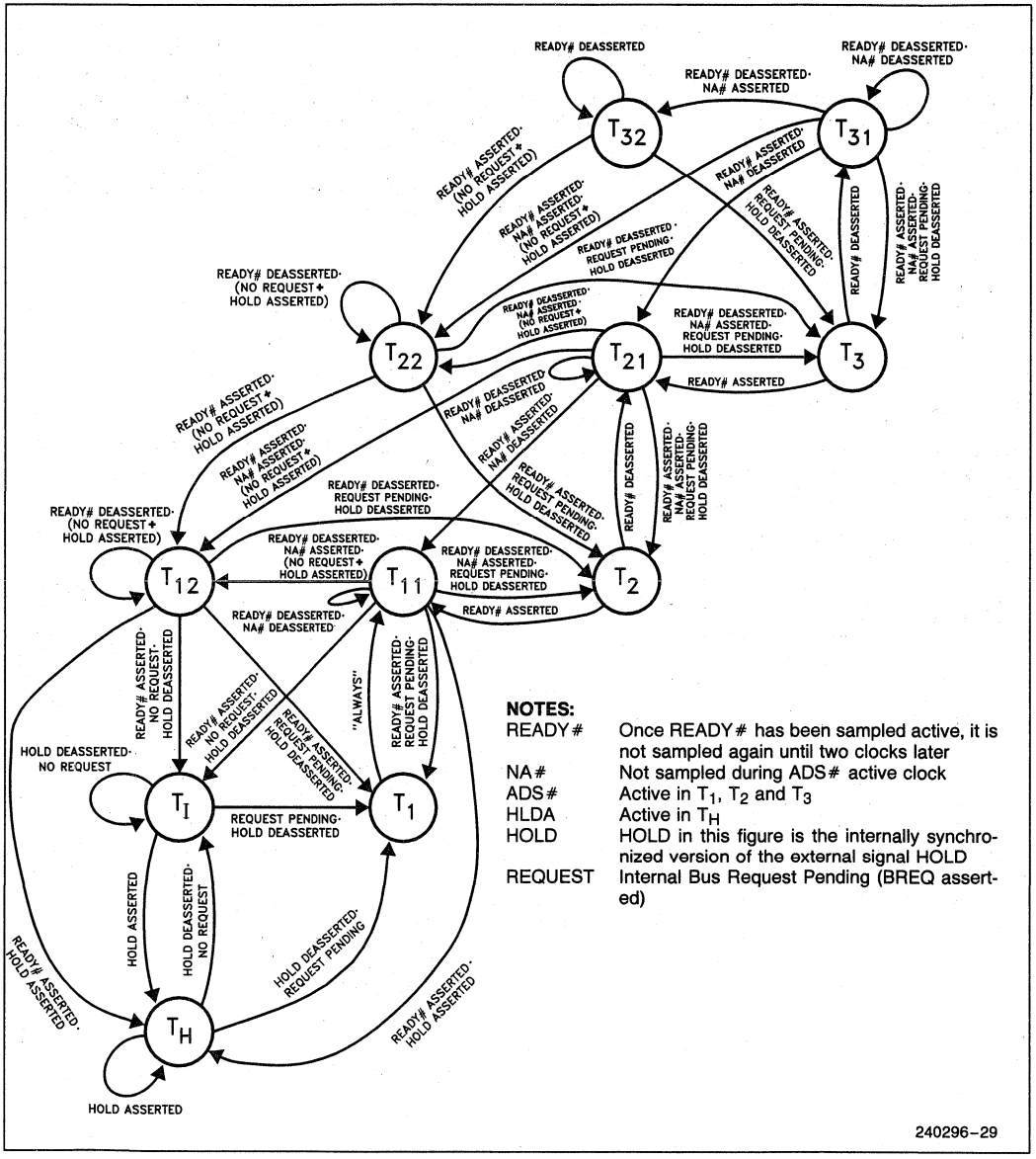


Figure 4.1. Bus State Machine

If two bus cycles are already outstanding (as indicated by T_{2k} for **k** = {1,2}) and NA# is latched active but READY# is not active, one more bus request causes entry into state T_3. Transitions from this state are similar to those from T_2.

In general, if there is an internal bus request each time both READY# and NA# are active, the state machine continues to oscillate between T_{j1} and T_j, for **j** = {2,3}.

When NA# is sampled active while there is a pending bus request, ADS# is activated in the next clock period (provided no more than two cycles are already outstanding).

Internal pending bus requests start new bus cycles only if no HOLD request has been recognized. T_H is entered from the idle state T_I, T_{11}, and T_{12}. HLDA is active in this state. There is a one clock delay to synchronize the HOLD input when the signal meets the respective minimum setup and hold time requirements. The state machine uses the synchronized HOLD to move from state to state.

4.3 Bus Cycles

Figures 4.2 through 4.10 illustrate combinations of bus cycles.

4.3.1 NONPIPELINED READ CYCLES

A read cycle begins with the clock in which ADS# is asserted. The i860 microprocessor begins driving the address during this clock. It samples READY# for active state every clock after the first clock. A minimum of two clocks is required per cycle. Data is latched when READY# is found active when sampled at the end of a clock period. Figure 4.2 illustrates nonpipelined read cycles with zero wait states.

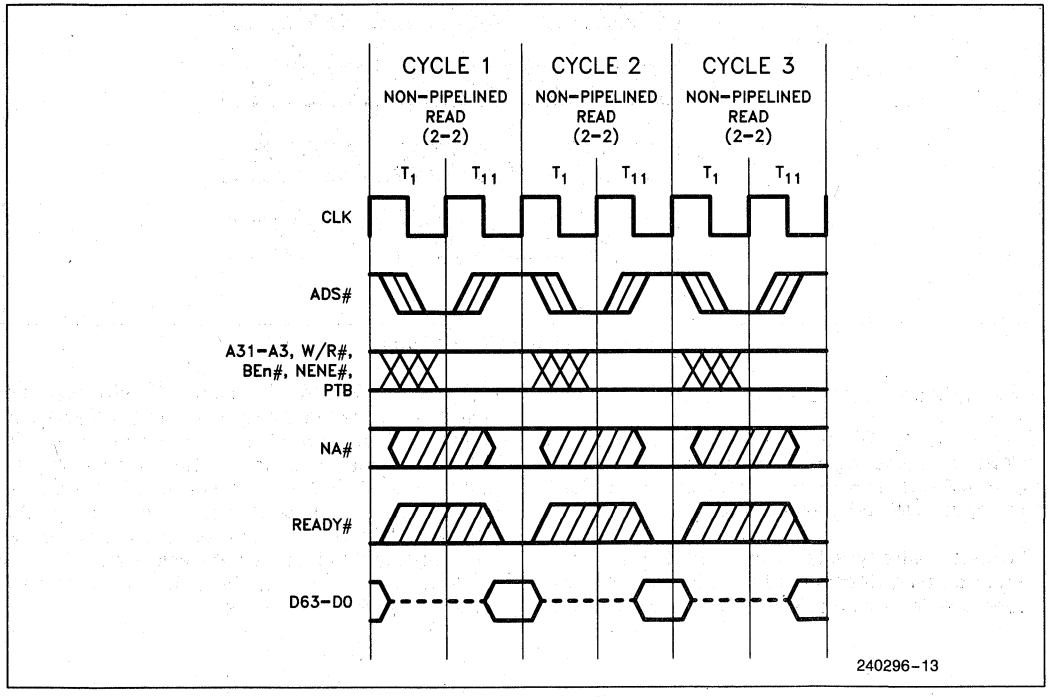


Figure 4.2. Fastest Read Cycles

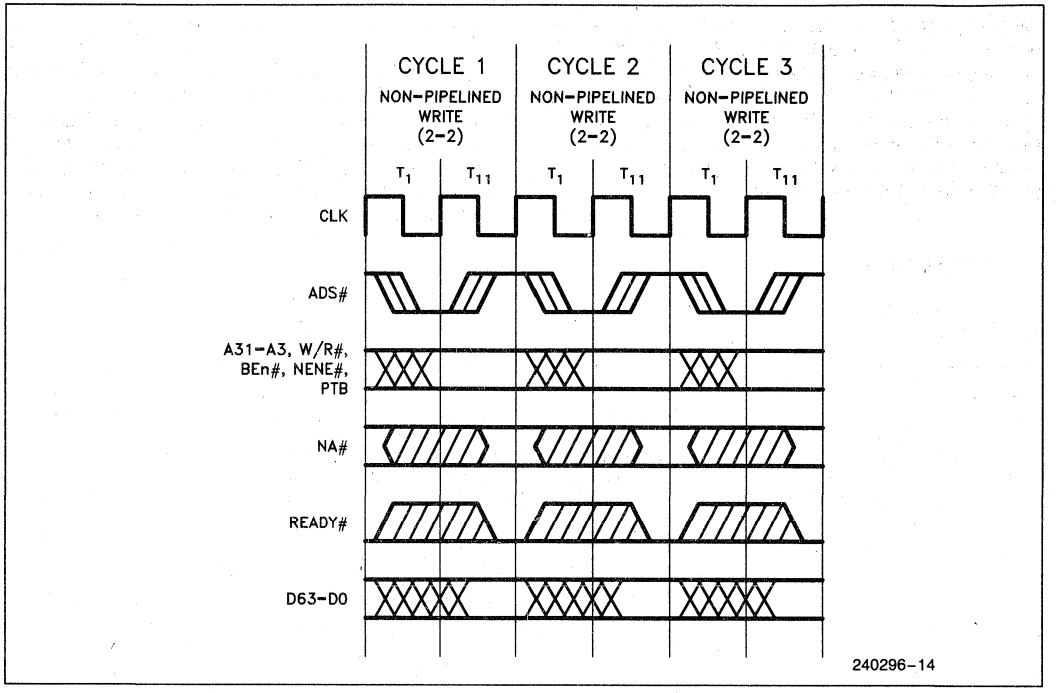


Figure 4.3. Fastest Write Cycles

4.3.2 NONPIPELINED WRITE CYCLES

The ADS# and READY# activity for write cycles follows the same logic as that for read cycles, as Figure 4.3 illustrates for back-to-back, nonpipelined write cycles with zero wait-states.

The fastest write cycle takes only two clocks to complete. However, when a read cycle immediately precedes a write cycle, the write cycle must contain a wait state, as illustrated in Figure 4.4. Because the device being read might still be driving the data bus during the first clock of the write cycle, there is a potential for bus contention. To help avoid such contention, the i860 microprocessor does not drive the data bus until the second clock of the write cycle. The wait state is required to provide the additional time necessary to terminate the write cycle. In other read-write combinations, the i860 microprocessor does not require a wait state.

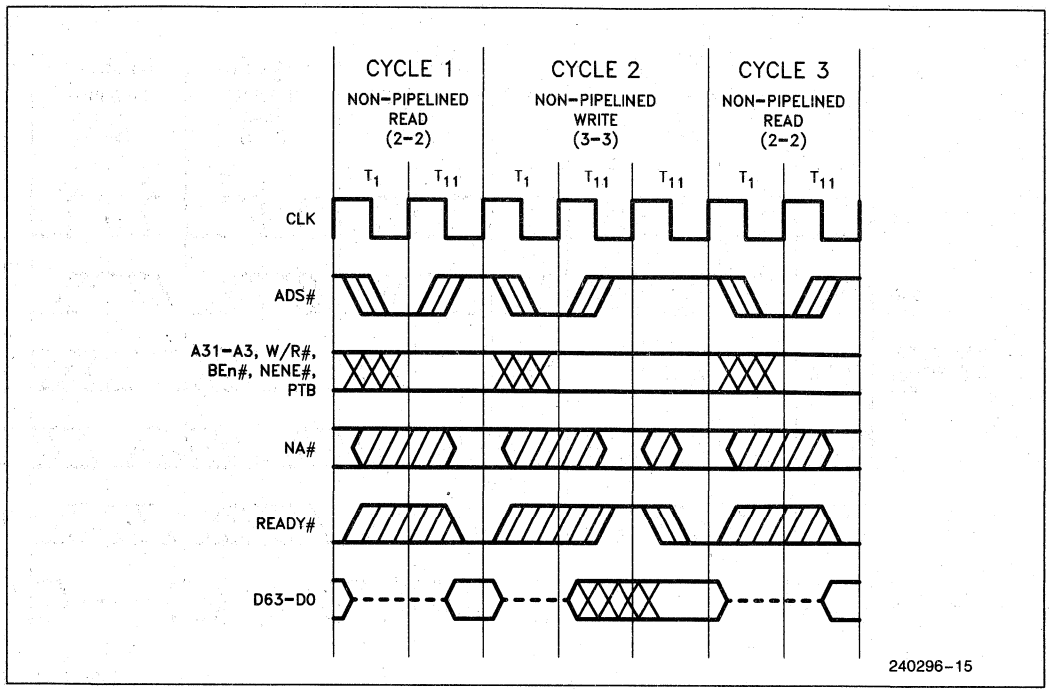


Figure 4.4. Fastest Read/Write Cycles

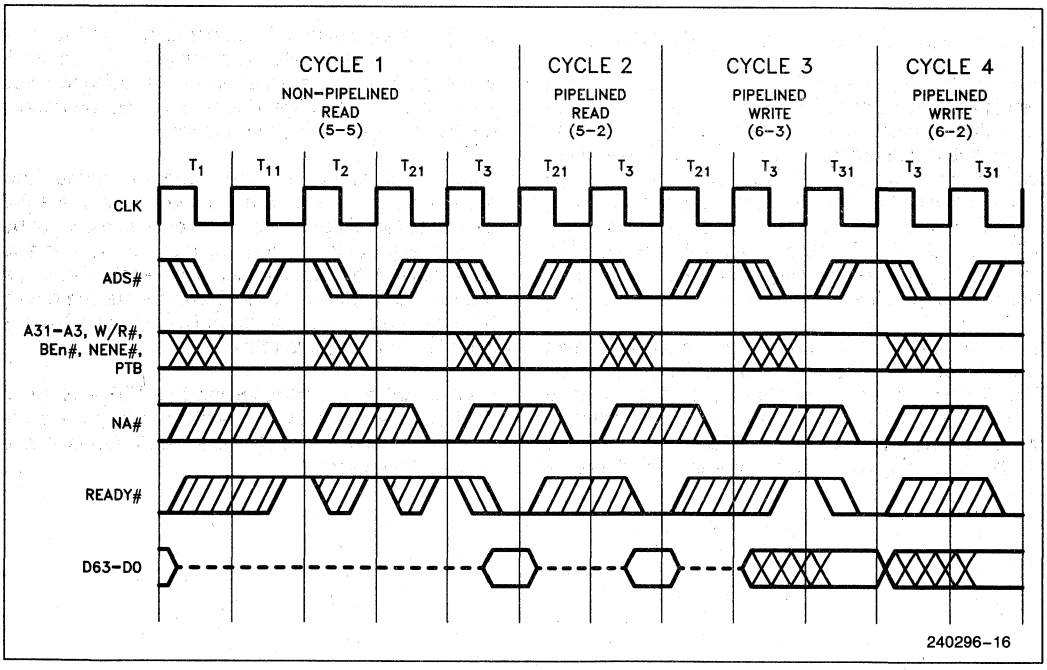


Figure 4.5. Pipelined Read Followed by Pipelined Write

6

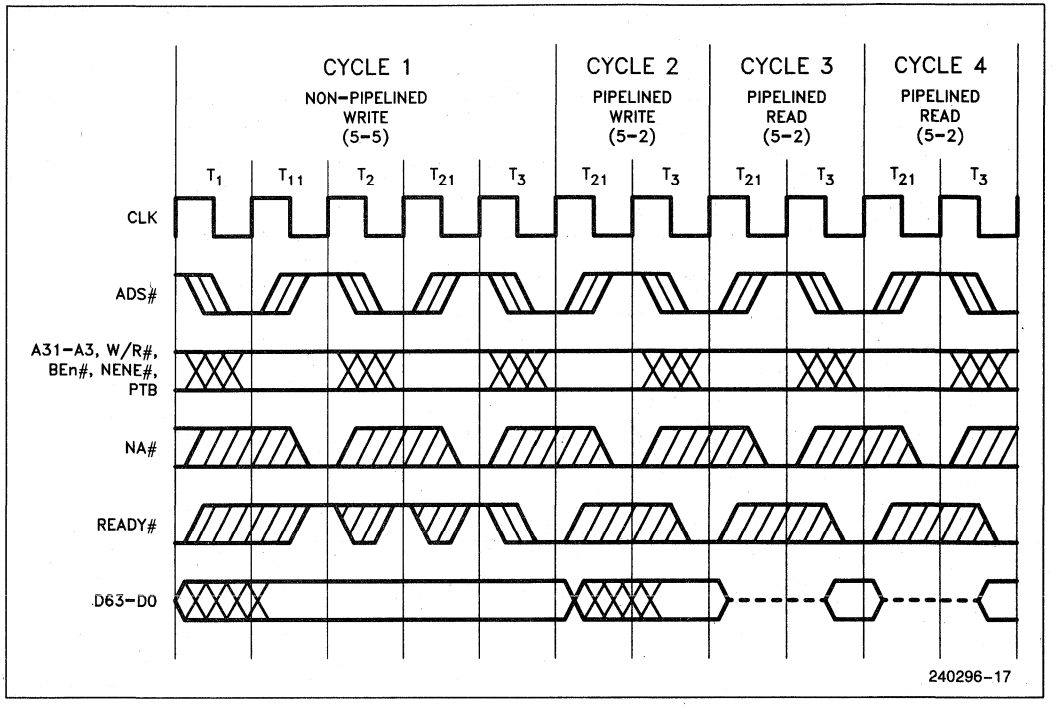


Figure 4.6. Pipelined Write Followed by Pipelined Read

4.3.3 PIPELINED READ AND WRITE CYCLES

Figures 4.5 and 4.6 illustrate combinations of non-pipelined and pipelined read and write cycles. The following description applies to both diagrams. While Cycle 1 is still in progress, two new cycles are initiated. By the time READY# first becomes active, the state machine has moved through states T_1, T_{11}, T_2, T_{21}, and T_3. Cycles 3 and 4 show how activating READY# terminates the corresponding outstanding cycle, and yet activating NA# while there is an internal request pending adds a new outstanding cycle.

In Figure 4.5, Cycle 3 is a write cycle following a read cycle; therefore, one wait state must be inserted. The i860 microprocessor does not drive the data bus until one clock after the read data is returned from the preceding read cycle. During Cycles 3 and 4, the state machine oscillates between states T_3 and T_{31} maintaining full bus capacity (two levels of pipelining; three outstanding cycles). Cycles 2, 3, and 4 in Figure 4.6 are 5-2 cycles; i.e. each requires a total cycle time of five clocks while the throughput rate is one cycle every two clocks.

Figure 4.7 illustrates in a more general manner how the NA# signal controls pipelining. Cycle 1 is a 2-2 cycle, the fastest possible. The next cycle cannot be started any earlier; therefore, there is no need to activate NA# to start the next cycle early. Cycle 2, a 3-3 read, is different. Cycle 3 can be started during the third state (a wait state) of Cycle 2, and NA# is asserted to accomplish this.

NA# is not activated following the ADS# clock of Cycle 3, thereby allowing Cycle 3 to terminate before the start of Cycle 4. As a result, Cycle 4 is a nonpipelined cycle.

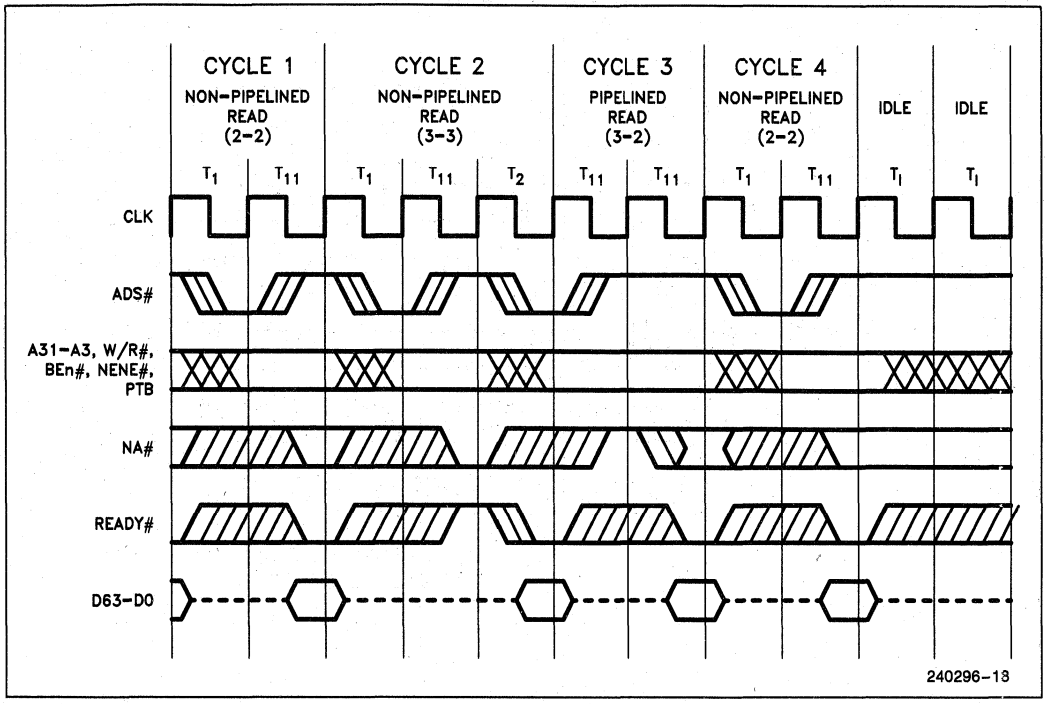


Figure 4.7. Pipelining Driven by NA#

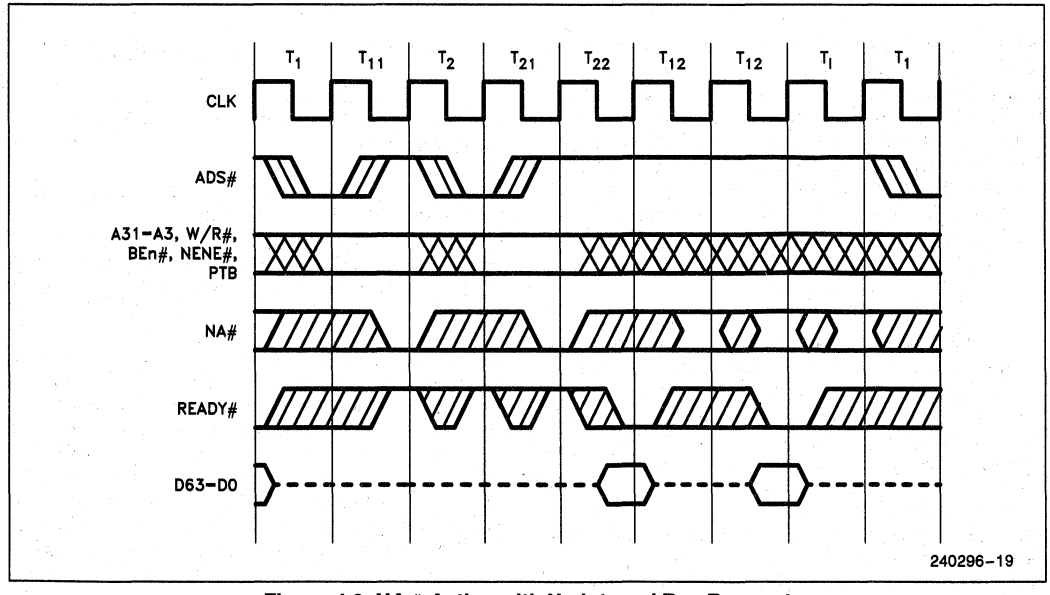


Figure 4.8. NA# Active with No Internal Bus Request

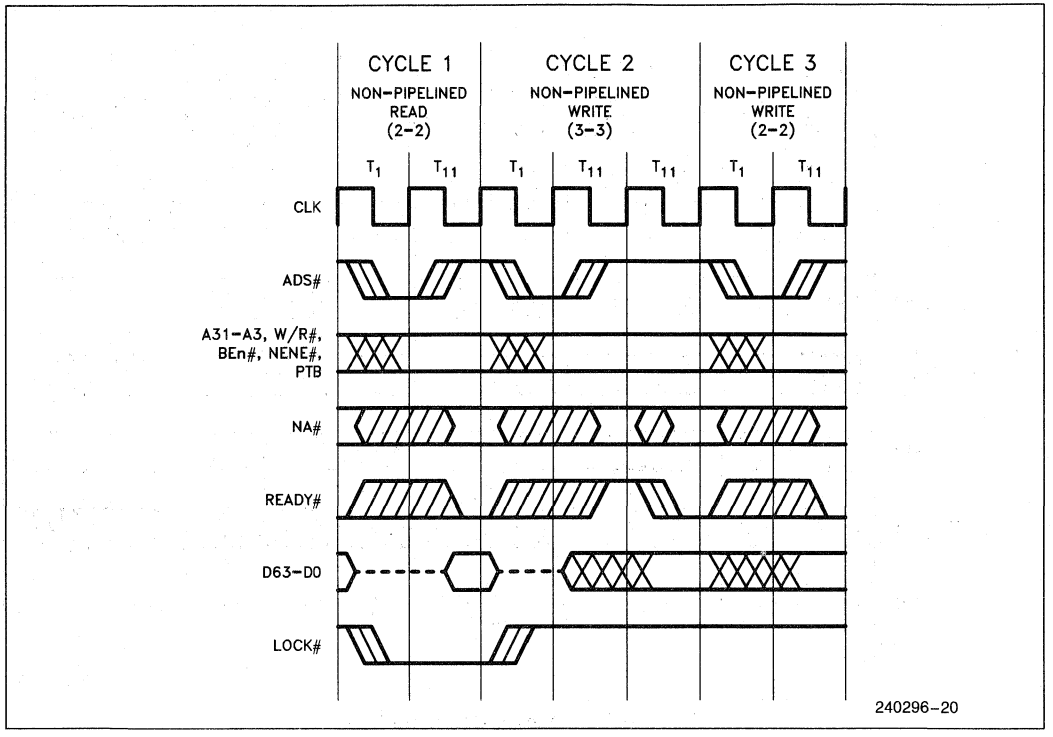


Figure 4.9. Locked Cycles

When there is no internal bus request, activating NA# does not start a new cycle; the i860 microprocessor, however, remembers that NA# has been activated. Figure 4.8 illustrates the situation where NA# is active but no internal bus request is pending. NA# is activated when two cycles are outstanding. Because there is no internal request pending until after one idle state, no new bus cycle is started during that period.

4.3.4 LOCKED CYCLES

The LOCK# signal is asserted when the current bus cycle is to be locked with the next bus cycle. Assertion of LOCK# may be initiated by a program's setting the BL bit of the **dirbase** register using the **lock** instruction (refer to section 2) or by the i860 microprocessor itself during page table updates.

In Figure 4.9, the first read cycle is to be locked with the following write cycle. If there were idle states between the cycles, the LOCK# signal would remain asserted. This is the case for a read/modify/write operation. Cycle 3 is not locked because LOCK# is no longer asserted when Cycle 2 starts.

4.3.5 HOLD AND BREQ ARBITRATION CYCLES

The HOLD, HLDA, and BREQ signals permit bus arbitration between the i860 microprocessor and another bus master.

See Figure 4.10. When HOLD is asserted, the i860 microprocessor does not relinquish control of the bus until all outstanding cycles are completed. If HOLD were asserted one clock earlier, the last i860 microprocessor bus cycle before HLDA would not be started.

The outputs (except HLDA and BREQ) float when HLDA is asserted. HOLD is sampled at the end of the clock in which it is activated. Recommended setup and hold times must be met to guarantee sampling one clock after external HOLD activation. When HOLD is sampled active, a one clock delay for internal synchronization follows. Likewise when HOLD is deasserted, there is a one-clock delay for internal synchronization before HLDA is deasserted.

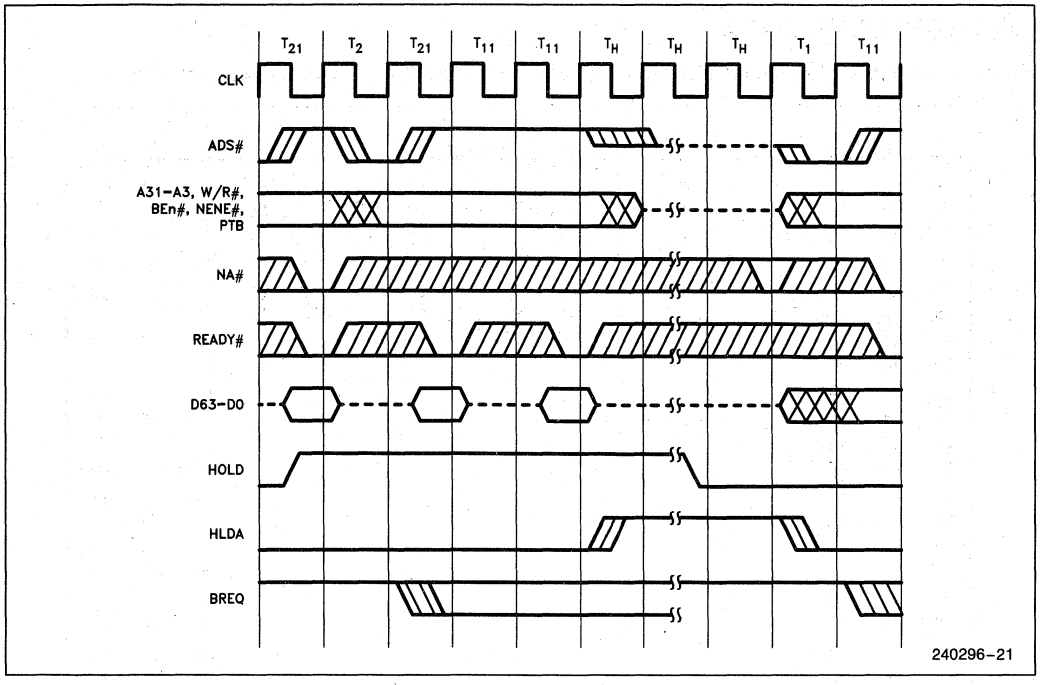


Figure 4.10. HOLD, HLDA, and BREQ

If, during a HOLD cycle, an internal bus request is generated, BREQ is activated even though HLDA is asserted. It remains active at least until the clock after ADS# is activated for the requested cycle.

4.4 Bus States During RESET

Figure 4.11 shows how INT/CS8 is sampled during the clock period just before the falling edge of RESET. If INT/CS8 is sampled active, the i860 microprocessor enters CS8 mode. No inputs (except for HOLD and INT/CS8) are sampled during RESET.

Note that, because HOLD is recognized even while RESET is active, the HLDA output signal may also become active during RESET. Refer to Table 3.4 "Output Pin Status during Reset".

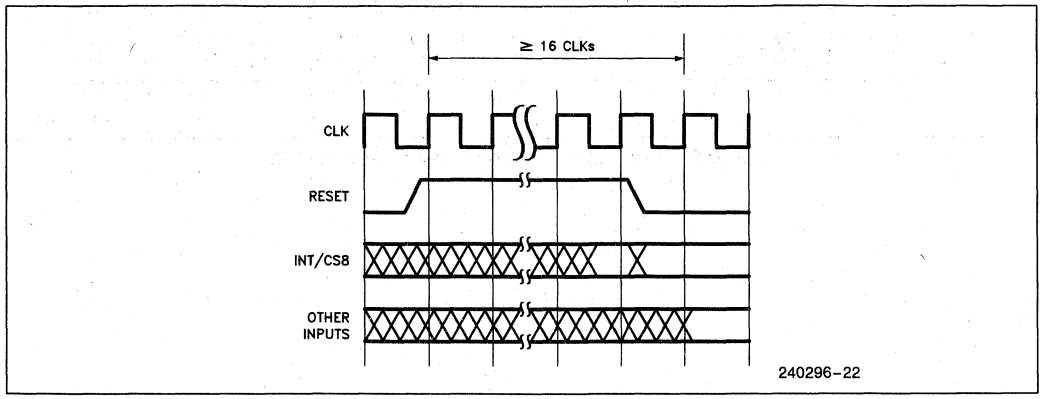


Figure 4.11. Reset Activities

5.0 MECHANICAL DATA

Figures 5.1 and 5.2 show the locations of pins; Tables 5.1 and 5.2 help to locate pin identifiers.

	S	R	Q	P	N	M	L	K	J	H	G	F	E	D	C	B	A	
1	( ) V_{CC}	( ) V_{SS}	( ) V_{CC}	( ) V_{SS}	( ) A12	( ) A17	( ) A19	( ) A21	( ) A23	( ) A25	( ) A29	( ) A31	( ) V_{CC}	( ) V_{SS}	( ) V_{CC}	( ) V_{SS}	( ) V_{CC}	1
2	( ) V_{SS}	( ) V_{CC}	( ) V_{SS}	( ) A8	( ) A10	( ) A13	( ) A15	( ) A18	( ) A20	( ) A24	( ) A27	( ) A28	( ) CC0	( ) V_{CC}	( ) V_{SS}	( ) V_{CC}	( ) V_{SS}	2
3	( ) V_{CC}	( ) V_{SS}	( ) A6	( ) A7	( ) A9	( ) A11	( ) A14	( ) A16	( ) CLK	( ) A22	( ) A26	( ) A30	( ) CC1	( ) D62	( ) D60	( ) V_{SS}	( ) V_{CC}	3
4	( ) V_{SS}	( ) V_{CC}	( ) A5												( ) D63	( ) D59	( ) V_{SS}	4
5	( ) V_{CC}	( ) A4	( ) A3												( ) D61	( ) D58	( ) D56	5
6	( ) W/R#	( ) NENE#	( ) PTB												( ) D57	( ) D54	( ) D52	6
7	( ) ADS#	( ) HLDA	( ) BREQ												( ) D55	( ) D53	( ) D50	7
8	( ) LOCK#	( ) KEN#	( ) READY#												( ) D51	( ) D49	( ) D48	8
9	( ) INT/CS8	( ) NA#	( ) HOLD												( ) D47	( ) D45	( ) D46	9
10	( ) BE5#	( ) BE7#	( ) BE6#												( ) D43	( ) D42	( ) D44	10
11	( ) BE3#	( ) BE2#	( ) BE4#												( ) D39	( ) D41	( ) D40	11
12	( ) SHI	( ) BE1#	( ) BE0#												( ) D37	( ) D36	( ) D38	12
13	( ) RESET	( ) SCAN	( ) BSCN												( ) D35	( ) D34	( ) V_{CC}	13
14	( ) V_{SS}	( ) D0	( ) D1												( ) D33	( ) V_{CC}	( ) V_{SS}	14
15	( ) V_{CC}	( ) V_{SS}	( ) D2	( ) D3	( ) D5	( ) D7	( ) D11	( ) D13	( ) D17	( ) D21	( ) D23	( ) D27	( ) D29	( ) D31	( ) D32	( ) V_{SS}	( ) V_{CC}	15
16	( ) V_{SS}	( ) V_{CC}	( ) V_{SS}	( ) V_{CC}	( ) D4	( ) D9	( ) D8	( ) D15	( ) D14	( ) D19	( ) D22	( ) D25	( ) D28	( ) D30	( ) V_{SS}	( ) V_{CC}	( ) V_{SS}	16
17	( ) V_{CC}	( ) V_{SS}	( ) V_{CC}	( ) V_{SS}	( ) V_{CC}	( ) D6	( ) D10	( ) D12	( ) D16	( ) D18	( ) D20	( ) D24	( ) D26	( ) V_{SS}	( ) V_{CC}	( ) V_{SS}	( ) V_{CC}	17
	S	R	Q	P	N	M	L	K	J	H	G	F	E	D	C	B	A	

240296-23

Figure 5.1. Pin Configuration—View from Top Side

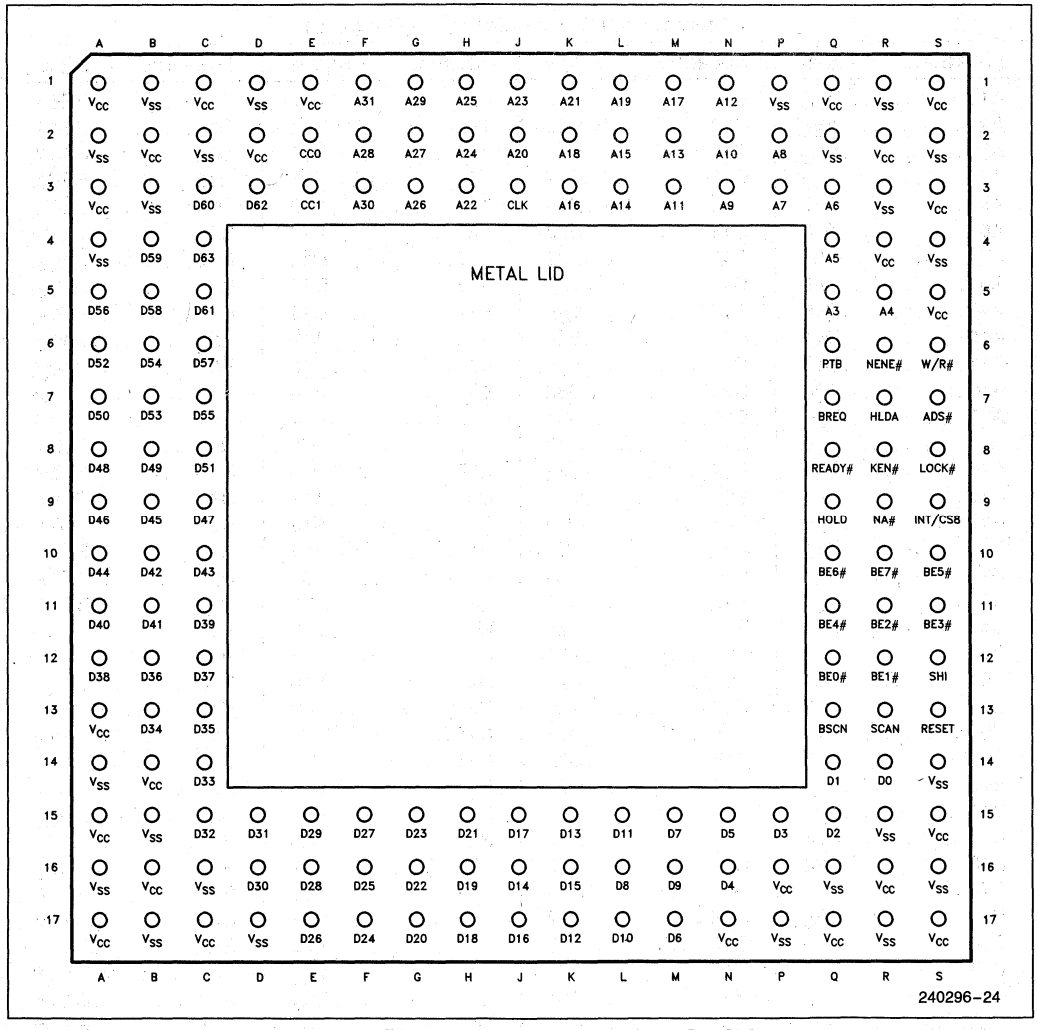


Figure 5.2. Pin Configuration—View from Pin Side

6

Table 5.1. Pin Cross Reference by Location

Location	Signal	Location	Signal	Location	Signal	Location	Signal
A1	V_{CC}	C9	D47	J15	D17	Q10	BE6#
A2	V_{SS}	C10	D43	J16	D14	Q11	BE4#
A3	V_{CC}	C11	D39	J17	D16	Q12	BE0#
A4	V_{SS}	C12	D37	K1	A21	Q13	BSCN
A5	D56	C13	D35	K2	A18	Q14	D1
A6	D52	C14	D33	K3	A16	Q15	D2
A7	D50	C15	D32	K15	D13	Q16	V_{SS}
A8	D48	C16	V_{SS}	K16	D15	Q17	V_{CC}
A9	D46	C17	V_{CC}	K17	D12	R1	V_{SS}
A10	D44	D1	V_{SS}	L1	A19	R2	V_{CC}
A11	D40	D2	V_{CC}	L2	A15	R3	V_{SS}
A12	D38	D3	D62	L3	A14	R4	V_{CC}
A13	V_{CC}	D15	D31	L15	D11	R5	A4
A14	V_{SS}	D16	D30	L16	D8	R6	NENE#
A15	V_{CC}	D17	V_{SS}	L17	D10	R7	HLDA
A16	V_{SS}	E1	V_{CC}	M1	A17	R8	KEN#
A17	V_{CC}	E2	CC0	M2	A13	R9	NA#
B1	V_{SS}	E3	CC1	M3	A11	R10	BE7#
B2	V_{CC}	E15	D29	M15	D7	R11	BE2#
B3	V_{SS}	E16	D28	M16	D9	R12	BE1#
B4	D59	E17	D26	M17	D6	R13	SCAN
B5	D58	F1	A31	N1	A12	R14	D0
B6	D54	F2	A28	N2	A10	R15	V_{SS}
B7	D53	F3	A30	N3	A9	R16	V_{CC}
B8	D49	F15	D27	N15	D5	R17	V_{SS}
B9	D45	F16	D25	N16	D4	S1	V_{CC}
B10	D42	F17	D24	N17	V_{CC}	S2	V_{SS}
B11	D41	G1	A29	P1	V_{SS}	S3	V_{CC}
B12	D36	G2	A27	P2	A8	S4	V_{SS}
B13	D34	G3	A26	P3	A7	S5	V_{CC}
B14	V_{CC}	G15	D23	P15	D3	S6	W/R#
B15	V_{SS}	G16	D22	P16	V_{CC}	S7	ADS#
B16	V_{CC}	G17	D20	P17	V_{SS}	S8	LOCK#
B17	V_{SS}	H1	A25	Q1	V_{CC}	S9	INT/CS8
C1	V_{CC}	H2	A24	Q2	V_{SS}	S10	BE5#
C2	V_{SS}	H3	A22	Q3	A6	S11	BE3#
C3	D60	H15	D21	Q4	A5	S12	SHI
C4	D63	H16	D19	Q5	A3	S13	RESET
C5	D61	H17	D18	Q6	PTB	S14	V_{SS}
C6	D57	J1	A23	Q7	BREQ	S15	V_{CC}
C7	D55	J2	A20	Q8	READY#	S16	V_{SS}
C8	D51	J3	CLK	Q9	HOLD	S17	V_{CC}

Table 5.2. Pin Cross Reference by Pin Name

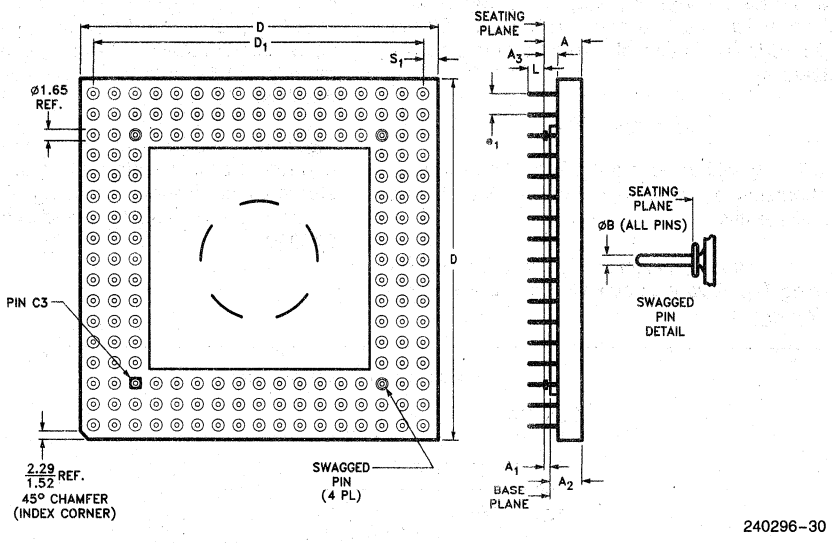
Signal	Location	Signal	Location	Signal	Location	Signal	Location
A3	Q5	CLK	J3	D41	B11	V_{CC}	B16
A4	R5	D0	R14	D42	B10	V_{CC}	C1
A5	Q4	D1	Q14	D43	C10	V_{CC}	C17
A6	Q3	D2	Q15	D44	A10	V_{CC}	D2
A7	P3	D3	P15	D45	B9	V_{CC}	E1
A8	P2	D4	N16	D46	A9	V_{CC}	N17
A9	N3	D5	N15	D47	C9	V_{CC}	P16
A10	N2	D6	M17	D48	A8	V_{CC}	Q1
A11	M3	D7	M15	D49	B8	V_{CC}	Q17
A12	N1	D8	L16	D50	A7	V_{CC}	R2
A13	M2	D9	M16	D51	C8	V_{CC}	R4
A14	L3	D10	L17	D52	A6	V_{CC}	R16
A15	L2	D11	L15	D53	B7	V_{CC}	S1
A16	K3	D12	K17	D54	B6	V_{CC}	S3
A17	M1	D13	K15	D55	C7	V_{CC}	S5
A18	K2	D14	J16	D56	A5	V_{CC}	S15
A19	L1	D15	K16	D57	C6	V_{CC}	S17
A20	J2	D16	J17	D58	B5	V_{SS}	A2
A21	K1	D17	J15	D59	B4	V_{SS}	A4
A22	H3	D18	H17	D60	C3	V_{SS}	A14
A23	J1	D19	H16	D61	C5	V_{SS}	A16
A24	H2	D20	G17	D62	D3	V_{SS}	B1
A25	H1	D21	H15	D63	C4	V_{SS}	B3
A26	G3	D22	G16	HLDA	R7	V_{SS}	B15
A27	G2	D23	G15	HOLD	Q9	V_{SS}	B17
A28	F2	D24	F17	INT/CS8	S9	V_{SS}	C2
A29	G1	D25	F16	KEN#	R8	V_{SS}	C16
A30	F3	D26	E17	LOCK#	S8	V_{SS}	D1
A31	F1	D27	F15	NA#	R9	V_{SS}	D17
ADS#	S7	D28	E16	NENE#	R6	V_{SS}	P1
BE0#	Q12	D29	E15	PTB	Q6	V_{SS}	P17
BE1#	R12	D30	D16	READY#	Q8	V_{SS}	Q2
BE2#	R11	D31	D15	RESET	S13	V_{SS}	Q16
BE3#	S11	D32	C15	SCAN	R13	V_{SS}	R1
BE4#	Q11	D33	C14	SHI	S12	V_{SS}	R3
BE5#	S10	D34	B13	V_{CC}	A1	V_{SS}	R15
BE6#	Q10	D35	C13	V_{CC}	A3	V_{SS}	R17
BE7#	R10	D36	B12	V_{CC}	A13	V_{SS}	S2
BREQ	Q7	D37	C12	V_{CC}	A15	V_{SS}	S4
BSCN	Q13	D38	A12	V_{CC}	A17	V_{SS}	S14
CC0	E2	D39	C11	V_{CC}	B2	V_{SS}	S16
CC1	E3	D40	A11	V_{CC}	B14	W/R#	S6

6

Table 5.3. Ceramic PGA Package Dimension Symbols

Letter or Symbol	Description of Dimensions
A	Distance from seating plane to highest point of body
A_1	Distance between seating plane and base plane (lid)
A_2	Distance from base plane to highest point of body
A_3	Distance from seating plane to bottom of body
B	Diameter of terminal lead pin
D	Largest overall package dimension of length
D_1	A body length dimension, outer lead center to outer lead center
e_1	Linear spacing between true lead position centerlines
L	Distance from seating plane to end of lead
S_1	Other body dimension, outer lead center to edge of body

NOTES:
1. Controlling dimension: millimeter.
2. Dimension "e_1" ("e") is non-cumulative.
3. Seating plane (standoff) is defined by P.C. board hole size: 0.0415–0.0430 inch.
4. Dimensions "B", "B_1" and "C" are nominal.
5. Details of Pin 1 identifier are optional.



Family: Ceramic Pin Grid Array Package						
Symbol	Millimeters			Inches		
	Min	Max	Notes	Min	Max	Notes
A	3.56	4.57		0.140	0.180	
A_1	0.64	1.14	SOLID LID	0.025	0.045	SOLID LID
A_2	23	0.30	SOLID LID	0.110	0.140	SOLID LID
A_3	1.14	1.40		0.045	0.055	
B	0.43	0.51		0.017	0.020	
D	44.07	44.83		1.735	1.765	
D_1	40.51	40.77		1.595	1.605	
e_1	2.29	2.79		0.090	0.110	
L	2.54	3.30		0.100	0.130	
N	168			168		
S_1	1.52	2.54		0.060	0.100	
ISSUE	IWS REV X 7/15/88					

Figure 5.3. 168 Lead Ceramic PGA Package Dimensions

6.0 PACKAGE THERMAL SPECIFICATIONS

For this section, let:

P = maximum power consumption

T_C = case temperature

T_A = ambient air temperature

θ_{CA} = thermal resistance from case to ambient air

θ_{JC} = thermal resistance from junction to case

θ_{JA} = thermal resistance from junction to ambient air

The i860 microprocessor is specified for operation when T_C is within the range of 0°C–85°C. T_C may be measured in any environment to determine whether the i860 microprocessor is within specified operating range. The case temperature should be measured at the center of the top surface opposite the pins.

T_A can be calculated from θ_{CA} (thermal resistance from case to ambient) with the following equation:

$$T_A = T_C - P^*\theta_{CA}$$

Typical values for θ_{CA} and θ_{JC} at various airflows are given in Table 6.1 for the 1.75 sq. in., 168 pin, ceramic PGA. θ_{JC} is also shown so that θ_{JA} can be calculated by:

$$\theta_{CA} = \theta_{JA} = \theta_{JC}$$

Note that θ_{JC} with a heatsink differs from θ_{JC} without a heatsink because case temperature is measured differently.

Table 6.2 shows the maximum T_A allowable (without exceeding T_C) at various airflows and operating frequencies (f_{CLK}).

Note that T_A is greatly improved by attaching "fins" or a "heat sink" to the package. P (the maximum power consumption) is calculated by using the maximum I_{CC} at 5V as tabulated in the *DC Characteristics* of section 7.

Figure 6.1 gives typical I_{CC} derating with case temperature. For more information on heat sinks, measurement techniques, or package characteristics, refer to *Intel Packaging Handbook*, order number 240800.

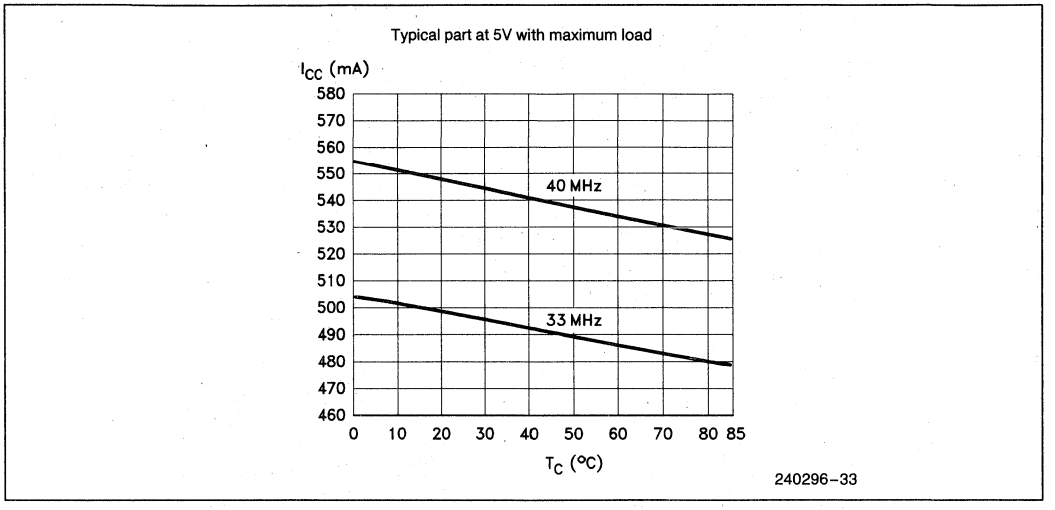


Figure 6.1. I_{CC} vs Case Temperature

Table 6.1. θ_{CA} at Various Airflows and θ_{JC}

In °C/Watt

	Airflow-ft/min (m/sec)						
	θ_{JC}	0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
θ_{CA} with Heat Sink*	2	13	9	5.5	5	3.9	3.4
θ_{CA} without Heat Sink	1.5	17.5	13	11	9.5	8.5	8

*Nine-fin, unidirectional heat sink (fin dimensions: 0.350" height, 0.040 width, 0.115" center-to-center spacing, 1.530" length).

Table 6.2. Maximum T_A at Various Airflows

In °C

	f_{CLK} (MHz)	Airflow-ft/min (m/sec) 0 (0)	200 (1.01)	400 (2.03)	600 (3.04)	800 (4.06)	1000 (5.07)
T_A with Heat Sink*	33.3	52	67	73	75.5	75.5	78.5
	40	49	65.5	72	74.5	77	77.8
T_A without Heat Sink	33.3	32.5	46	52	56.5	59.5	61
	40	28	43	49	54	57.5	59

*Nine-fin unidirectional heat sink (fin dimensions: 0.350" height, 0.040 width, 0.115" center-to-center spacing, 1.530" length).

7.0 ELECTRICAL DATA

Inputs and outputs are TTL compatible, except for CLK. All input and output timings are specified relative to the 1.5 volt level of the rising edge of CLK and refer to the point that the signals reach 1.5V.

NOTICE: This data sheet contains preliminary information on new products in production. The specifications are subject to change without notice.

**WARNING: Stressing the device beyond the "Absolute Maximum Ratings" may cause permanent damage. These are stress ratings only. Operation beyond the "Operating Conditions" is not recommended and extended exposure beyond the "Operating Conditions" may affect device reliability.*

7.1 Absolute Maximum Ratings

Case Temperature T_C under Bias .....0°C to 85°C

Storage Temperature .....−65°C to +150°C

Voltage on Any Pin
with Respect to Ground.....−0.5 to 6.5V

7.2 D.C. Characteristics

Table 7.1. DC Characteristics

T_C = 0°C to 85°C, V_{CC} = 5V ±5%

Symbol	Parameter	Min	Max	Units	Notes
V_{IL}	Input LOW Voltage	−0.3	+0.8	V	
V_{IH}	Input HIGH Voltage	2.0	V_{CC}+0.3	V	
V_{ILC}	CLK Input LOW Voltage	−0.3	+0.8	V	
V_{IHC}	CLK Input HIGH Voltage	3.0	V_{CC} + 0.3	V	
V_{OL}	Output LOW Voltage		0.45	V	(Note 1)
V_{OH}	Output HIGH Voltage	2.4		V	(Note 2)
I_{CC}	Power Supply Current				
	CLK = 33.3 MHz		600	mA	V_{CC} @5V
	CLK = 40.0 MHz		650	mA	V_{CC} @5V
I_{LI}	Input Leakage Current		±15	μA	No pullup or pulldown
I_{LO}	Output Leakage Current		±15	μA	
C_{IN}	Input Capacitance		15	pF	(Note 3)
C_O	I/O or Output Capacitance		15	pF	(Note 3)
C_{CLK}	Clock Capacitance		20	pF	(Note 3)

NOTES:
1. This parameter is measured at 4.0 mA for A31–A3, D63–D0, BE7#–BE0#; at 5.0 mA for all other outputs.
2. This parameter is measured at 1.0 mA for A31–A3, D63–D0, BE7#–BE0#; at 0.9 mA all other outputs.
3. These are not tested. They are guaranteed by design characterization.

7.3 A.C. Characteristics

Table 7.2. A.C. Characteristics

T_C = 0°C to 85°C, V_{CC} = 5V ±5%

All timings measured at CLK = 1.5V unless otherwise specified.

Symbol	Parameter	33 MHz		40 MHz		50 MHz		Notes
		Min (ns)	Max (ns)	Min (ns)	Max (ns)	Min (ns)	Max (ns)	
t1	CLK Period	30	125	25	125	20	125	
t2	CLK High Time	5		3		3		at 3V
t3	CLK Low Time	7		5		5		at 0.8V
t4	CLK Fall Time		7		7		6	3V–0.8V
t5	CLK Rise Time		7		7		6	0.8V–3V
t6a	A31–A3, PTB, W/R#, NENE# Valid Delay	3.5	23	3.5	19	3.5	14	50 pF Load
t6b	BEn#* Valid Delay	3.5	25	3.5	21	3.5	17	50 pF Load
t7	Float Time, All	3.5	30	3.5	25	3.5	20	(Note 1)
t8	ADS#, BREQ, LOCK#, HLDA Valid Delay	3.5	20	3.5	15	3.5	10	50 pF Load
t9	D63–D0 Valid Delay	3.5	35	3.5	31	3.5	26	50 pF Load
t10	Setup Time, All Inputs except DATA	11		8		5		(Note 2)
t11	Hold Time All Inputs except DATA	4		3		2		(Note 2)
t14	Data Setup Time	11		8		6.5		
t15	DATA Hold Time	4		3		2.5		

NOTES:

1. Float condition occurs when maximum output current becomes less than I_{LO} in magnitude. Float delay is not tested.
2. INT and HOLD are asynchronous inputs. The setup and hold specifications are given for test purposes or to assure recognition on a specific rising edge of CLK. INT should remain asserted until software acknowledges the interrupt.

* n = 0, 1, ..., 7

ADVANCE INFORMATION ON INTEL PRODUCT

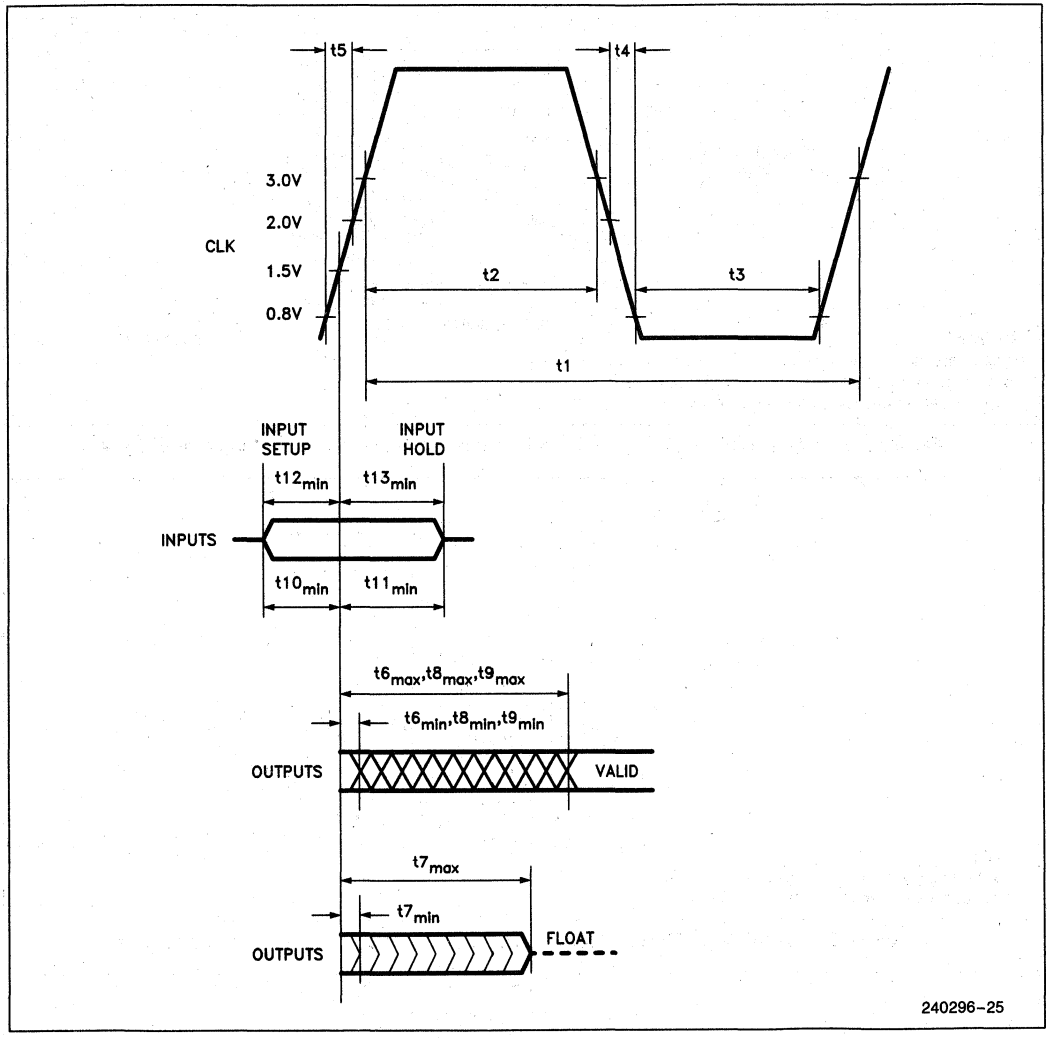
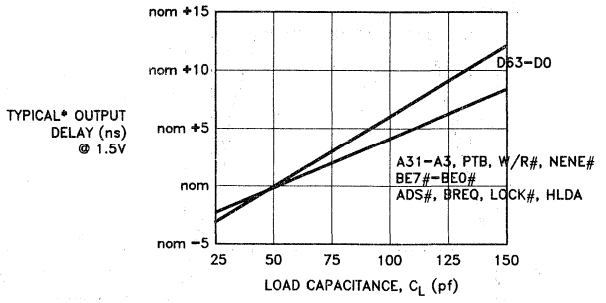


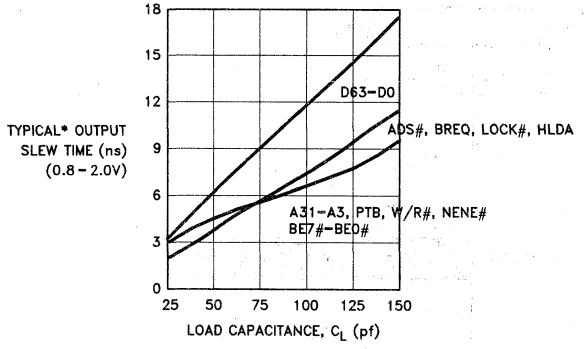
Figure 7.1. CLK, Input, and Output Timings



240296-26

NOTES:
Graphs are not linear outside the C_L range shown.
nom = nominal value given in the AC timing table.
*Typical part under worst-case conditions.

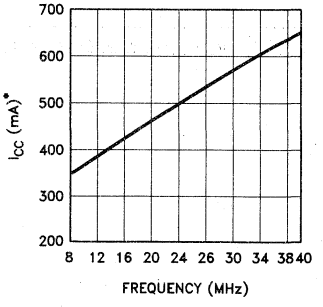
Figure 7.2. Typical Output Delay vs Load Capacitance under Worst-Case Conditions



240296-27

NOTES:
Graphs are not linear outside the C_L range shown.
*Typical part under worst-case conditions.

Figure 7.3. Typical Slew Time vs Load Capacitance under Worst-Case Conditions



240296-28

NOTES:
Graphs are not linear outside the frequency range shown.
*Worst-case supply current at 5V.

Figure 7.4. Typical I_{CC} vs Frequency

8.0 INSTRUCTION SET

Key to abbreviations:

For register operands, the abbreviations that describe the operands are composed of two parts. The first part describes the type of register:

c — One of the control registers **fir, psr, epsr, dirbase, db,** or **fsr**

f — One of the floating-point registers: **f0** through **f31**

i — One of the integer registers: **r0** through **r31**

The second part identifies the field of the machine instruction into which the operand is to be placed:

src1 — The first of the two source-register designators, which may be either a register or a 16-bit immediate constant or address offset. The immediate value is zero-extended for logical operations and is sign-extended for add and subtract operations (including **addu** and **subu**) and for all addressing calculations.

src1ni — Same as *src1* except that no immediate constant or address offset value is permitted.

src1s — Same as *src1* except that the immediate constant is a 5-bit value that is zero-extended to 32 bits.

src2 — The second of the two source-register designators.

dest — The destination register designator.

Thus, the operand specifier *isrc2*, for example, means that an integer register is used and that the encoding of that register must be placed in the *src2* field of the machine instruction.

Other (nonregister) operands are specified by a one-part abbreviation that represents both the type of operand required and the instruction field into which the value of the operand is placed:

#const — A 16-bit immediate constant or address offset that the i860 microprocessor sign-extends to 32 bits when computing the effective address.

lbroff — A signed, 26-bit, immediate, relative branch offset.

sbroff — A signed, 16-bit, immediate, relative branch offset.

brx — A function that computes the target address by shifting the offset (either *lbroff* or *sbroff*) left by two bits, sign-extending it to 32 bits, and adding the result to the current instruction pointer plus four. The resulting target address may lie anywhere within the address space.

Unless otherwise specified, floating-point operations accept single- or double-precision source operands and produce a result of equal or greater precision. Both input operands must have the same precision. The source and result precision are specified by a two-letter suffix to the mnemonic of the operation.

Other abbreviations include:

.p — Precision specification **.ss, .sd,** or **.dd** (**.ds** not permitted). Refer to Table 8.1.

.r — Precision specification **.ss, .sd, .ds,** or **.dd**. Refer to Table 8-1.

.v — **.sd** or **.dd**. Refer to Table 8-1.

.w — **.ss** or **.dd.** Refer to Table 8.1.

.x — **.b** (8 bits), **.s** (16 bits), or **.l** (32 bits)

.y — **.l** (32 bits), **.d** (64 bits), or **.q** (128 bits)

.z — **.l** (32 bits), or **.d** (64 bits)

Table 8.1. Precision Specification

Suffix	Source Precision	Result Precision
.ss	single	single
.sd	single	double
.dd	double	double
.ds	double	single

mem.x(address) The contents of the memory location indicated by *address* with a size of *x*.

PM The pixel mask, which is considered as an array of eight bits PM[7]..PM[0], where PM[0] is the least significant bit.

8.1 Instruction Definitions in Alphabetical Order

adds *isrc1, isrc2, idest* ..... **Add Signed**
idest ← *isrc1* + *isrc2*
OF ← (bit 31 carry ≠ bit 30 carry)
CC set if *isrc2* < −*isrc1* (signed)
CC clear if *isrc2* ≥ −*isrc1* (signed)

addu *isrc1, isrc2, idest* ..... **Add Unsigned**
idest ← *isrc1* + *isrc2*
OF ← bit 31 carry
CC ← bit 31 carry

and *isrc1, isrc2, idest* ..... **Logical AND**
idest ← *isrc1* and *isrc2*
CC set if result is zero, cleared otherwise

andh *#const, isrc2, idest* ..... **Logical AND High**
idest ← (*#const* shifted left 16 bits) and *isrc2*
CC set if result is zero, cleared otherwise

andnot *isrc1, isrc2, idest* ..... **Logical AND NOT**
idest ← not *isrc1* and *isrc2*
CC set if result is zero, cleared otherwise

andnoth *#const, isrc2, idest* ..... **Logical AND NOT High**
idest ← not (*#const* shifted left 16 bits) and *isrc2*
CC set if result is zero, cleared otherwise

bc *lbroff* ..... **Branch on CC**
IF CC = 1
THEN continue execution at *brx(lbroff)*
FI

bc.t *lbroff* ..... **Branch on CC, Taken**
IF CC = 1
THEN execute one more sequential instruction
continue execution at *brx(lbroff)*
ELSE skip next sequential instruction
FI

bla *isrc1ni, isrc2, sbroff* ..... **Branch on LCC and Add**
LCC-temp clear if *isrc2* < −*isrc1ni* (signed)
LCC-temp set if *isrc2* ≥ −*isrc1ni* (signed)
isrc2 ← *isrc1ni* + *isrc2*
Execute one more sequential instruction
IF LCC
THEN LCC ← LCC-temp
continue execution at *brx(sbroff)*
ELSE LCC ← LCC-temp
FI

bnc *lbroff* ..... **Branch on Not CC**
IF CC = 0
THEN continue execution at *brx(lbroff)*
FI

bnc.t *lbroff* ..... **Branch on Not CC, Taken**
IF CC = 0
THEN execute one more sequential instruction
continue execution at *brx(lbroff)*
ELSE skip next sequential instruction
FI

br *lbroff* ..... **Branch Direct Unconditionally**

Execute one more sequential instruction.
Continue execution at *brx(lbroff)*.

bri [*isrc1ni*] ..... **Branch Indirect Unconditionally**

```
Execute one more sequential instruction
IF      any trap bit in psr is set
THEN    copy PU to U, PIM to IM in psr
        clear trap bits
        IF      DS is set and DIM is reset
        THEN    enter dual-instruction mode after executing one
                  instruction in single-instruction mode
        ELSE    IF      DS is set and DIM is set
                THEN    enter single-instruction mode after executing one
                          instruction in dual-instruction mode
                ELSE    IF      DIM is set
                        THEN    enter dual-instruction mode
                                  for next two instructions
                        ELSE    enter single-instruction mode
                                  for next two instructions
                        FI
                FI
        FI
FI
```

Continue execution at address in *isrc1ni*
(The original contents of *isrc1ni* is used even if the next instruction modifies *isrc1ni*. Does not trap if *isrc1ni* is misaligned.)

bte *isrc1s, isrc2, sbroff* ..... **Branch If Equal**

```
IF      isrc1s = isrc2
THEN    continue execution at brx(sbroff)
FI
```

btne *isrc1s, isrc2, sbroff* ..... **Branch If Not Equal**

```
IF      isrc1s ≠ isrc2
THEN    continue execution at brx(sbroff)
FI
```

call *lbroff* ..... **Subroutine Call**

r1 ← address of next sequential instruction + 4 (+8 in dual mode)
Execute one more sequential instruction
Continue execution at *brx(lbroff)*

calli [*isrc1ni*] ..... **Indirect Subroutine Call**

r1 ← address of next sequential instruction + 4 (+8 in dual mode)
Execute one more sequential instruction
Continue execution at address in *isrc1ni*
(The original contents of *isrc1ni* is used even if the next instruction modifies *isrc1ni*. Does not trap if *isrc1ni* is misaligned. The register *isrc1ni* must not be **r1**.)

fadd.p *fsrc1, fsrc2, fdest* ..... **Floating-Point Add**

fdest ← *fsrc1* + *fsrc2*

faddp *fsrc1, fsrc2, fdest* ..... **Add with Pixel Merge**

fdest ← *fsrc1* + *fsrc2*
Shift and load MERGE register as defined in Table 8.2

faddz *fsrc1, fsrc2, fdest* ..... **Add with Z Merge**

fdest ← *fsrc1* + *fsrc2*
Shift MERGE right 16 and load fields 31..16 and 63..48

famov.r *fsrc1, fdest* ..... **Floating-Point Adder Move**

fdest ← *fsrc1*
Send *fsrc1* through the floating-point adder. (Preserves −0 (minus zero) when *fsrc1 is −0. fsrc2* must be coded as **f0** by the assembler.)

fiadd.w *fsrc1, fsrc2, fdest* ..... **Long-Integer Add**
fdest ← *fsrc1* + *fsrc2*

fisub.w *fsrc1, fsrc2, fdest* ..... **Long-Integer Subtract**
fdest ← *fsrc1* − *fsrc2*

fix.v *fsrc1, fdest* ..... **Floating-Point to Integer Conversion**
fdest ← 64- bit value with low-order 32 bits equal to integer part of *fsrc1* rounded

Floating-Point Load

fld.y *isrc1(isrc2), fdest* ..... **(Normal)**
fld.y *isrc1(isrc2)*++, *fdest* ..... **(Autoincrement)**
fdest ← mem.y (*isrc1* + *isrc2*)
IF autoincrement
THEN *isrc2* ← *isrc1* + *isrc2*
FI

Cache Flush

flush *#const(isrc2)* ..... **(Normal)**
flush *#const(isrc2)*++ ..... **(Autoincrement)**
Replace block in data cache with address (*#const* + *isrc2*).
Contents of block undefined.
IF autoincrement
THEN *isrc2* ← *#const* + *isrc2*
FI

fmlow.dd *fsrc1, fsrc2, fdest* ..... **Floating-Point Multiply Low**
fdest ← low-order 53 bits of *fsrc1* mantissa × *fsrc2* mantissa
fdest bit 53 ← most significant bit of mantissa

fmov.r *fsrc1, fdest* ..... **Floating-Point Reg-Reg Move**
Assembler pseudo-operation
fmov.ss *fsrc1, fdest* = **fiadd.ss** *fsrc1*, **f0,** *fdest*
fmov.dd *fsrc1, fdest* = **fiadd.dd** *fsrc1*, **f0,** *fdest*
fmov.sd *fsrc1, fdest* = **famov.sd** *fsrc1, fdest*
fmov.ds *fsrc1, fdest* = **famov.ds** *fsrc1, fdest*

fmul.p *fsrc1, fsrc2, fdest* ..... **Floating-Point Multiply**
fdest ← *fsrc1* × *fsrc2*

fnop ..... **Floating-Point No Operation**
Assembler pseudo-operation
fnop = **shrd r0, r0, r0**

form *fsrc1, fdest* ..... **OR with MERGE Register**
fdest ← *fsrc1* OR MERGE
MERGE ← 0

frcp.p *fsrc2, fdest* ..... **Floating-Point Reciprocal**
fdest ← 1/*fsrc2* with maximum mantissa error < 2^{-7}

frsqr.p *fsrc2, fdest* ..... **Floating-Point Reciprocal Square Root**
fdest ← 1/SQRT (*fsrc2*) with maximum mantissa error < 2^{-7}

Floating-Point Store

fst.y *fdest, isrc1(isrc2)* ..... **(Normal)**
fst.y *fdest, isrc1(isrc2)*++ ..... **(Autoincrement)**
mem.y (*isrc2* + *isrc1*) ← *fdest*
IF autoincrement
THEN *isrc2* ← *isrc1* + *isrc2*
FI

fsub.p *fsrc1, fsrc2, fdest* ..... **Floating-Point Subtract**
fdest ← *fsrc1* − *fsrc2*

ftrunc.v *fsrc1, fdest* ..... **Floating-Point to Integer Conversion**
fdest ← 64-bit value with low-order 32 bits equal to integer part of *fsrc1*

fxfr *fsrc1, idest* ..... **Transfer F-P to Integer Register**
idest ← *fsrc1*

fzchkl *fsrc1, fsrc2, fdest* ..... **32-Bit Z-Buffer Check**

Consider *fsrc1, fsrc2,* and *fdest* as arrays of two 32-bit
fields *fsrc1*(0)..*fsrc1*(1), *fsrc2*(0)..*fsrc2*(1), and *fdest*(0)..*fdest*(1)
where zero denotes the least-significant field.
PM ← PM shifted right by 2 bits
FOR i = 0 to 1
DO
PM [i + 6] ← *fsrc2*(i) ≤ *fsrc1*(i) (unsigned)
fdest(i) ← smaller of *fsrc2*(i) and *fsrc1*(i)
OD
MERGE ← 0

fzchks *fsrc1, fsrc2, fdest* ..... **16-Bit Z-Buffer Check**

Consider *fsrc1, fsrc2,* and *fdest* as arrays of four 16-bit
fields *fsrc1*(0)..*fsrc1*(3), *fsrc2*(0)..*fsrc2*(3), and *fdest*(0)..*fdest*(3)
where zero denotes the least-significant field.
PM ← PM shifted right by 4 bits
FOR i = 0 to 3
DO
PM [i + 4] ← *fsrc2*(i) ≤ *fsrc1*(i) (unsigned)
fdest(i) ← smaller of *fsrc2*(i) and *fsrc1*(i)
OD
MERGE ← 0

intovr ..... **Software Trap on Integer Overflow**

If OF in **epsr** = 1, generate trap with IT set in **psr**.

ixfr *isrc1ni, fdest* ..... **Transfer Integer to F-P Register**

fdest ← *isrc1ni*

ld.c *csrc2, idest* ..... **Load from Control Register**

idest ← *csrc2*

ld.x *isrc1*(*isrc2*), *idest* ..... **Load Integer**

idest ← *mem.x* (*isrc1* + *isrc2*)

lock ..... **Begin Interlocked Sequence**

Set BL in **dirbase**. The next load or store that misses the cache locks that location.
Disable interrupts until the bus is unlocked.

mov *isrc2, idest* ..... **Register-Register Move**

Assembler pseudo-operation
mov *isrc2, idest* = **shl r0,** *isrc2, idest*

mov *const32, idest* ..... **Constant-to-Register Move**

Assembler pseudo-operation
adds *l%const32*, **r0**, *idest*
... when *const32* < 0x8000

orh *h%const32*, **r0**, *idest*
or *l%const32, idest, idest*
... when *const32* ≥ 0x8000

nop ..... **Core-Unit No Operation**

Assembler pseudo-operation
nop = **shl r0, r0, r0**

or *isrc1, isrc2, idest* ..... **Logical OR**

idest ← *isrc1* OR *isrc2*
CC set if result is zero, cleared otherwise

orh *#const, isrc2, idest* ..... **Logical OR High**

idest ← (*#const* shifted left 16 bits) OR *isrc2*
CC set if result is zero, cleared otherwise

pfadd.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Add**
fdest ← last stage Adder result
Advance A pipeline one stage
A pipeline first stage ← *fsrc1* + *fsrc2*

pfaddp *fsrc1, fsrc2, fdest* ..... **Pipelined Add with Pixel Merge**
fdest ← last stage Graphics result
last stage Graphics result ← *fsrc1* + *fsrc2*
Shift and load MERGE register from last stage Graphics result as defined in Table 8.2

pfaddz *fsrc1, fsrc2, fdest* ..... **Pipelined Add with Z Merge**
fdest ← last stage Graphics result
last stage Graphics result ← *fsrc1* + *fsrc2*
Shift MERGE right 16 and load fields 31..16 and 63..48 from last stage Graphics result

pfam.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Add and Multiply**
fdest ← last stage Adder result
Advance A and M pipeline one stage (operands accessed before advancing pipeline)
A pipeline first stage ← A-op1 + A-op2
M pipeline first stage ← M-op1 × M-op2

pfamov.r *fsrc1, fdest* ..... **Pipelined Floating-Point Adder Move**
fdest ← last stage Adder result
Advance A pipeline one stage
A pipeline first stage ← *fsrc1*

pfeq.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Equal Compare**
fdest ← last stage Adder result
CC set if *fsrc1* = *fsrc2*, else cleared
Advance A pipeline one stage
A pipeline first stage is undefined, but no result exception occurs

pfgt.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Greather-Than Compare**
(Assembler clears R-bit of instruction)
fdest ← last stage Adder result
CC set if *fsrc1* > *fsrc2*, else cleared
Advance A pipeline one stage
A pipeline first stage is undefined, but no result exception occurs

pfiadd.w *fsrc1, fsrc2, fdest* ..... **Pipelined Long-Integer Add**
fdest ← last stage Graphics result
last stage Graphics result ← *fsrc1* + *fsrc2*

pfisub.w *fsrc1, fsrc2, fdest* ..... **Pipelined Long-Integer Subtract**
fdest ← last stage Graphics result
last stage Graphics result ← *fsrc1* − *fsrc2*

pfix.v *fsrc1, fdest* ..... **Pipelined Floating-Point to Integer Conversion**
fdest ← last stage Adder result
Advance A pipeline one stage
A pipeline first stage ← 64-bit value with low-order 32 bits
equal to integer part of *fsrc1* rounded

Pipelined Floating-Point Load

pfld.z *isrc1*(*isrc2*), *fdest* ..... **(Normal)**
pfld.z *isrc1*(*isrc2*)++, *fdest* ..... **(Autoincrement)**
fdest ← mem.z (third previous **pfld's** (*isrc1* + *isrc2*))
(where .z is precision of third previous **pfld.z)**
If autoincrement
THEN *isrc2* ← *isrc1* + *isrc2*
FI

pfle.p *fsrc1, fsrc2, fdest* ..... **Pipelined F-P Less-Than or Equal Compare**
Assembler pseudo-operation, identical to **pfgt.p** except that
assembler sets R-bit of instruction.
fdest ← last stage Adder result
CC clear if *fsrc1* ≤ *fsrc2*, else set
Advance A pipeline one stage
A pipeline first stage is undefined, but no result exception occurs

pfmam.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Add and Multiply**
fdest ← last stage Multiplier result
Advance A and M pipeline one stage (operands accessed before advancing pipeline)
A pipeline first stage ← A-op1 − A-op2
M pipeline first stage ← M-op1 × M-op2

pfmov.r *fsrc1, fdest* ..... **Pipelined Floating-Point Reg-Reg Move**
Assembler pseudo-operation
pfmov.ss *fsrc1, fdest* = **pfiadd.ss** *fsrc1*, **f0,** *fdest*
pfmov.dd *fsrc1, fdest* = **pfiadd.dd** *fsrc1*, **f0,** *fdest*
pfmov.sd *fsrc1, fdest* = **pfamov.sd** *fsrc1, fdest*
pfmov.ds *fsrc1, fdest* = **pfamov.ds** *fsrc1, fdest*

pfmsm.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Subtract and Multiply**
fdest ← last stage Multiplier result
Advance A and M pipeline one stage (operands accessed before advancing pipeline)
A pipeline first stage ← A-op1 − A-op2
M pipeline first stage ← M-op1 × M-op2

pfmul.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Multiply**
fdest ← last stage Multiplier result
Advance M pipeline one stage
M pipeline first stage ← *fsrc1* × *fsrc2*

pfmul3.dd *fsrc1, fsrc2, fdest* ..... **Three-Stage Pipelined Multiply**
fdest ← last stage Multiplier result
Advance 3-Stage M pipeline one stage
M pipeline first stage ← *fsrc1* × *fsrc2*

pform *fsrc1, fdest* ..... **Pipelined OR to MERGE Register**
fdest ← last stage Graphics result
last stage Graphics result ← *fsrc1* OR MERGE
MERGE ← 0

pfsm.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Subtract and Multiply**
fdest ← last stage Adder result
Advance A and M pipeline one stage (operands accessed before advancing pipeline)
A pipeline first stage ← A-op1 − A-op2
M pipeline first stage ← M-op1 × M-op2

pfsub.p *fsrc1, fsrc2, fdest* ..... **Pipelined Floating-Point Subtract**
fdest ← last stage Adder result
Advance A pipeline one stage
A pipeline first stage ← *fsrc1* + *fsrc2*

pftrunc.v *fsrc1, fdest* ..... **Pipelined Floating-Point to Integer Conversion**
fdest ← last stage Adder result
Advance A pipeline one stage
A pipeline first stage ← 64-bit value with low-order 32 bits
equal to integer part of *fsrc1*

pfzchkl *fsrc1, fsrc2, fdest* ..... **Pipelined 32-Bit Z-Buffer Check**
Consider *fsrc1, fsrc2,* and *fdest*, as arrays of two 32-bit
fields *fsrc1*(0)..*fsrc1*(1), *fsrc2*(0)..*fsrc2*(1), and *fdest*(0)..*fdest*(1)
where zero denotes the least significant field.
PM ← PM shifted right by 2 bits
FOR i = 0 to 1
DO
PM [i + 6] ← *fsrc2*(i) ≤ *fsrc1*(i) (unsigned)
fdest(i) ← last stage Graphics result
last stage Graphics result ← smaller of *fsrc2*(i) and *fsrc1*(i)
OD
MERGE ← 0

pfzchks *fsrc1, fsrc2, fdest* ..... **Pipelined 16-Bit Z-Buffer Check**

Consider *fsrc1, fsrc2,* and *fdest,* as arrays of four 16-bit
fields *fsrc1*(0)..*fsrc1*(3), *fsrc2*(0)..*fsrc2*(3), and *fdest*(0)..*fdest*(3)
where zero denotes the least significant field.
PM ← PM shifted right by 4 bits
FOR i = 0 to 3
DO
PM [i + 4] ← *fsrc2*(i) ≤ *fsrc1*(i) (unsigned)
fdest(i) ← last stage Graphics result
last stage Graphics result ← smaller of *fsrc2*(i) and *fsrc1*(i)
OD
MERGE ← 0

pst.d *fdest, #const(isrc2)* ..... **Pixel Store**

pst.d *fdest, #const(isrc2)+ +* ..... **Pixel Store Autoincrement**

Pixels enabled by PM in mem.d (*isrc2* + *#const*) ← *fdest*
Shift PM right by 8/pixel size (in bytes) bits
IF autoincrement
THEN *isrc2* ← *#const* + *isrc2*
FI

shl *isrc1, isrc2, idest* ..... **Shift Left**

idest ← *isrc2* shifted left by *isrc1* bits

shr *isrc1, isrc2, idest* ..... **Shift Right**

SC (in **psr**) ← *isrc1*
idest ← *isrc2* shifted right by *isrc1* bits

shra *isrc1, isrc2, idest* ..... **Shift Right Arithmetic**

idest ← *isrc2* arithmetically shifted right by *isrc1* bits

shrd *isrc1, isrc2, idest* ..... **Shift Right Double**

idest ← low-order 32 bits of *isrc1:isrc2* shifted right by SC bits

st.c *isrc1ni, csrc2* ..... **Store to Control Register**

csrc2 ← *isrc1ni*

st.x *isrc1ni, #const(isrc2)* ..... **Store Integer**

mem.x (isrc2 + *#const)* ← *isrc1ni*

subs *isrc1, isrc2, idest* ..... **Subtract Signed**

idest ← *isrc1* − *isrc2*
OF ← (bit 31 carry ≠ bit 30 carry)
CC set if *isrc2* > *isrc1* (signed)
CC clear if *isrc2* ≤ *isrc1* (signed)

subu *isrc1, isrc2, idest* ..... **Subtract Unsigned**

idest ← *isrc1* − *isrc2*
OF ← NOT (bit 31 carry)
CC ← bit 31 carry
(i.e. CC set if *isrc2* ≤ *isrc1* (unsigned)
CC clear if *isrc2* > *isrc1* (unsigned)

trap *isrc1ni, isrc2, idest* ..... **Software Trap**

Generate trap with IT set in **psr**

unlock ..... **End Interlocked Sequence**

Clear BL in **dirbase**. The next load or store unlocks the bus.
Enable interrupts after bus is unlocked.

xor *isrc1, isrc2, idest* ..... **Logical Exclusive OR**

idest ← *isrc1* XOR *isrc2*
CC set if result is zero, cleared otherwise

xorh *#const, isrc2, idest* ..... **Logical Exclusive OR High**

idest ← (*#const* shifted left 16 bit) XOR *isrc2*
CC set if result is zero, cleared otherwise

Table 8.2. FADDP MERGE Update

Pixel Size (from PS)	Fields Loaded From Result into MERGE	Right Shift Amount (Field Size)
8	63..56, 47..40, 31..24, 15..8	8
16	63..58, 47..42, 31..26, 15..10	6
32	63..56, 31..24	8

8.2 Instruction Format and Encoding

All instructions are 32 bits long and begin on a four-byte boundary. When operands are registers, the register encodings shown in Table 8.3 are used. There are two general core-instruction formats, REG-format and CTRL-format, as well as a separate format for floating-point instructions.

8.2.1 REG-FORMAT INSTRUCTIONS

Within the REG-format are several variations as shown in Figure 8.1. Table 8.4 gives the encodings for these instructions. One encoding is an escape code that defines yet another variation: the core escape instructions. Figure 8.2 shows the format of this group, and Table 8.5 shows the encodings.

In these instructions, the *src2* field selects one of the 32 integer registers (most instructions) or five control registers (**st.c** and **ld.c**). *Dest* selects one of the 32 integer registers (most instructions) or floating-point registers (**fld**, **fst**, **pfld**, **pst**, **ixfr**). For instructions where *src1* is optionally an immediate value, bit 26 of the opcode (I-bit) indicates whether *src1* is an immediate. If bit 26 is clear, an integer register is used; if bit 26 is set, *src1* is contained in the low-order 16 bits, except for **bte** and **btne** instructions. For **bte** and **btne**, the five-bit immediate value is contained in the *src1* field. For **st**, **bte**, **btne**, and **bla**, the upper five bits of the *offset* or *broffset* are contained in the *dest* field instead of *src1*, and the lower 11 bits of *offset* are the lower 11 bits of the instruction.

Table 8.3. Register Encoding

Register	Encoding
r0	0
.	.
.	.
.	.
r31	31
f0	0
.	.
.	.
.	.
f31	31
Fault Instruction	0
Processor Status	1
Directory Base	2
Data Breakpoint	3
Floating-Point Status	4
Extended Process Status	5

For **ld** and **st**, bits 28 and zero determine operand size as follows:

Bit 28	Bit 0	Operand Size
0	0	8-bits
0	1	8-bits
1	0	16-bits
1	1	32-bits

When *src1* is an immediate and bit 28 is set, bit zero of the immediate value is forced to zero.

For **fld**, **fst**, **pfld**, **pst**, and **flush**, bit 0 selects autoincrement addressing if set. For **fld**, **fst**, **pfld**, and **pst**, bits one and two select the operand size as follows:

Bit 1	Bit 2	Operand Size
0	0	64-bits
0	1	128-bits
1	0	32-bits
1	1	32-bits

When *src1* is an immediate value, bits zero and one of the immediate value are forced to zero to maintain alignment. When bit one of the immediate value is clear, bit two is also forced to zero.

For **flush**, bits one and two must be zero.

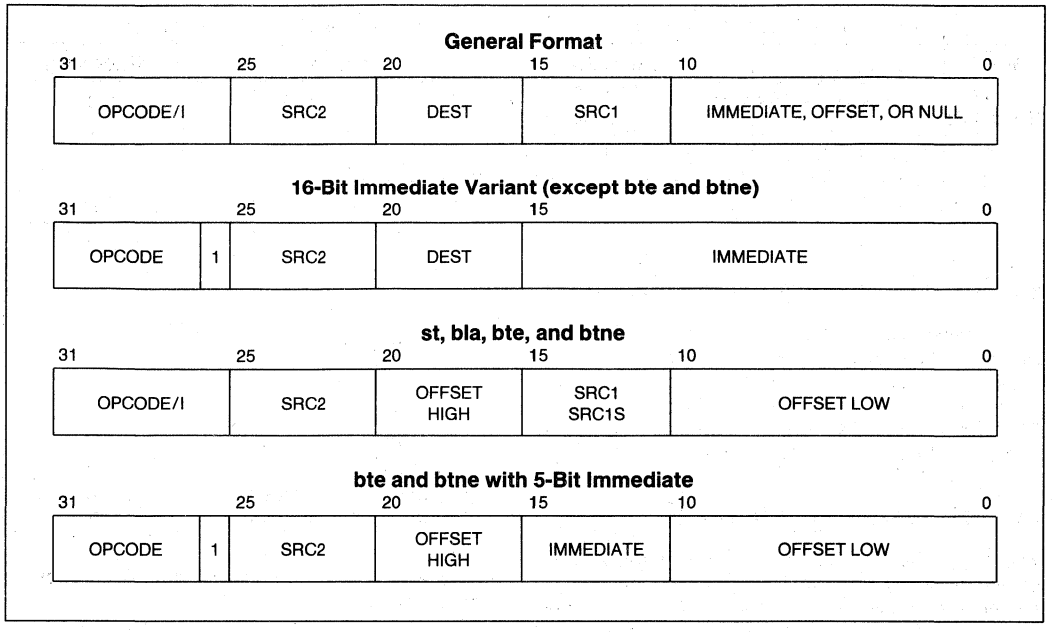


Figure 8.1. REG-Format Variations

Table 8.4. REG-Format Opcodes

		31					26
ld.x	Load Integer	0	0	0	L	0	I
st.x	Store Integer	0	0	0	L	1	1
ixfr	Integer to F-P Reg Transfer	0	0	0	0	1	0
	(reserved)	0	0	0	1	1	0
fld.x, fst.x	Load/Store F-P	0	0	1	0	LS	I
flush	Flush	0	0	1	1	0	1
pst.d	Pixel Store	0	0	1	1	1	1
ld.c, st.c	Load/Store Control Register	0	0	1	1	LS	0
bri	Branch Indirect	0	1	0	0	0	0
trap	Trap	0	1	0	0	0	1
	(Escape for F-P Unit)	0	1	0	0	1	0
	(Escape for Core Unit)	0	1	0	0	1	1
bte, btne	Branch Equal or Not Equal	0	1	0	1	E	I
pfld.y	Pipelined F-P Load	0	1	1	0	0	I
	(CTRL-Format Instructions)	0	1	1	x	x	x
addu, -s, subu, -s,	Add/Subtract	1	0	0	SO	AS	I
shl, shr	Logical Shift	1	0	1	0	LR	I
shrd	Double Shift	1	0	1	1	0	0
bla	Branch LCC Set and Add	1	0	1	1	0	1
shra	Arithmetic Shift	1	0	1	1	1	I
and(h)	AND	1	1	0	0	H	I
andnot(h)	ANDNOT	1	1	0	1	H	I
or(h)	OR	1	1	1	0	H	I
xor(h)	XOR	1	1	1	1	H	I
	(reserved)	1	1	x	x	1	0

L Integer Length
- 0 —8 bits
- 1 —16 or 32 bits (selected by bit 0)

LS Load/Store
- 0 —Load
- 1 —Store

SO Signed/Ordinal
- 0 —Ordinal
- 1 —Signed

H High
- 0 —**and, or, andnot, xor**
- 1 —**andh, orh, andnoth, xorh**

AS Add/Subtract
- 0 —Add
- 1 —Subtract

LR Left/Right
- 0 —Left Shift
- 1 —Right Shift

E Equal
- 0 —Branch on Not Equal
- 1 —Branch on Equal

I Immediate
- 0 —*src1* is register
- 1 —*src1* is immediate

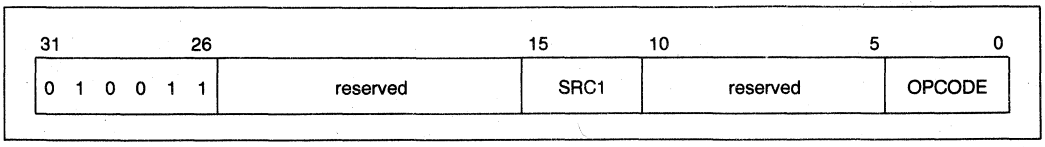


Figure 8.2. Core Escape Instruction Format

Table 8.5. Core Escape Opcodes

		4				0
	(reserved)	0	0	0	0	0
lock	Begin Interlocked Sequence	0	0	0	0	1
calli	Indirect Subroutine Call	0	0	0	1	0
	(reserved)	0	0	0	1	1
intovr	Trap on Integer Overflow	0	0	1	0	0
	(reserved)	0	0	1	0	1
	(reserved)	0	0	1	1	0
unlock	End Interlocked Sequence	0	0	1	1	1
	(reserved)	0	1	x	x	x
	(reserved)	1	0	x	x	x
	(reserved)	1	1	x	x	x

8.2.2 CTRL-FORMAT INSTRUCTIONS

The CTRL instructions do not refer to registers, so instead of the register fields, they have a 26-bit relative branch offset. Figure 8.3 shows the format of these instructions and Table 8.6 defines the encodings.

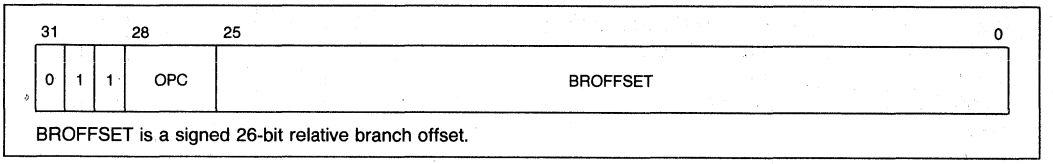


Figure 8.3. CTRL Instruction Format

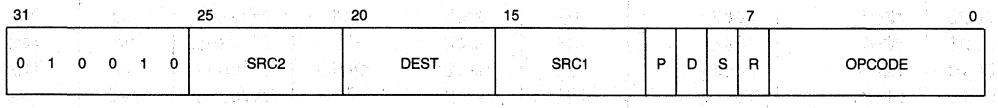
Table 8.6. CTRL-Format Opcodes

		28		**26**
	(reserved)	0	0	0
	(reserved)	0	0	1
br	Branch Direct	0	1	0
call	Call	0	1	1
bc(.t)	Branch on CC Set	1	0	T
bnc(.t)	Branch on CC Clear	1	1	T

T Taken
0 —**bc** or **bnc**
1 —**bc.t** or **bnc.t**

8.2.3 FLOATING-POINT INSTRUCTIONS

The floating-point instructions also constitute an escape series. All these instructions begin with the bit sequence 010010. Figure 8.4 shows the format of the floating point instructions, and Table 8.7 gives the encodings. Within the dual-operation instructions is a subcode DPC whose values are given in Table 8.8 along with the mnemonic that corresponds to each.



SRC1, SRC2 —Source; one of 32 floating-point registers
DEST —Destination register
(instructions other than **fxfr**) one of 32 floating-point registers
(**fxfr**) one of 32 integer registers

P Pipelining
1 —Pipelined instruction mode
0 —Scalar instruction mode

D Dual-Instruction Mode
1 —Dual-instruction mode
0 —Single-instruction mode

S Source Precision
1 —Double-precision source operands
0 —Single-precision source operands

R Result Precision
1 —Double-precision result
0 —Single-precision result

Figure 8.4. Floating-Point Instruction Encoding

Table 8.7. Floating-Point Opcodes

		6						0
pfam	Add and Multiply*							
pfmam	Multiply with Add*	0	0	0	DPC			
pfsm	Subtract and Multiply*							
pfmsm	Multiply with Subtract*	0	0	1	DPC			
(p)fmul	Multiply	0	1	0	0	0	0	0
fmlow	Multiply Low	0	1	0	0	0	0	1
frcp	Reciprocal	0	1	0	0	0	1	0
frsqr	Reciprocal Square Root	0	1	0	0	0	1	1
pfmul3.dd	3-Stage Pipelined Multiply	0	1	0	0	1	0	0
(p)fadd	Add	0	1	1	0	0	0	0
(p)fsub	Subtract	0	1	1	0	0	0	1
(p)fix	Fix	0	1	1	0	0	1	0
(p)famov	Adder Move	0	1	1	0	0	1	1
pfgt/pfle**	Greater Than	0	1	1	0	1	0	0
pfeq	Equal	0	1	1	0	1	0	1
(p)ftrunc	Truncate	0	1	1	1	0	1	0
fxfr	Transfer to Integer Register	1	0	0	0	0	0	0
(p)fiadd	Long-Integer Add	1	0	0	1	0	0	1
(p)fisub	Long-Integer Subtract	1	0	0	1	1	0	1
(p)fzchkl	Z-Check Long	1	0	1	0	1	1	1
(p)fzchks	Z-Check Short	1	0	1	1	1	1	1
(p)faddp	Add with Pixel Merge	1	0	1	0	0	0	0
(p)faddz	Add with Z Merge	1	0	1	0	0	0	1
(p)form	OR with MERGE Register	1	0	1	1	0	1	0

*pfam and pfsm have P-bit set; pfmam and pfmsm have P-bit clear.
pfgt has R bit cleared; **pfle has R bit set.

NOTE:
All opcodes not shown are **reserved**.

The following table shows the opcode mnemonics that generate the various encodings of DPC and explains each encoding.

Table 8.8. DPC Encoding

DPC	PFAM Mnemonic	PFSM Mnemonic	M-Unit op1	M-Unit op2	A-Unit op1	A-Unit op2	T Load	K Load*
0000	**r2p1**	**r2s1**	KR	src2	src1	M result	No	No
0001	**r2pt**	**r2st**	KR	src2	T	M result	No	Yes
0010	**r2ap1**	**r2as1**	KR	src2	src1	A result	Yes	No
0011	**r2apt**	**r2ast**	KR	src2	T	A result	Yes	Yes
0100	**i2p1**	**i2s1**	KI	src2	src1	M result	No	No
0101	**i2pt**	**i2st**	KI	src2	T	M result	No	Yes
0110	**i2ap1**	**i2as1**	KI	src2	src1	A result	Yes	No
0111	**i2apt**	**i2ast**	KI	src2	T	A result	Yes	Yes
1000	**rat1p2**	**rat1s2**	KR	A result	src1	src2	Yes	No
1001	**m12apm**	**m12asm**	src1	src2	A result	M result	No	No
1010	**ra1p2**	**ra1s2**	KR	A result	src1	src2	No	No
1011	**m12ttpa**	**m12ttsa**	src1	src2	T	A result	Yes	No
1100	**iat1p2**	**iat1s2**	KI	A result	src1	src2	Yes	No
1101	**m12tpm**	**m12tsm**	src1	src2	T	M result	No	No
1110	**ia1p2**	**ia1s2**	KI	A result	src1	src2	No	No
1111	**m12tpa**	**m12tsa**	src1	src2	T	A result	No	No

DPC	PFMAM Mnemonic	PFMSM Mnemonic	M-Unit op1	M-Unit op2	A-Unit op1	A-Unit op2	T Load	K Load*
0000	**mr2p1**	**mr2s1**	KR	src2	src1	M result	No	No
0001	**mr2pt**	**mr2st**	KR	src2	T	M result	No	Yes
0010	**mr2mp1**	**mr2ms1**	KR	src2	src1	M result	Yes	No
0011	**mr2mpt**	**mr2mst**	KR	src2	T	M result	Yes	Yes
0100	**mi2p1**	**mi2s1**	KI	src2	src1	M result	No	No
0101	**mi2pt**	**mi2st**	KI	src2	T	M result	No	Yes
0110	**mi2mp1**	**mi2ms1**	KI	src2	src1	M result	Yes	No
0111	**mi2mpt**	**mi2mst**	KI	src2	T	M result	Yes	Yes
1000	**mrmt1p2**	**mrmt1s2**	KR	M result	src1	src2	Yes	No
1001	**mm12mpm**	**mm12msm**	src1	src2	M result	M result	No	No
1010	**mrm1p2**	**mrm1s2**	KR	M result	src1	src2	No	No
1011	**mm12ttpm**	**mm12ttsm**	src1	src2	T	A result	Yes	No
1100	**mimt1p2**	**mimt1s2**	KI	M result	src1	src2	Yes	No
1101	**mm12tpm**	**mm12tsm**	src1	src2	T	M result	No	No
1110	**mim1p2**	**mim1s2**	KI	M result	src1	src2	No	No
1111	Intel-Reserved							

*If K-load is set, KR is loaded when operand-1 of the multiplier is KR; KI is loaded when operand-1 of the multiplier is KI.

8.3 Instruction Timings

i860 microprocessor instructions take one clock to execute unless a freeze condition is invoked. Freeze conditions and their associated delays are shown in the table below. Freezes due to multiple simultaneous cache misses result in a delay that is the sum of the delays for processing each miss by itself. Other multiple freeze conditions usually add only the delay of the longest individual freeze.

Freeze Condition	Delay
Instruction-cache miss	Number of clocks to read instruction (from ADS clock to first READY# clock) plus time to last READY# of block when jump or freeze occurs during miss processing plus two clocks if data-cache being accessed when instruction-cache miss occurs.
Reference to destination of **ld** instruction that misses	One plus number of clocks to read data (from ADS# clock to first READY# clock) minus number of instructions executed since load (not counting instruction that references load destination)
fld miss	One plus number of clocks until first READY# returned (for 32- or 64-bit read cycles) or until second READY# returned (for 128-bit **fld.q** read cycles)
call, calli, ixfr, fxfr, ld.c, or **st.c** and data cache load miss processing in progress	One plus number of clocks until first READY# returned (for 64-bit read cycles) or until second READY# returned (for 128-bit **fld.q** read cycles)
ld/st/pfld/fld/fst and data cache load miss processing in progress	One plus number of clocks until last READY# returned
Reference to *dest* of **ld, call, calli, fxfr,** or **ld.c** in the next instruction. (*Dest* of **call** and **calli** is **r1**.)	One clock

6

Freeze Condition	Delay
Reference to *dest* of **fld/pfld/ixfr** in the next two instructions	Two clocks in the first instruction; one in the second instruction
bc/bnc/bc.t/bnc.t following **fadd/fsub/pfeg/pfgt**	One clock
Fsrc1 of multiplier operation refers to result of previous operation	One clock
Floating-point operation or graphics-unit instruction or **fst,** and scalar operation in progress other than **frcp** or **frsqr**	If the scalar operation is **fadd, fix, fmlow, fmul.ss, fmul.sd, ftrunc,** or **fsub,** two minus the number of instructions (or dual-mode pairs) already executed after the scalar operation. If the scalar operation is **fmul.dd,** three minus the number of instructions (or dual-mode pairs) executed after it. Add one if either or both of these two situations occur: 1. There is an overlap between the result register of the previous scalar operation and the source of the floating-point operation, and the destination precision of the scalar operation is different than the source precision of the floating-point operation. 2. The floating-point operation is pipelined and its destination is not **f0.** There is no delay if the result is negative.
Multiplier operation preceded by a double-precision multiply	One clock
Multiplier operation with data pattern requiring extra rounding operation	One clock
TLB miss	Five plus the number of clocks to finish two reads plus the number of clocks to set A-bits (if necessary)
pfld when three **pfld**'s are outstanding	One plus the number of clocks to return data from first **pfld**
pfld hits in the data cache	Two plus the number of clocks to finish all outstanding accesses
st, pst or **fst** miss, **ld** miss, or **flush** with modified block when store path full (two stores or one 256-bit write-back internally waiting for bus plus external bus pipeline full)	One plus the number of clocks until READY# active on next 64-bit write cycle or second READY# of next 128-bit write cycle.
ld, fld, pfld, st, pst, or **fst** when address path full (one address internally waiting for bus plus external bus pipeline full)	Number of clocks until next nonrepeated address can be issued (i.e., an address that is not the 2nd–4th cycle of a cache fill, the 2nd–8th cycle of a CS8 mode instruction fetch, nor the 2nd cycle of a 128-bit write)
ld/fld following **st/fst** hit	One clock

Freeze Condition	Delay
Delayed branch not taken	One clock
Nondelayed branch taken:	
bc, bnc	One clock
bte, btne	Two clocks
Indirect branch **bri** or call **calli**	One clock
st.c	Two clocks
Result of graphics-unit instruction (other than **fmov.dd**) used in next instruction when the next instruction is an adder- or multiplier-unit instruction	One clock
Result of graphics-unit instruction used in next instruction when the next instruction is a graphics-unit instruction	One clock
flush followed by **flush**	Three clocks minus the number of instructions between the two **flush** instructions. There is no delay if the result is negative.
fst or **pst** followed by pipelined floating-point operation that overwrites the register being stored	One clock

8.4 Instruction Characteristics

The following table lists some of the characteristics of each instruction. The characteristics are:

- What processing unit executes the instruction. The codes for processing units are:
  - A Floating-point adder unit
  - E Core execution unit
  - G Graphics unit
  - M Floating-point multiplier unit
- Whether the instruction is pipelined or not. A *P* indicates that the instruction is pipelined.
- Whether the instruction is a delayed branch instruction. A *D* marks the delayed branches.
- Whether the instruction changes the condition code CC. A *CC* marks those instructions that change CC.
- Which faults can be caused by the instruction. The codes used for exceptions are:
  - IT Instruction Fault
  - SE Floating-Point Source Exception
  - RE Floating-Point Result Exception, including overflow, underflow, inexact result
  - DAT Data Access Fault

  Note that this is not the same as specifying at which instructions faults may be reported. A result exception is reported on the subsequent floating-point instruction, **pst, fst,** or sometimes **fld, pfld**, and **ixfr**.

  The instruction access fault IAT and the interrupt trap IN are not shown in the table because they can occur for any instruction.
- Performance notes. These comments regarding optimum performance are recommendations only. If these recommendations are not followed, the i860 microprocessor automatically waits the necessary number of clocks to satisfy internal hardware requirements. The following notes define the numeric codes that appear in the instruction table:
  1. The following instruction should not be a conditional branch (**bc**, **bnc**, **bc.t**, or **bnc.t**).
  2. The destination should not be a source operand of the next two instructions.
  3. A load should not directly follow a store that is expected to hit in the data cache.
  4. When the prior instruction is scalar, *fsrc1* should not be the same as the *fdest* of the prior operation.
  5. The *fdest* should not reference the destination of the next instruction if that instruction is a pipelined floating-point operation.
  6. The destination should not be a source operand of the next instruction. (For **call** and **calli**, the destination is **r1**.)
  7. When the prior operation is scalar and multiplier *op1* is *fsrc1*, *fsrc2* should not be the same as the *fdest* of the prior operation.
  8. When the prior operation is scalar, *fsrc1* and *fsrc2* of the current operation should not be the same as *fdest* of the prior operation.
  9. A **pfld** should not immediately follow a **pfld**.
- Programming restrictions. These indicate combinations of conditions that must be avoided by programmers, assemblers, and compilers. The following notes define the alphabetic codes that appear in the instruction table:
  a. The sequential instruction following a delayed control-transfer instruction may not be another control-transfer instruction (except in the case of external interrupts), nor a trap instruction, nor the target of a control-transfer instruction.
  b. When using a **bri** to return from a trap handler, programmers should take care to prevent traps from occurring on that or on the next sequential instruction. IM should be zero (interrupts disabled) when the **bri** is executed.
  c. If *fdest* is not zero, *fsrc1* must not be the same as *fdest*.
  d. When *fsrc1* goes to the multiplier *op1*, KR, or KI, *fsrc1* must not be the same as *fdest*.
  e. If *fdest* is not zero, *fsrc1* and *fsrc2* must not be the same as *fdest*.
  f. *isrc1* must not be the same as *isrc2* for the autoincrementing form of this instruction.
  g. *isrc1* must not be the same as *isrc2*.

Table 8.9 Instruction Characteristics

Instruction	Execution Unit	Pipelined? Delayed?	Sets CC?	Faults	Performance Notes	Programming Restrictions
adds	E		CC		1	
addu	E		CC		1	
and	E		CC			
andh	E		CC			
andnot	E		CC			
andnoth	E		CC			
bc	E					
bc.t	E	D				a
bla	E	D				a, g
bnc	E					
bnc.t	E	D				a
br	E	D				a
bri	E	D				a, b
bte	E					
btne	E					
call	E	D			6	a
calli	E	D			6	a
fadd.p	A			SE, RE		
faddp	G				8	
faddz	G				8	
famov.r	A			SE, RE		
fiadd.z	G				8	
fisub.z	G				8	
fix.p	A			SE, RE		
fld.y	E			DAT	2, 3	f
flush	E					
fmlow.p	M				4	
fmul.p	M			SE, RE	4	
form	G				8	
frep.p	M			SE, RE		
frsqr.p	M			SE, RE		
fst.y	E			DAT	5	f
fsub.p	A			SE, RE		
ftrunc.p	A			SE, RE		
fxfr	G				6, 8	
fzchkl	G				8	
fzchks	G				8	
intovr	E			IT		
ixfr	E				2	
ld.c	E					
ld.x	E			DAT	6	
or	E		CC			
orh	E		CC			
pfadd.p	A	P		SE, RE		
pfaddp	G	P			8	e

Instruction	Execution Unit	Pipelined? Delayed?	Sets CC?	Faults	Performance Notes	Programming Restrictions
pfaddz	G	P			8	e
pfam.p	A&M	P		SE, RE	7	d
pfamov.r	A	P		SE, RE		
pfeq.p	A	P	CC	SE	1	
pfgt.p	A	P	CC	SE	1	
pfiadd.z	G	P			8	e
pfisub.z	G	P			8	e
pfix.p	A	P		SE, RE		
pfld.z	E	P		DAT	2, 9	f
pfmam.p	A & M	P		SE, RE	7	d
pfmsm.p	A & M	P		SE, RE	7	d
pfmul.p	M	P		SE, RE	4	c
pfmul3.dd	M	P		SE, RE	4	c
pform	G	P			8	e
pfsm.p	A&M	P		SE, RE	7	d
pfsub.p	A	P		SE, RE		
pftrunc.p	A	P		SE, RE		
pfzchkl	G	P			8	
pfzchks	G	P			8	
pst.d	E			DAT		f
shl	E					
shr	E					
shra	E					
shrd	E					
st.c	E					
st.x	E			DAT		
subs	E		CC		1	
subu	E		CC		1	
trap	E			IT		
xor	E		CC			
xorh	E		CC			

DATA SHEET REVISION REVIEW

The following list represents the key differences between version 002 and version 001 of the i860 Microprocessor Data Sheet.

1. Big-endian description in section 2.3 has been expanded.
2. Bit 17 of the Extended Processor Status Register (EPSR) is the INT bit which reflects the value on the interrupt pin (INT), as described in section 2.2.4 entitled "EXTENDED PROCESSOR STATUS REGISTER". This is a documentation update only.
3. The cacheability of a page is controlled by NOR'ing the value of the CD, WT bits and the KEN# input pin, as described in section 2.5 entitled "Caching and Cache Flushing" and section 3.1.14 entitled "Cache Enable (KEN#)". This is a documentation update only.
4. The NOTE section in section 2.5 entitled "Caching and Cache Flushing" has been updated to clarify the paging requirement on changing the DTB field in the **dirbase** register.
5. Information on register encoding is added in section 8.2 entitled "Instruction Format and Encoding". This is a documentation update only.

The following list represents the key differences between version 003 and version 002 of the i860 Microprocessor Data Sheet.

Specification Changes:

1. Specification changes for improved AC performance are in section 7.3.
2. HOLD is acknowledged during locked bus cycles. See section 3.1.8.
3. Additional paths have been added to the bus state diagram to allow direct transitions from states T12 and T11 to state TH. See Figures 4.1 and 4.10.
4. Two new instructions, **(p)famov.r,** have been added. These replace **(p)fadd.ds** and **(p)fadd.sd** in the assembler pseudo-ops **(p)fmov.r**. These changes are in section 8.1 and tables 2.7, 8.7, and 8.9.

Documentation Changes:

1. Big and little endian description has been expanded in sections 2.2.2, 2.3, and Figure 2.8.
2. The actions and explanations of the **lock, unlock,** and **st.c dirbase** changing the BL bit have been updated in sections 2.2.4, 3.1.5, 3.1.8, 4.3.4, 4.3.5, and 8.1.
3. The explanation of the AA and MA bits of the **fpsr** have been expanded in section 2.2.8.
4. The explanation of the WT bit of the Page Table Entries has been expanded in sections 2.4.4.4 and 2.5.
5. A change concerning the locking of the bus during address translation is explained in sections 2.4.5 and 2.8.5.
6. A further explanation on when to flush the data cache is given in section 2.5.
7. The explanation of the floating point multiplier pipeline has been expanded in section 2.6.1.
8. The explanation of BREQ has been expanded in section 3.1.4 and Figure 4.1.
9. The explanation of result exceptions has been expanded in sections 2.8 and 3.2.
10. Instruction fetch identification has been clarified in section 3.1.6 and table 3.2.
11. Bus cycle diagrams in Figures 4.7, 4.8, and 4.10 have been clarified/corrected.
12. Precision specification **.r** has been added to section 8.0 and table 8.1.
13. In section 8.4, performance note 9 has been added, programming restriction d has been changed, and programming restriction f has been added. Table 8.9 has been updated to reflect these changes.
14. The description of testability has changed in sections 3.3. and 3.3.2. RESET and HOLD must be asserted by the tester to force the chip outputs to float (tri-state).

The following list represents the major differences between this version and version 003 of the i860 Microprocessor Data Sheet:

Section 2.2.4	The explanation of the **WP** bit of the **epsr** has been expanded.
Section 2.8.2	More information on the instruction trap has been added.
Section 2.8.4	The instruction access trap has been clarified.
Section 2.8.7	The values of registers after a reset trap have been specified.
Section 3.1.4	BREQ timing has been clarified.

Section 3.1.5 The calculation of interrupt latency has been corrected.

Section 3.1.6 The description of the byte-enable signals has been expanded.

Section 3.1.8 The relation between the **lock** instruction and the LOCK# signal has been clarified. The BL bit should no longer be changed by writing to the **dirbase** register.

Section 6.0 The thermal specifications have been updated.

Section 7.3 The A.C. characteristics for CLK have changed.

Section 7.3 Advance timing information for the 50 MHz clock rate has been added. These timings are subject to change without notice. Contact Intel Corporation for design-in information.

Section 8.0 The operand naming conventions have improved.

Section 8.2.1 The encoding of the **flush** instuction has been corrected.

The following list represents the major differences between this version and version -003 of the i860 Microprocessor Data Sheet:

Section 2.2.4 The explanation of the WP bit of the **espr** has been expanded.

Section 2.8.2 More information on the instruction trap has been added.

Section 2.8.4 The instruction access trap has been clarified.

Section 2.8.7 The values of registers after a reset trap have been specified.

Section 3.1.4 BREQ timing has been clarified.

Section 3.1.5 The calculation of interrupt latency has bee corrected.

Section 3.1.6 The description of the byte-enable signals has been expanded.

Section 3.1.8 The relation between the **lock** instruction and the LOCK# signal has been clarified. The BL bit should no longer be changed by writing to the **dirbase** register.

Section 6.0 The thermal specifications have been updated.

Section 7.3 The A.C. Characteristics for CLK have changed.

Section 7.3 Advance timing information for the 50 MHz clock rate has been added. These timings are subject to change without notice.

Section 8.0 The operand naming conventions have improved.

Section 8.2.1 The encoding of the **flush** instruction has been corrected.

Section 8.3 The data-dependent multiplier freeze has been eliminated. Other freeze conditions have been corrected or clarified.

intel®

APPLICATION NOTE

AP-434

November 1989

Using i860™ Microprocessor Graphics Instructions for 3-D Rendering

6

Order Number: 240856-001

USING i860™ MICROPROCESSOR GRAPHICS INSTRUCTIONS FOR 3-D RENDERING

CONTENTS

PAGE

INTRODUCTION 6-84

1.0 3-D RENDERING 6-84

2.0 DISTANCE INTERPOLATION 6-86

3.0 COLOR INTERPOLATION 6-88

4.0 BOUNDARY CONDITIONS 6-89

4.1 Z-Buffer Masking 6-89

4.2 Accumulator Initialization 6-91

5.0 THE INNER LOOP 6-91

6.0 ALTERNATIVE IMPLEMENTATIONS 6-95

FIGURES

PAGE

Figure 1 Z-BUFFER Interpolation 6-86
Figure 2 **faddz** Operands 6-87
Figure 3 Pixel Interpolation for Gouraud Shading 6-88
Figure 4 **faddp** Operands 6-89

TABLES

PAGE

Table 1 **faddz** Visualization 6-87
Table 2 Accumulator Initial Values 6-91
Table 3 Accumulator Initialization Table 6-91

EXAMPLES

PAGE

Example 1: Setting Pixel Size 6-84
Example 2: Register Assignments 6-85
Example 3: Construction of Z Interpolants 6-88
Example 4: Construction of Color Interpolants 6-89
Example 5: Z Mask Procedure 6-90
Example 6: Accumulator Initialization 6-92
Example 7: 3-D Rendering (1 of 2) 6-93
Example 7: 3-D Rendering (2 of 2) 6-94
Example 8: Inner Loop of Renderers for Two Pixel Sizes 6-95

Introduction

The i860™ 64-bit microprocessor is a general-purpose CPU with on-chip integer unit, floating point, memory management, caches, and graphics. The i860 microprocessor supports 3-D graphics software with the following functions:

1. Hidden surface elimination
2. Distance interpolation
3. Intensity interpolation for 3-D shading

The **fzchks** (Z-buffer Check) and **pst** (Pixel Store) instructions expedite hidden surface elimination. Distance interpolation is accomplished with **faddz** (Add with Z merge), and intensity interpolation occurs with **faddp** (Add with Pixel Merge). The purpose of this application note is to illustrate the intended use of these instructions in a manner independent of any graphics environment in which the instructions might be used. It is not the purpose of this application note to present the most efficient instruction sequences. While the inner loop of Example 7 has as few instructions as logically possible, the other examples are intended to present general concepts, not optimum implementations. Tuning for maximum performance depends on the specific environment.

This application note assumes familiarity with the *i860™ 64-bit Microprocessor Programmer's Reference Manual* (Intel order number 240329); the i860 microprocessor instructions for graphics are detailed in section 6.6.

1.0 3-D RENDERING

This series of examples are routines that might be used at the lowest level of a graphics software system to convert a machine-independent description of a 3-D image into values for the frame buffer of a color video display. Typically, higher-level graphics routines represent an object as a set of polygons that together roughly describe the surfaces of the objects to be displayed. The graphics system maintains a database that describes these polygons in terms of their colors, properties of reflectance or translucence, and the locations in 3-D space of their vertices. Due to the roughness of the representation, the amount of information in the database is considerably less than that which must be delivered to the video display. A rendering procedure, such as Example 7, uses interpolation to derive the detailed information needed for each pixel in the graphics frame buffer. The rendering procedure also performs pixel-by-pixel hidden-surface elimination.

The focus of this series of examples is Example 7, which operates on a segment of a scan line. The segment is bounded by two points of given location and color: from point (*X1, Y0, Z1*) with color intensities *Red1, Grn1, Blu1* to point (*X2, Y0, Z2*) with color intensities *Red2, Grn2, Blu2*. The points and color intensities are determined by higher-level graphics software. The points represent the intersection of the scan line with two edges of the projected image of a polygon. For a given scan line, the rendering procedure is executed once for each polygon that projects onto that scan line. The higher-level graphics software is responsible for orienting the objects with respect to the viewer, for making perspective calculations, for scaling, and for determining the amount of light that falls on each polygon vertex.

The 16-bit pixel format is used, giving ample resolution for color shading: 2^6 intensity values for red, 2^6 intensity values for green, and 2^4 intensity values for blue. Example 1 shows how to set the pixel size. For hidden-surface elimination, the Z-buffer (or depth buffer) technique is employed, each Z value having a resolution of 16-bits.

Because the examples presented here use almost all of the registers of the i860 microprocessor, the registers are given symbolic names, as defined by Example 2. In a real application, it is likely that some of the inputs to the rendering procedure would be passed in floating-point registers instead of the integer registers employed here. The register allocation shown in Example 2 simplifies the examples by avoiding the need to use any register for multiple purposes.

```
// SET PIXEL SIZE TO 16
    ld.c        psr,    Ra    // Work on psr
    andnoth     0x00C0, Ra, Ra// Clear PS
    orh         0x0040, Ra, Ra// PS = 16-bit pixels
    st.c        Ra,     psr   //
```

Example 1. Setting Pixel Size

```
// REGISTER DEFINITIONS FOR RENDERING PROCEDURE
//      INTEGER LOCALS
    Ra    = r4    // Temporary
    Rb    = r5    // Temporary
    Rc    = r6    // Temporary
    Rd    = r7    // Temporary
//      INTEGER INPUTS
    X1    = r16   // X coordinate of starting point of line segment in pixels
    dX    = r17   // Width of scan line segment in number of pixels
    ZBP   = r18   // Z-buffer pointer to the current line segment
    Z1    = r19   // Initial Z value, fixed-point 16.16 format
    mZ    = r20   // Z slope, fixed-point 16.16 format
    FBP   = r21   // Graphics frame buffer pointer to the current line segment
    Red1  = r22   // Initial red intensity, fixed-point 6.10 format, plus .5
    Grn1  = r23   // Initial green intensity, fixed-point 6.10 format, plus .5
    Blu1  = r24   // Initial blue intensity, fixed-point 6.10 format, plus .5
    mR    = r25   // Red slope, fixed-point 6.10 format
    mG    = r26   // Green slope, fixed-point 6.10 format
    mB    = r27   // Blue slope, fixed-point 6.10 format
//      REAL LOCALS
    aZ    = f2    // Accumulated Z values
    aZh   = f3    //
    iZ1   = f4    // Z interpolant, coefficient 1.0
    iZ1h  = f5    //
    iZ3   = f6    // Z interpolant, coefficient 3.0
    iZ3h  = f7    //
    oldz  = f8    // Original values from the Z-buffer
    newz  = f10   // New Z-buffer values
    newzh = f11   //
    newi  = f12   // New pixel values
    iR    = f14   // Red interpolant, coefficient 4.0
    iRh   = f15   //
    aR    = f16   // Accumulated red intensities
    aRh   = f17   //
    iG    = f18   // Green interpolant, coefficient 4.0
    iGh   = f19   //
    aG    = f20   // Accumulated green intensities
    aGh   = f21   //
    iB    = f22   // Blue interpolant, coefficient 4.0
    iBh   = f23   //
    aB    = f24   // Accumulated blue intensities
    aBh   = f25   //
    lZmask   = f26  // left-end Z mask
    lZmaskh  = f27  //
    rZmask   = f28  // right-end Z mask
    rZmaskh  = f29  //
```

Example 2. Register Assignments

2.0 DISTANCE INTERPOLATION

To perform hidden surface elimination at each pixel, the rendering routine first interpolates the value of Z at each pixel. Distance interpolation consists of calculating the slope of Z over the given line segment, then increasing the Z value of each successive pixel by that amount, starting from *X1*. The width of the line segment in pixels is . . .

$$dX = X2 - X1$$

Calculate the reciprocal of *dX*:

$$RdX = 1/dX$$

The value of *dX* is used several times as a divisor. It is most efficient to calculate its reciprocal once, then, instead of dividing by *dX*, multiply by *RdX*. The slope of Z is . . .

$$mZ = (Z2 - Z1)*RdX$$

Because each polygon is a plane, the value of *mZ* is constant for all scan lines that intersect the polygon; therefore *mZ* needs to be calculated only once for each polygon. Example 7 assumes that *dX* and *mZ* have already been calculated, and all that remains is to apply *mZ* to successive pixels. Let *Z(Xn)* be the Z value at pixel *Xn*. Then . . .

$$Z(X1) = Z1$$
$$Z(X1 + 1) = Z1 + mZ$$
$$Z(X1 + 2) = Z1 + 2*mZ$$
$$\vdots$$
$$Z(X1 + N) = Z1 + N*mZ$$
$$\vdots$$
$$Z(X1 + dX) = Z1 + dX*mZ = Z(X2)$$

Figure 1 illustrates this Z-value interpolation.

The **faddz** instruction helps to perform the above calculations 64 bits at a time. Because a Z value is 16 bits wide, Example 7 operates on the Z buffer in groups of four. The **faddz** instruction, however, treats the interpolation values (*N*mZ*) as 32-bit fixed-point numbers; therefore, two **faddz** instructions are executed for each group of four pixels. Because of the way the **faddz** shifts

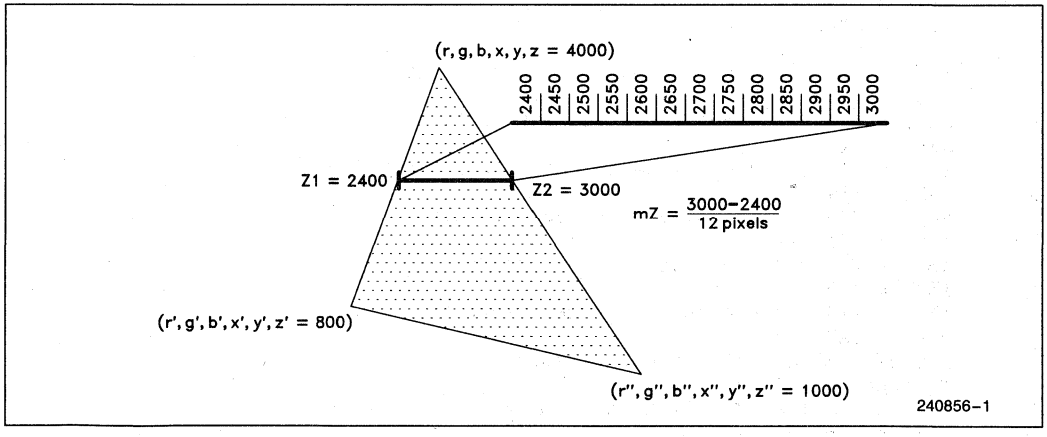


Figure 1. Z-Buffer Interpolation

the MERGE register, the first **faddz** corresponds to even-numbered pixels, while the second corresponds to odd-numbered pixels. Instead of starting with the value for the first pixel (*Z(X1)*) and adding *mZ* to each pixel to produce the value for the next pixel, the example procedure starts with the values for the first two even-numbered pixels and adds 1**mZ* to each of these values to produce the values for the adjacent odd-numbered pair. Adding 3**mZ* to each of the Z values of an odd-numbered pair produces the values for the next even-numbered pair. Figure 2 shows one way of constructing the operands before starting the distance interpolations. (The initial value given to *src1* depends on the alignment of the first pixel.) Table 1 helps to visualize the process.

After two **faddz** instructions, the MERGE register holds the Z values for four adjacent pixels (in the correct order). The **form** instruction copies MERGE into one of the 64-bit floating-point registers.

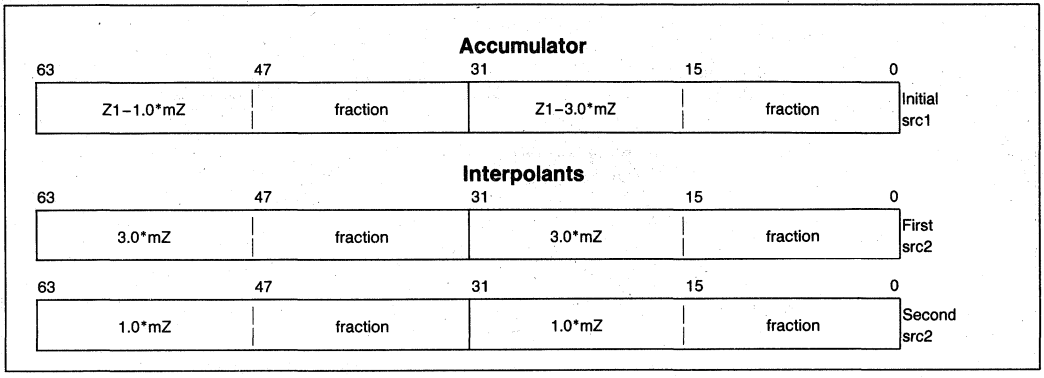


Figure 2. faddz Operands

Table 1. faddz Visualization

Operands	63-32	31-0	MERGE Register			
			63-48	47-32	31-16	15-0
src1	−1.0	−3.0				
src2	3.0	3.0				
rdest/src1	2.0	0.0	2		0	
src2	1.0	1.0				
rdest/src1	3.0	1.0	3	2	1	0
src2	3.0	3.0				
rdest/src1	6.0	4.0	6		4	
src2	1.0	1.0				
rdest/src1	7.0	5.0	7	6	5	4
src2	3.0	3.0				
rdest/src1	10.0	8.0	10		8	
src2	1.0	1.0				
rdest/src1	11.0	9.0	11	10	9	8
src2	3.0	3.0				
rdest/src1	14.0	12.0	14		12	
src2	1.0	1.0				
rdest	15.0	11.0	15	14	13	12

Because the values of *Z1* and *mZ* are constant for each loop through the rendering routine, the numbers shown here are the values of the coefficient *N*, where the actual operands have the values *Z1* + *N*mZ*. For each execution of **faddz**, *src1* is the same as *rdest* of the prior **faddz**. After every two **faddz** instructions, a **form** instruction empties the MERGE register.

```
// CONSTRUCT INTERPOLANTS iZ1 AND iZ3 GIVEN mZ
    ixfr        mZ,     iZ1         // Join each half in 64-bit register
    shl         1,      mZ,     Ra  // Ra = 2*mZ
    adds        Ra,     mZ,     Ra  // Ra = 3*mZ
    ixfr        Ra,     iZ3         // Join each half in 64-bit register
    fmov.ss     iZ1,    iZ1h        // Join each half in 64-bit register
    fmov.ss     iZ3,    iZ3h        // Join each half in 64-bit register
```

Example 3. Construction of Z Interpolants

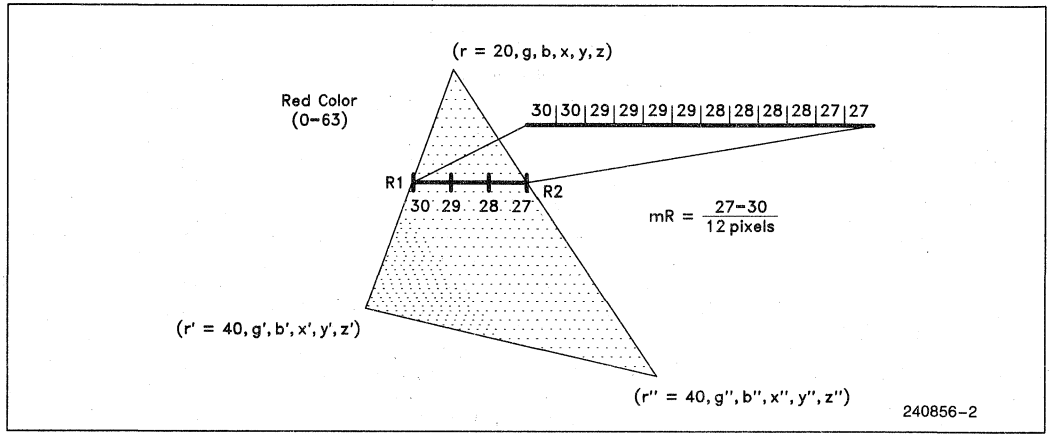


Figure 3. Pixel Interpolation for Gouraud Shading

The same register is used as both *src1* and *rdest* in all **faddz** instructions. This register serves to accumulate Z values for successive pixels; therefore, it is called an *accumulator*. The registers used as *src2* are called *interpolants*. The code in Example 3 constructs the interpolants; it needs to be executed only once for each polygon.

3.0 COLOR INTERPOLATION

To determine the RGB color intensities at each pixel, the rendering routine interpolates between the color intensities at the end points. (This rendering technique is called "Gouraud shading" after H. Gouraud, "Continuous Shading of Curved Sufaces," *IEEE Transactions on Computers*, C-20(6), June 1971, pp. 623-628.) Let the symbol C (color) represent either R (red), G (green), or B (blue). Color interpolation consists of calculating the slope of C over the given line segment, then increasing the C values of each successive pixel by that amount, starting from the values for *X1*. This must be done for C=R, C=G, and C=B. The slope of C is . . .

$$mC = (C2 - C1)*RdX$$

. . . where $RdX = 1/dX$

The value of *mC* is constant for all scan lines that intersect a given pair of polygon edges; therefore *mC* needs to be calculated only once for each such pair. Example 7 assumes that *mC* has already been calculated for all colors, and all that remains is to apply *mC* to successive pixels. Let *C(Xn)* be a C value at pixel *Xn*. Then . . .

$$C(X1) = C1$$
$$C(X1 + 1) = C1 + mC$$
$$C(X1 + 2) = C1 + 2*mC$$
$$\vdots$$
$$C(X1 + N) = C1 + N*mC$$
$$\vdots$$
$$C(X1 + dX) = C1 + dX*mC = C(X2)$$

Figure 3 illustrates Gouraud shading of a triangle.

The **faddp** instruction performs the above calculations 64 bits at a time. Because a pixel is 16 bits wide, Example 7 operates on pixels in groups of four. Instead of starting with the value for the first pixel (*C(X1)*) and adding *mC* to each pixel to produce the value for the next pixel, the example procedure starts with the values for the first four pixels and adds $4*mC$ to each group of

four to produce the values for the next four. Three **faddp** instructions are executed for each group of four pixels. The first increments the blue values; the second, green; the third, red. Figure 4 shows one way of constructing the operands for each color before starting the color interpolations. (The initial value given to *src1* depends on the alignment of the first pixel.)

Setup of the accumulator and interpolants is similar to that of the Z-buffer. The code in Example 4 constructs the interpolants; it needs to be executed only once for each pair of edges in each polygon.

4.0 BOUNDARY CONDITIONS

The i860 microprocessor operates on 64-bit quantities that are aligned on 8-byte boundaries. The code in this example takes full advantage of this design, handling four 16-bit pixels in each loop. However, if the first or last pixel of a line segment is not on an 8-byte boundary, two kinds of special considerations are required:

1. Masking of Z values near the end points.
2. Initialization of the accumulators.

4.1 Z-Buffer Masking

When either the first or last pixel of the line segment is not at an 8-byte boundary, the rendering procedure must mask the first or last set of new Z-buffer values (**newz**) so that the Z-buffer and the frame buffer are not erroneously updated. Sometimes both the first and last pixels are in the same 4-pixel set, in which case either one may not be on an 8-byte boundary. A function that looks up and calculates masks is shown in Example 5.

Because the value 0xFFFF is used for masking, the Z-buffer is initialized with 0xFFFE, so that the **fzchks** instruction always finds the mask to be greater than any Z-buffer contents.

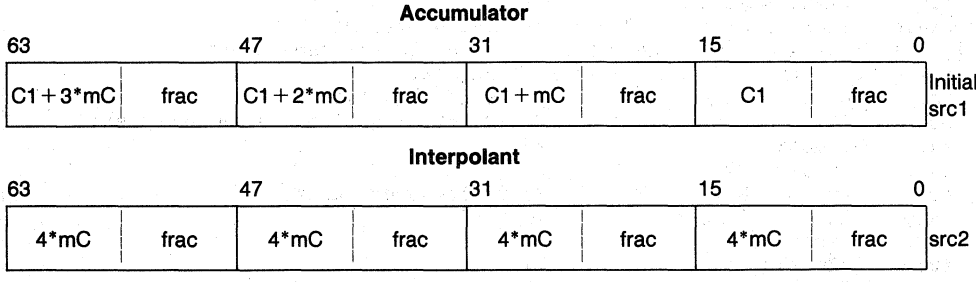


Figure 4. faddp Operands

```
// CONSTRUCT INTERPOLANTS iR, iG, iB GIVEN mR, mG, mB
    shl         18,     mR,     Ra  // Multiply each color slope by four, then
    shl         18,     mG,     Rb  //  shift by 16 to put the significant
    shl         18,     mB,     Rc  //  bits into the high-order half
    shr         16,     Ra,     mR  // Return significant 16 bits
    shr         16,     Rb,     mG  //  to low-order half. Any sign bits
    shr         16,     Rc,     mB  //  in high-order half are gone.
    or          mR,     Ra,     Ra  // Join 16-bit quarters
    or          rG,     Rb,     Rb  // in 32-bit register
    or          mB,     Rc,     Rc  //
    ixfr        Ra,     iR          // Join 32-bit halves
    ixfr        Rb,     iG          // in 64-bit register
    ixfr        Rc,     iB          //
    fmov.ss     iR,     iRh         //
    fmov.ss     iG,     iGh         //
    fmov.ss     iB,     iBh         //
```

Example 4. Construction of Color Interpolants

```
.macro zmask l_align, r_align, Rx, Ry
// l_align, r_align - left- and right-end alignment [0..3] in 2-byte units
// Rx, Ry           - scratch registers
    .data
    .align 8
left_mask:: //low        high
    .long 0x00000000, 0x00000000    // 0 mod 4
    .long 0x0000FFFF, 0x00000000    // 1 mod 4
    .long 0xFFFFFFFF, 0x00000000    // 2 mod 4
    .long 0xFFFFFFFF, 0x0000FFFF    // 3 mod 4
right_mask:://low        high
    .long 0xFFFF0000, 0xFFFFFFFF    // 0 mod 4
    .long 0x00000000, 0xFFFFFFFF    // 1 mod 4
    .long 0x00000000, 0xFFFF0000    // 2 mod 4
    .long 0x00000000, 0x00000000    // 3 mod 4

    .text
    shl     3,  l_align,    l_align // Multiply by 8
    mov     left_mask,      Rx      //
    fld.d   l_align (Rx),   lZmask  // Load 8-byte mask

    shl     3,  r_align,    r_align // Multiply by 8
    mov     right_mask,     Rx      //
    fld.d   r_align (Rx),   rZmask  // Load 8-byte mask
// If the first and last pixels are contained in the same 64-bit
// aligned set, then lZmask = lZmask OR rZmask.
    andh    0x8000, dX,     r0      // Is dX negative
    bc      L2                      // If not, right end is in other set
    fxfr    lZmask, Rx              //
    fxfr    rZmask, Ry              //
    or      Rx,     Ry,     Rx      // OR low-order half
    ixfr    Rx,     lZmask          //
    fxfr    lZmaskh,        Rx      //
    fxfr    rZmaskh,        Ry      //
    or      Rx,     Ry,     Rx      // OR high-order half
    ixfr    Rx,     lZmaskh         //
L2: nop                             //
    .endm
```

Example 5. Z Mask Procedure

Table 2. Accumulator Initial Values

Alignment	Initial Z Accumulator Values	
0	Z1 − 1*mZ	Z1 − 3*mZ
2	Z1 − 2*mZ	Z1 − 4*mZ
4	Z1 − 3*mZ	Z1 − 5*mZ
6	Z1 − 4*mZ	Z1 − 6*mZ

Alignment	Initial Color Accumulator Values C = R, G, B			
0	C1 − 1*mC	C1 − 2*mC	C1 − 3*mC	C1 − 4*mC
2	C1 − 2*mC	C1 − 3*mC	C1 − 4*mC	C1 − 5*mC
4	C1 − 3*mC	C1 − 4*mC	C1 − 5*mC	C1 − 6*mC
6	C1 − 4*mC	C1 − 5*mC	C1 − 6*mC	C1 − 7*mC

Table 3. Accumulator Initialization Table

Alignment	Table Values			
	*mZ	*mR	*mG	*mB
0	−1, −3	−1, −2, −3, −4	−1, −2, −3, −4	−1, −2, −3, −4
2	−2, −4	−2, −3, −4, −5	−2, −3, −4, −5	−2, −3, −4, −5
4	−3, −5	−3, −4, −5, −6	−3, −4, −5, −6	−3, −4, −5, −6
6	−4, −6	−4, −5, −6, −7	−4, −5, −6, −7	−4, −5, −6, −7

4.2 Accumulator Initialization

When the first pixel of the line segment is not at an 8-byte boundary, initial values placed in the accumulators (*aZ, aB, aG,* and *aR*) must be selected so that *Z1, Red1, Grn1,* and *Blu1* correspond to the correct pixel. The desired result is that shown by Table 2. However, each value is a composite of two terms: one that is constant for each edge pair (*n*mZ, n*mR, n*mG, n*mB*) and one that can vary with each scan line (*Z1, Red1, Grn1, Blu1*). The example assumes that the constant values have all been calculated and stored in a memory table of the format shown by Table 3. At the beginning of each line segment the values appropriate to the alignment of the line segment are retrieved from the table and added to the initial Z and color values, as shown in Example 6.

5.0 THE INNER LOOP

Once the proper preparations have been made, only a minimal amount of code is needed to render each scan-line segment of a polygon. The code shown in Example 7 operates on four pixels in each loop. The left and right ends of the line segment go through different logic paths so that the Z-buffer masks can be applied by the **form** instruction. All the interior points are handled by the tight inner loop.

The controlling variable **dX** is zero-relative and is expressed as a number of pixels. The value of **dX** also indicates alignment of the end-points with respect to the 4-pixel groups. Unaligned left-end pixels are subtracted from **dX** before entering the inner loop; therefore, subsequent values of **dX** indicate the alignment of the right end. A value that is 3 mod 4 indicates that the right end is aligned, which explains the test for a value of −5 near the end of the loop (−5 mod 4 = 3). The fact that the value −5 is loaded into register **Rb** on every execution of the loop does not represent a programming inefficiency, because there is nothing else for the core unit to do at that point anyway.

```
// ACCUMULATOR INITIALIZATION TABLE
    .data; .align .double
acc_init_tab::  .double [16] 0
    .dsect
aBi:    .double     // Four initial 16-bit blue values
aGi:    .double     // Four initial 16-bit green values
aRi:    .double     // Four initial 16-bit red values
aZi:    .double     // Two initial 32-bit Z values
    .end
    .text
// INITIALIZE ACCUMULATORS
.macro acc_init  Lalign, Rtab, Rx, Ry, Fx, Fxh
//  Lalign - left-end alignment (0..3) in two-byte units
//  Rtab   - register to use for addressing the table
//  Rx, Ry, Fx, Fxh - scratch registers
    mov         acc_init_tab,  Rtab    //
    shl,        5,     Lalign, Lalign  // Multiply by row width
    adds        Lalign, Rtab,  Rtab    // Index row corresponding to alignment
    fld.d       aZi(Rtab),     aZ      // Z
    ixfr        Zl,    Fx              // Z
    fld.d       aRi(Rtab),     aR      // R-Load constant values
    shl         16,    Redl,  Rx      // R-Shift starting value to hi-order
    fmov.ss     Fx,    Fxh             // Z
    shr         16,    Rx,    Ry      // R-Redl stripped of sign bits
    fiadd.dd    Fx,    aZ,    aZ      // Z
    or          Rx,    Ry,    Ry      // R-Form (Redl,Redl)
    ixfr        Ry,    Fx              // R-Put in 64-bit register
    fld.d       aGi(Rtab),     aG      // G
    shl         16,    Grnl,  Rx      // G
    fmov.ss     Fx,    Fxh             // R-Form (Redl,Redl,Redl,Redl)
    shr         16,    Rx,    Ry      // G
    fiadd.dd    Fx,    aR,    aR      // R-Add variables to constants
    or          Rx,    Ry,    Ry      // G
    ixfr        Ry,    Fx              // G
    fld.d       aBi(Rtab),     aB      // B
    shl         16,    Blul,  Rx      // B
    fmov.ss     Fx,    Fxh             // G
    shr         16,    Rx,    Ry      // B
    fiadd.dd    Fx,    aG,    aG      // G
    or          Rx,    Ry,    Ry      // B
    ixfr        Ry,    Fx              // B
    fmov.ss     Fx,    Fxh             // B
    fiadd.dd    Fx,    aB,    aB      // B
.endm
```

Example 6. Accumulator Initialization

```
//  RENDERING PROCEDURE
//      16-bit pixels, 16-bit Z-buffer
    and         3,      X1,     Ra  // Determine alignment of starting-point
    acc_init  Ra, Rb, Rc, Rd, Fa, Fah   // Initialize accumulators
    subs        4,      Ra,     Rb  // 4 - alignment
    subs        dX,     Rb,     dX  // Adjust dX by X1 alignment
    // If dX <= 0, then right end is in same set as left end
    and         3,      dX,     Rb  // Determine alignment of right end
    zmask       Ra, Rb, Rc, Rd      // Prepare both left- and right-end masks
left_end::  // Handle boundary conditions
    d.faddz     aZ,     iZ3,    aZ  // Interpolate 2 even Z values
      adds      -8,     FBP,    FBP // Anticipate autoincrement
    d.faddz     aZ,     iZ1,    aZ  // Interpolate 2 odd Z values
      adds      -8,     ZBP,    ZBP // Anticipate autoincrement
    d.form      1Zmask, newz        // Mask 4 new Z values
      fld.d     8(ZBP), oldz        // Fetch 4 old Z values
    d.faddp     aB,     iB,     aB  // Interpolate 4 blue intensities
      mov       -4,     Ra          // Loop increment: 4 pixels
    d.faddp     aG,     iG,     aG  // Interpolate 4 green intensities
      adds      -4,     dX,     dX  // Prepare dX for bla at end of loop
    d.faddp     aR,     iR,     aR  // Interpolate 4 red intensities
      bla       Ra,     dX,     L1  // Initialize LCC
    d.form      f0,     newi        // Move 4 new pixels to 64-bit reg
      adds      5,      dX,     r0  // Are there any whole sets (dX < -5)?
L1: d.fzchks    oldz,   newz,   newz// Mark closer points in PM[7..4]
      bc        short_segment       // Get out now if no whole set
    d.fnop                          //
      fld.d     16(ZBP),    oldz    // Fetch 4 old Z values
inner_loop:: // Handle all interior points
    d.faddz     aZ,     iZ3,    aZ  // Interpolate 2 even Z values
      nop                           //
    d.faddz     aZ,     iZ1,    aZ  // Interpolate 2 odd Z values
      fst.d     newz,   8(ZBP)++    // Update Z buf from prior loop
    d.form      f0,     newz        // Move 4 new Z values to 64-bit reg
      nop                           //
    d.fzchks    f0,     f0,     f0  // Shift PM[7..4] to PM[3..0]
      mov       -5,     Rb          // -5 mod 4 = 3, aligned right end
    d.faddp     aB,     iB,     aB  // Interpolate 4 blue intensities
      pst.d     newi,   8(FBP)++    // Store pixels indicated by PM[3..0]
    d.faddp     aG,     iG,     aG  // Interpolate 4 green intensities
      xor       Rb,     dX,     r0  // Are we at an aligned right end?
    d.faddp     aR,     iR,     aR  // Interpolate 4 red intensities
      bc        aligned_end         // Taken if at an aligned right end
    d.form      f0,     newi        // Move 4 new pixels to 64-bit reg
      bla       Ra, dX, inner_loop  // Loop if not at end of line segment
    d.fzchks    oldz,   newz,   newz// Mark closer points in PM[7..4]
      fld.d     16(ZBP),    oldz    // Fetch 4 old Z values for next loop
// End of inner_loop.  Right end not aligned
```

Example 7. 3-D Rendering (1 of 2)

6

```
right_end:: // Handle boundary conditions
    d.faddz     aZ,     iZ3,    aZ  // Interpolate 2 even Z values
      nop                           //
    d.faddz     aZ,     iZ1,    aZ  // Interpolate 2 odd Z values
      fst.d     newz,   8(ZBP)++    // Update Z buf from prior loop
    d.form      rZmask, newz        // Mask 4 new Z values
      nop                           //
    d.fzchks    f0,     f0,     f0  // Shift PM[7..4] to PM[3..0]
      nop                           //
    d.faddp     aB,     iB,     aB  // Interpolate 4 blue intensities
      pst.d     newi,   8(FBP)++    // Store pixels indicated by PM[3..0]
    d.faddp     aG,     iG,     aG  // Interpolate 4 green intensities
      nop                           //
    d.faddp     aR,     iR,     aR  // Interpolate 4 red intensities
      nop                           //

aligned_end::  // No special boundary conditions
    d.form      f0,     newi        // Move 4 new pixels to 64-bit reg
      br        wrap_up             //
    d.fzchks    oldz,   newz,   newz// Mark closer points in PM[7..4]
      nop                           //

short_segment::
    d.fnop                          //
      adds      8,      dX,     r0  // Is right end in same set as left?
    d.fnop                          //
      bnc.t     right_end           // Branch taken if no.
    d.fnop                          //
      fld.d     16(ZBP),    oldz    // Fetch 4 old Z values

wrap_up:: // Store the unstored and leave dual mode.
    fzchks      f0,     f0,     f0  // Shift PM[7..4] to PM[3..0]
      fst.d     newz,   8(ZBP)++    // Update Z buf from prior loop
    fnop
      pst.d     newi,   8(FBP)++    // Store pixels indicated by PM[3..0]
```

Example 7. 3-D Rendering (2 of 2)

6.0 ALTERNATIVE IMPLEMENTATIONS

Example 8 contrasts the inner loop of the 16-bit pixel rendering procedure with that of an 8-bit procedure. For 8-bit pixels, two **faddp** instructions accomplish 64-bits of pixel intensity interpolation; there is no need to maintain three separate color accumulators. Four **faddz** instructions (rather than two) are required, because eight Z values are created for the eight pixels per loop.

```
      // 8-bit Pixels, 16-Bit Zbuffer = 8 Pixels in 15 Clocks
      //    G-Unit                    |          Core Unit
inner_loop::                          ;
      d.faddz  aZ,deltaZ1,aZ          ;     fld.q  16(ZBP),oldZ_A
      d.faddz  aZ,deltaZ2,aZ          ;     nop
      d.form   f0,newZ_A              ;     nop
      d.faddz  aZ,deltaZ1,aZ          ;     andh   0x8000,dX, r0
      d.faddzz aZ,deltaZ2,aZ          ;     bnc    rightend
      d.form   f0,newZ_B              ;     nop
      d.fzchks oldZ_A,newZ_A,newZ_A   ;     nop
      d.fzchks oldZ_B,newZ_B,newZ_B   ;     nop
      d.faddp  intens,dI,intens       ;     fst.q  newZ_A ,16(ZBP)++
      d.faddp  intens,dI2,intens      ;     bte    0,dX,end
      d.form   f0,newi                ;     bla    neg8,dX,inner_loop
      d.fnop                          ;     pst.d  newi,8(FBP)++
      //-------------------------------------------------------------

      // 16-Bit Pixels, 16-Bit Zbuffer = 4 Pixels in 10 Clocks
      //    G-Unit                    |          Core Unit
inner_loop::                          ;
      d.faddz  aZ,iz3,aZ              ;     nop
      d.faddz  aZ,iz1,aZ              ;     fst.d  newz,8(ZBP)++
      d.form   f0,newz                ;     nop
      d.fzchks f0,f0,f0               ;     mov    -5,Rb
      d.faddp  aB,iB,aB               ;     pst.d  newi,8(FBP)++
      d.faddp  aG,iG,aG               ;     xor    Rb,dX,r0
      d.faddp  aR,iR,aR               ;     bc     aligned_end
      d.form   f0,newi                ;     bla    neg4,dX,inner_loop
      d.fzchks oldz,newz,newz         ;     fld.d  16(ZBP),oldz
      //-------------------------------------------------------------
```

Example 8. Inner Loop of Renderers for Two Pixel Sizes

intel®

APPLICATION NOTE

PRELIMINARY
AP-435

March 1990

FAST Fourier Transforms on the i860™ Microprocessor

MARK ATKINS
APPLICATIONS ENGINEER

Order Number: 240658-001

FAST FOURIER TRANSFORMS ON THE i860™ MICROPROCESSOR

CONTENTS

PAGE

1.0 INTRODUCTION TO FFTs 6-98

2.0 BUTTERFLY DEFINED 6-98

3.0 BIT REVERSAL 6-100

4.0 FFT IMPLEMENTATION ON THE i860™ CPU 6-100

5.0 CODE DESIGN 6-101

5.1 Cache Utilization 6-101

5.2 Pfld 6-101

5.3 Fst.q 6-101

5.4 Bit Reversal Code 6-101

CONTENTS

PAGE

6.0 PIPELINE SCHEDULING 6-101

7.0 PERFORMANCE MEASUREMENTS 6-103

7.1 Cache Fill and Writeback Time .. 6-103

8.0 CODE HIERARCHY 6-104

9.0 CONCLUSION 6-104

APPENDIX A: PROGRAM LISTINGS 6-105

ABSTRACT

The i860 Processor computes floating-point results rapidly, lending itself to DSP (digital signal processing) as well as general-purpose computing. With this high performance, DSP functions can be added to any system containing an i860 CPU. A Fast Fourier Transform (FFT) illustrates this DSP power. Complete code for the FFT is presented in this application note, as well as performance measurements. Both complex and real input data FFTs are included, as well as both Decimation in Time and Decimation in Frequency.

1.0 INTRODUCTION TO FAST FOURIER TRANSFORMS

Discrete Fourier Transforms (DFTs) change time-domain data samples into a frequency-domain profile of the sampled signal. The frequency-domain representation consists of the magnitudes of sine waves at various frequencies, which would recreate the original data if superimposed. To accomplish the transform, a DFT adds combinations of the input data samples, after multiplying some of those inputs with weighting factors. The number of samples, "N", is usually a power of two.

Each result in the frequency domain comes from a weighted sum of all data samples. The weighting ("W") factors are called "twiddles", and are complex cosine/sine values for each particular frequency.

The FFT (Fast Fourier Transform) is an efficient implementation of the DFT, defined by:

$x(n)$ = time domain samples of the signal, $n = 0, 1, \ldots N-1$

$X(k)$ = the Discrete Fourier Transform of $x(n)$, $k = 0,1, \ldots N-1$

$= \Sigma\, x(n) * W^{nk}$, $n = 0$ to N-1, and $W^{nk} = e^{-j2\pi nk/N}$, where $j = \sqrt{-1}$

$= \Sigma\, x(n) * (\cos(2\pi nk/N) - j * \sin(2\pi nk/N))$

The (N-1) complex adds and (N-1) complex multiplications required for each X(k) make the DFT an Order (N^2) computation. Fortunately, the FFT decomposes this to an Order ($N * \log_2 N$) algorithm by splitting the N-sum into units of 2-sums. These units are called "butterflies" because they produce 2 output values from 2 inputs, with the butterfly-shaped dataflow shown below. (Some FFT algorithms, called Radix-4, use 4-input, 4-output butterflies.) The butterfly calculations are executed in stages, with $\log_2 N$ stages and N/2 butterflies per stage.

The subdivision, or decimation, of the N-sum into butterflies can be done via two different methods: "Decimation in Time" (DIT) or "Decimation in Frequency" (DIF). The methods differ in the ordering of twiddles and the form of the butterfly arithmetic, but they yield the same answer. They are based on different mathematical derivations of the FFT: DIT results from recursively splitting the input time-domain samples into an even-indexed group and an odd-indexed, while DIF comes from splitting the DFT output frequency-domain points into odd/even groups.

2.0 BUTTERFLY DEFINED

Let A = the first input to the butterfly (complex number, composed of Real part AR and Imaginary part AI)

B = the second input to the butterfly (complex, BR and BI)

W = twiddle factor (also complex, WR and WI)

Anew = complex result #1, which overwrites A

Bnew = result #2, which overwrites B

For a "Decimation-in-Frequency" butterfly,

Anew = A + B

Bnew = (A − B) * W

The complex add, subtract, and multiply of a butterfly decompose into 4 real multiplies, 3 real adds, and 3 real subtracts:

AnewR = AR + BR tempR = AR-BR

AnewI = AI + BI tempI = AI-BI

BnewR = (tempR * WR) − (tempI * WI)

BnewI = (tempR * WI) + (tempI * WR)

For a "Decimation-in-Time" butterfly,

Anew = A + (B * W)

Bnew = A − (B * W)

The number of real operations remains 4 multiplies and 6 add/subtracts, but the equations differ and the multiplies must be done first:

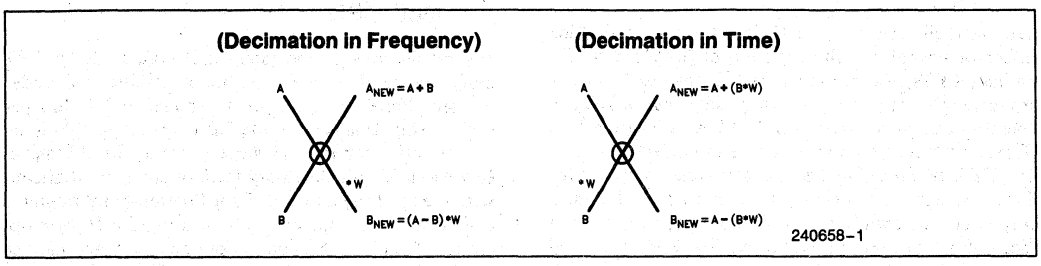
tempR = (WR * BR) − (WI * BI)

tempI = (WR * BI) + (WI * BR)

AnewR = AR + tempR BnewR = AR-tempR

AnewI = AI + tempI BnewI = AI-tempI

Butterfly Dataflow:



The stages, twiddles, and butterflies for 8-point FFTs are shown in Figures 1 and 2. For larger values of N, the dataflow patterns are very similar, with N/2 butterflies executed at each stage, and a greater number of stages. Refer to a text on Digital Signal Processing for a complete discussion of FFT design, such as chapter 6 of *Theory and Application of Digital Signal Processing* (see the Bibliography at the end of this note).

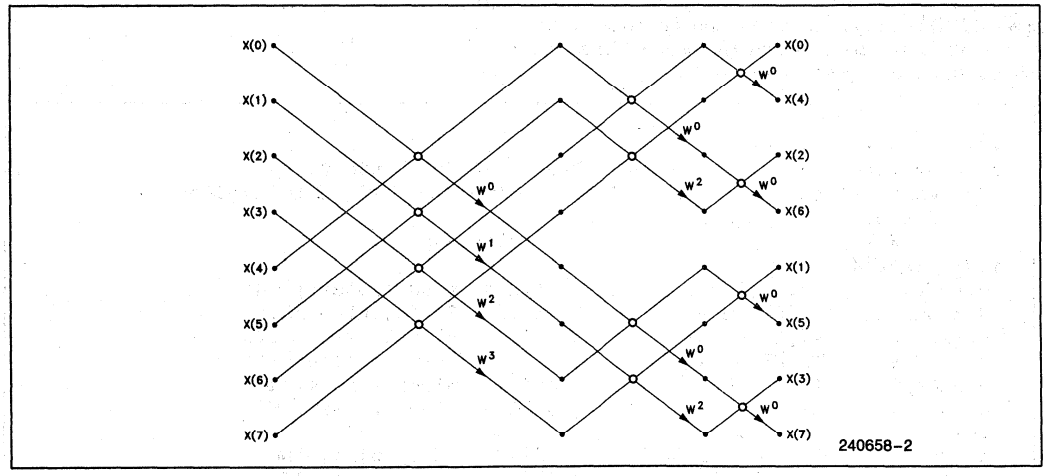


Figure 1. Decimation-In-Frequency FFT for 8 points

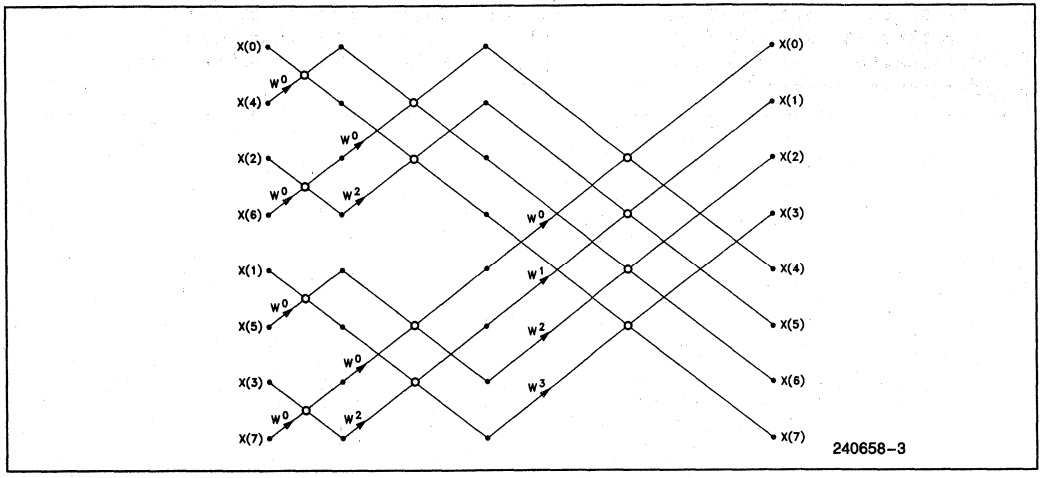


Figure 2. Decimation-In-Time FFT for 8 points

3.0 BIT REVERSAL

Due to their structure, FFT algorithms have the side-effect of scrambling the ordering of output data. For radix-2 FFTs, the output is in "bit-reversed" order—for example, the value for frequency one is NOT at location one in the output array, but at location N/2. Time to unscramble the output is often NOT included in FFT benchmarking, because scrambled output is fine for some signal-processing uses such as convolution. In any event, unscrambling consists of swapping the locations of pairs of output values. Alternatively, input values can be shuffled, as Decimation in Time usually does before the first stage (as shown in Figure 2). Otherwise, to avoid the shuffling of input in DIT, the twiddles must be accessed in bit-reversed order. As an example of bit-reversal, for 256 points the reordering involves:

SWAP X(i) and X(j), where i = 'klmnopqr'b and j = 'rqponmlk'b. The second index (j) contains the same bits as (i), but in opposite order.

4.0 FFT IMPLEMENTATION ON THE i860 CPU

Several features of the i860 CPU contribute to FFT performance. The floating-point multiplier and adder can simultaneously produce 1 product and 1 sum per cycle, using **Dual-Operation** FP instructions. To fetch the butterfly inputs and store outputs, **Dual-Instruction-Mode** allows a memory fetch or store simultaneous with the multiply and add. Four floating-point numbers can be stored by one instruction, using the 16-byte-operand "**fst.q**" instruction. Likewise, 16 bytes can be fetched from the data cache in one **fld.q** op.

The floating-point arithmetic of the i860 CPU conforms to IEEE 754 format, which some DSPs fail to do. Shown below is code for the crucial inner loop of the FFT:

```
//--------------------
//inner_loop: do 2 Decimation-In-Frequency FFT butterflies.
//  Twelve clocks for 2 butterflies - 12 FP add/sub, 8 multiplies,
//   6 8-byte loads, 4 8-byte stores.
//      FP-op                       ;     Core-op
inner_loop::
 d.r2pt.ss    WR,DI,BnewR          ; pfld.d  wind (wstart),WRo
 d.pfsub.ss   AR,BR,AnewRo         ; fld.d   8 (fetch)++,ARo
 d.ratls2.ss  AI,BI,AnewIo         ; fld.d   offset (fetch),BRo
 d.i2st.ss    WI,DR,BnewI          ; fst.q   AnewR,16(store)++
 d.ratlp2.ss  AR,BR,DR             ; adds    wincr,wind,wind
 d.ialp2.ss   AI,BI,DI             ; pfld.d  wind (wstart),WR
//--------------------
 d.r2pt.ss    WRo,DI,BnewRo        ; adds    wincr,wind,wind
 d.pfsub.ss   ARo,BRo,AnewR        ; fld.d   8 (fetch)++,AR
 d.ratls2.ss  AIo,BIo,AnewI        ; fld.d   offset (fetch),BR
 d.i2st.ss    WIo,DR,BnewIo        ; fst.q   BnewR, offset (store)
 d.ratlp2.ss  ARo,BRo,DR           ; bla     decrem,count,inner_loop
 d.ialp2.ss   AIo,BIo,DI           ; and     wlimit,wind,wind //modulo.
//--------------------
```

5.0 CODE DESIGN

Refer to the inner_loop above and code listings at the end of this application note for the discussions that follow. Refer to the *"i860™ 64-bit Microprocessor Programmer's Reference Manual"* (Intel order number 240329) for details on instructions and formats.

The programs include both assembly and Fortran components. Input data can number any power of 2 from 16 to 1024 points. The algorithms are radix-2, floating-point, in-place. Included in the listing are both Decimation-in-Time and Frequency, and both complex-input and real-input FFTs.

5.1 Cache Utilization

Because the instruction cache contains 4-Kbytes, all required code easily fits in cache. However, a 1024-point complex FFT fills the 8-Kbyte data cache with the input X() array. Thus the more rarely-used twiddle W() array is intentionally kept out of cache, as described in the "pfld" section.

A subroutine ("fetch.ss") is used to move the input data array efficiently into cache for the 1024-point FFT. "Fetch" allows all data to be brought into cache using the next-near (NENE#) accesses to DRAM. Without that routine, getting A and B from locations separated by 4 Kbytes (NOT the same DRAM page) makes fetches and writebacks from DRAM for the first stage slower, and adds 30% to overall execution time.

For larger FFTs (2048 points = 16 kB), straightforward expansion of the present algorithm would cause increased cache misses. Thus a larger FFT should be broken into multiple FFTs of 1024 points so that all 10 stages of each can achieve high cache hits. The algorithm becomes (assuming 2048 points, Decimation-In-Time):

1) Bit-reverse the entire input array
2) Do a 10-stage FFT on the second set of 1024 points. Cache hits should be high on those, since they were most recently accessed by the bit-reversal.
3) Do a 10-stage FFT on the first 1024 points. Prefetch before the first stage to ensure cache hits.
4) Combine the 2 separate 1024-point results with a final stage of butterflies, where A is offset from B by 8 Kbytes.

5.2 Pfld

Twiddle factors (W) are fetched with **pfld** (Pipelined Floating-Point Load), to avoid caching them. Only in the first stage are all the W() elements used; successive stages use fewer and fewer elements, which are separated by larger and larger strides. Thus placing W() in cache would be inefficient. The streaming of W() from main memory actually yields better performance than caching W(), for 512 and 1024 points. With the i860 CPU's 8-byte external data bus, a complex W() value can be transferred in a single bus cycle. Some FFT routines calculate W() on the fly, rather than fetching precalculated values; however, performance decreases due to the added run-time calculations.

5.3 Fst.q

Quad-word (16-byte) stores allow 4 floating-point register values to update the cache in one cycle. Likewise, **fld.q** (Quad Floating Point Load) transfers 4 values to the registers in a cycle. However, in some FFT stages, double-word fetches (**fld.d**) are used instead of **fld.q**; that allows the "background" fetch of a set of operands concurrent with arithmetic on the other set. For the same reason, the inner loop does 2 butterflies, rather than one.

5.4 Bit Reversal Code

The code for bit-reversal fetches the indices of 2 elements to be swapped from a pre-allocated array of indices, and swaps the data elements. Again, **pfld.d** keeps the indices out of cache, for the 1024 point case. That assembly version of bit-reversal is approximately 7 times faster than the standard Fortran routine. The array of indices was generated by printing out the values generated during operation of the standard Fortran version; similarly, the twiddle W() values can be pre-allocated and generated using a high-level- language program.

6.0 PIPELINE SCHEDULING

The adder pipeline is 3 stages, as is the multiplier; for the calculation of

$$BnewR = (AR - BR) * WR - (AI - BI) * WI$$

the adder result is fed back into the multiplier, and the product again feeds into the adder. The adder and multiplier pipes each advance one stage for each floating-point instruction issued.

The butterfly decomposes into 6 real add/subtracts and 4 real multiplies. Thus the best possible performance would be 6 clocks per butterfly, with the multiplies totally overlapping the adds. The overlap is accomplished with the Dual-Operation instructions:

r2pt	(KR*src2, Treg+Mout, load KR ← src1)
rat1s2	(KR*Aout, src1-src2, load T ← Mout)
i2st	(KI*src2, Treg-Mout, load KI ← src1)
rat1p2	(KR*Aout, src1+src2, load T ← Mout)
ia1p2	(KI*Aout, src1+src2, load KI ← src1)

KR, KI, and T are operand registers feeding the multiplier and adder, separate from the floating-point register file. They permit the 4 inputs for multiply and add, even thought the instruction format holds only 2 registers. "Aout" and "Mout" are adder and multiplier outputs.

The data path arrangements of some of these ops are illustrated in Figures 3 and 4. Fetching and storing of butterfly operands is overlapped with the calculations, using Dual Instruction Mode — the integer core op (such as a load or branch) and FP op are fetched simultaneously from the instruction cache and executed simultaneously.

Scheduling of instructions was done with a pipeline diagram, as illustrated in the comments of the code listing of difstep.ss in the Appendix. (The comments show the machine state after the instruction is processed.) Begin by placing the desired results in the rightmost column, then tracing progress backwards through the adder. When adder inputs are products (of the multiplier), one product is kept in the Treg for a cycle while the other propogates through the multiplier final stage. Those products can be traced back on the multiplier pipeline, to determine at what instruction the multiplier inputs must be provided.

For example, place the BnewR label in the "Write" stage of the pipe (the output of the Adder). Now

$$BnewR = WR * DR - WI * DI$$

Three instructions earlier, the adder inputs for BnewR must be fed to adder; those inputs are products, one of which comes directly from the multiplier output, and the other from the Treg. The multiplier output and Treg value must then be traced back through multiplier stages, requiring the following instructions:

i2st.ss	WIo,DR,BnewIo	as the 10th op of 12, to start (T − Mout)
rat1s2.ss	AIo,BIo,AnewI	as the 9th instruction, to update the Treg
ia1p2.ss	AI,BI,DI	as the 6th op, to multiply DI * WI
rat1p2.ss	AR,BR,DR	as the 5th op, to multiply DR * WR
rat1s2.ss	AI,BI,AnewIo	as the 3rd, to start DI into the adder
pfsub.ss	AR,BR,AnewRo	as the 2nd, to start DR into the adder

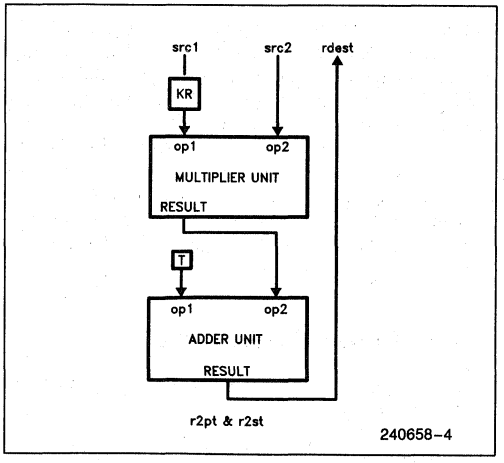


Figure 3. Datapath for r2pt op

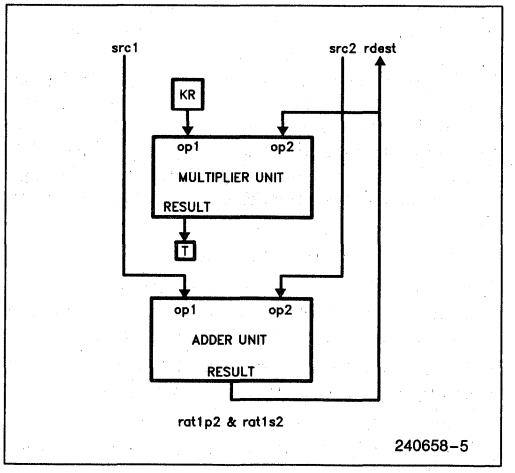


Figure 4. Datapath for rat1p2 op

Some trial-and-error ordering of the desired outputs is needed to devise a sequence which keeps the adder pipeline full. An op is chosen for each slot for its ability to load the KR or KI register, or to initiate an adder operation simultaneous with the multiplies required to calculate BnewR and BnewI.

Handy hints to assist dual-operation scheduling include:

1) *Feedback* the adder result to the multiplier, or visa versa, whenever possible. For example, the rat1p2 op feeds adder-out to multiplier. Thus both src1 and src2 fields of the instruction are available to feed the adder-in, and a simultaneous useful add and multiply are initiated.
2) *Freeze* one of the pipes, by using a pfadd or pfmul, when appropriate. In the butterfly, where 6 adds are done for every 4 multiplies, freezing of the multiplier does not degrade performance. The freeze allows multiplier results to be held until needed in the adder.
3) The *Treg* can hold a multiplier result for several cycles until needed in the adder.
4) *Unroll a* loop to do 2 iterations per loop. That provides time to fetch inputs for iteration 2 while calculating iteration 1, and store results of iteration 1 (and fetch more inputs) while calculting iteration 2.

7.0 PERFORMANCE MEASUREMENTS

The code was run on an evaluation card with DRAM memory only, no external cache, 33.33 MHz clock, and 5 wait-states or more for some accesses. Next-near accesses (address falls into the same DRAM page as the previous access) are zero wait-state, but far accesses take 5 or more wait-states. The code was run under a virtual-memory multitasking executive. Shown below are measured results:

System: 33.3 MHz 80860 with a single bank of static-column DRAM

Algorithm: Radix-2 FFT, in-place. Data is IEEE 754 single-precision floating point. Implemented in assembly-language and Fortran code.

Type of FFT	Time	Time (including bit-reversal)
1024-point-complex, DIF	1.17 ms	1.33 ms
1024-point-real		0.67 ms
512-point-complex, DIF	0.48 ms	0.56 ms
512-point-real		0.33 ms
256-point-complex, DIF	0.22 ms	0.26 ms
1024-point-complex, DIT		1.37 ms
512-point-complex, DIT		0.59 ms

7.1 Cache Fill and Writeback Time

Measured times do not include cache-fill and writeback. That is, the timings measured 200,000 executions of the FFT using the same input array. (Performance figures offered by other manufacturers for DSP chips likewise assume that the data is already in on-chip RAM. Of course, the i860 CPU will do that fetching automatically into its data cache.) The additional time for cache fill and writeback were measured as:

1024-point-complex 0.25 ms (8 Kbytes fetched, 8 Kbytes writeback)

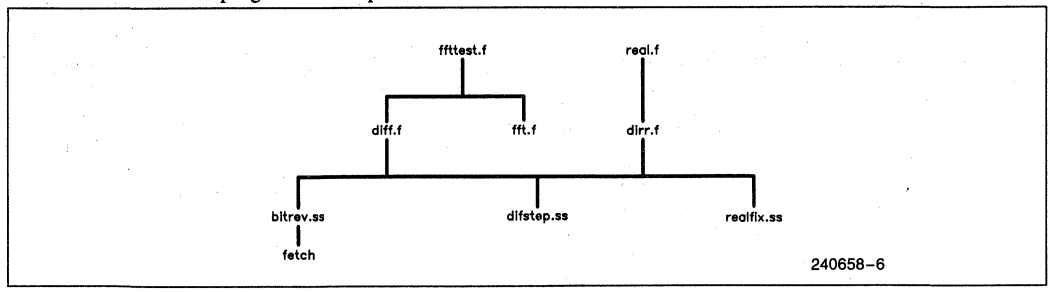
512-point-complex 0.12 ms (4 Kbytes)

To quantify the calculations in MFlops (Millions of FLoating-point OPerations per Second), consider that the 1024-point complex FFT is implemented with about 16,400 multiplies and 28,700 adds/subtracts. Thus the 1.17 ms translates to a sustained 38.5 MFlops rate. For 512 points, the required 20,000 Flops means 41.6 MFlops.

The overall FFT is about 10 times faster than the equivalent Fortran. Inner loop performance was measured at 13 cycles for the 24 instructions, which is 6.5 cycles per butterfly.

8.0 CODE HIERARCHY

Pictured below are the programs developed for the i860 CPU FFT:



The Fortran program **ffttest.f** is the highest-level program of those listed on the following pages. It calls two FFT subroutines, **diff.f** and **fft.f,** then compares their outputs. **Fft.f** is a Fortran decimation-in-time algorithm, while **diff.f** is the high-speed DIF routine. **Diff.f** is callable by C or Fortran applications. It in turn calls **difstep**, which is implemented in assembly code (**difstep.ss**). **Difstep** is called once per stage of the FFT. A Fortran version (**difstepf.f**) is shown, for comparison. Other assembly routines are the bit-reversal-data-movement (**bitrev.ss**) and prefetch ("fetch" inside **bitrev.ss**).

Difstep.ss contains approximately 225 assembly instructions, and **bitrev.ss** contains about 24. The Fortran **diff.f** compiles to about 80 instructions.

A Decimation-in-Time version of **diff.f** and **difstep.ss** can be found in **ditt.f** and **ditstep.ss**. The DIT version performs 5-10% slower than the Decimation-in-Frequency because the DIT loop takes 7 cycles per butterfly, while DIF takes 6.

A real-input algorithm is **dirr.f,** which can be called and tested using program **real.f**. **Dirr.f** calls **difstep** to do a complex DIF FFT on N real data points, but treats them as N/2 complex points. Then **realfix.ss** is called by dirr.f to fix the DIF output, compensating for the treatment of the N real points as N/2 complex. The derivation of the real-fix can be found in reference 3, *Numerical Recipes in C.*

The mixture of Fortran, C, and assembly code is accomplished by passing function inputs and outputs in registers. Only pointers and integer values were used in the above code, but floating point parameters can also be exchanged. A calling program feeds arguments to a function in r16, r17, and higher-numbered integer registers. The callee is permitted to destroy the contents of those registers, but r1:r15 must be preserved. For more details on parameter-passing conventions see the *i860 64-bit Microprocessor Programmer's Reference Manual,* Chapter 8.

9.0 CONCLUSION

The i860 CPU computes very Fast Fourier Transforms, quicker than most high-end dedicated DSP chips. Contributing to the FFT performance are the 8-kByte on-chip data cache and 4-kByte instruction cache. Also the 8-byte external data bus, pfld instruction, and 16-byte data cache width provide sufficient bandwidth to keep the arithmetic units busy. Dual-Operation instructions and Dual-Instruction-Mode allow parallel data movement and calculations. The 33.3 MHz clock rate allows both an add and a multiply every 30 ns, giving a time of 1.17 ms for a 1024-point complex FFT. A 40 MHz i860 Microprocessor will yield a time of less than 1 mSec.

ACKNOWLEDGEMENTS

The author wishes to thank Tricord Systems, Inc. for providing the key inner loop kernel design of the FFT.

BIBLIOGRAPHY

1. Gold, Bernard and Rabiner, Lawrence, *Theory and Application of Digital Signal Processing*, 1975, Prentice-Hall Inc., Englewood Cliffs, NJ. Pages 356-381,573ff

   [This text explains DFT and FFT basics well, with ample pictures]

2. Horden, Ira, "An FFT Algorithm For MCS(c)-96 Products Including Supporting Routines and Examples", Intel Application Note AP-275, order number 270189. (That Application Note can also be found in the Intel Embedded Controller Handbook, Volume II, order number 210918)

   [The note, dated 9/87, reviews FFT theory, real vs. complex, A/D issues, and waveforms]

3. Press, William, Flannery, Brian, et. al., *Numerical Recipes in C*, 1988, Cambridge University Press. Pages 398-424.

   [Numerical Recipes contains the C-code source for "realfix"]

APPENDIX A
PROGRAM LISTINGS

Pg.

A-2 1) diff.f:
Fortran module to do fast Decimation-In-Frequency (DIF) Radix-2 FFT.

A-3 2) difstep.ss:
Assembly code which does all DIF FFT butterflies; called by diff.f.

A-11 3) difstepf.f:
Fortran equivalent of difstep.ss. Included here for clarity.

A-13 4) bitrev.ss:
Assembly code to do bit-reversal.

A-17 5) ffttest.f:
Highest-level Fortran code. Tests diff.f or ditt.f.

A-21 6) ditt.f:
Fortran module to do fast Decimation-In-Time (DIT) Radix-2 FFT.

A-22 7) ditstep.ss:
Assembly code which does all DIT FFT butterflies; called by ditt.f.

A-30 8) dirr.f:
Fortran module for Real-Input Decimation-In-Frequency (DIF) Radix-2 FFT.

A-31 9) realfix.ss:
Assembly code required by dirr.f to compensate for Real-Input.

A-36 10) real.f:
Highest-level Fortran code, for Real-value input. Tests dirr.f.

A-40 11) fft.f:
Fortran FFT algorithm. Generates "correct" answers for comparison against the other code.

A-43 12) makefile:
Unix V/386 version of a makefile to maintain the FFT code, using the Unix "make" program-maintenance utility. Note that this makefile uses the Unix macro preprocessor "m4" to convert symbolic names to register numbers.

A-45 13) start.ss:
Assembly code preamble for Fortran runtime.

A-45 14) time.c:
Dummy routine, used to install breakpoints.

```
C-------------------
C File: diff.f
C  FFT - Decimation in Freq, radix-2, inplace, 1-dimen

C Intel assumes no responsibility for use or misuse of this code.

C  5/19/89: call fetch8() added for 1024-point caching.
C  6/01/89: fetch() CRUCIAL-30% performance loss if removed

C Inputs:
C   A= complex array of input, up to 1024 pts, single-prec float
C   M= log of number of pts
C    = (number of stages of FFT)
C   N = number of points. ie, N= 2**M = number of pts
C   W= complex array of twiddle factors, length N/2.
C   REV= 0 if bitreversed output ok. 1=must re-order output
C
C Outputs:
C  A= complex fft of input A
C
   subroutine diff(a,m,N,W,REV)
   integer m,N, i, j,k, REV,wlimit
   integer offset, stage, groups, wincr,powers2(0:10)
   complex a(n),w(N/2),temp

   data powers2 /1,2,4,8,16,32,64,128,256,512,1024/
C Powers2 to avoid calls to POW, DIV

C Twiddle factor array w(k) has (cos,-sin) of 2pi*k/N
CC Assume the caller provides w(k) constants ALREADY initialized
C------------
C Pre-touch data, lock into cache, for 8kByte fft:
   IF (N .gt. 513) THEN
            call fetch(a,%VAL(n))
   ENDIF
C------------
   wlimit = 8*((N/2) - 1)

C "DO 20" stage-loop
   DO 20 stage = 1,m
           groups = powers2(stage-1)
C groups=number of times the twiddle factors are used, ie, the number of
C  smaller DFTs the stage is split into.

C  offset gets N/2,N/4,N/8,N/16,...
           offset = powers2(m-stage)
           wincr = groups
           call difstep(a,w,groups,offset,wincr,wlimit)
20 CONTINUE

   IF (REV .ne. 0) THEN
cc REV .ne. 0 means must do bit-reversal reordering of output
   call bitrev(a,%VAL(M),n)
   ENDIF

   RETURN
   END
C------------
```

```
//--------------
// difstep.ss: do one stage of fft butterflies
// DIF = Decimation in Frequency, radix-2, inplace, 1-dimension
// (C) Copyright 1989 INTEL Corporation.
//   Inner loop developed with assistance from Tricord Systems, Inc.
//--------------
// 5/18/89: 1 pm - offset_2 added, as next-to-last stage was slow
// 5/19/89: 4 pm - fetch8() routine added, for cache miss avoidance.
// 5/31/89: am - use fst.q (13% perf improvement of inner_loop!)
//  last_bfly added, for performance.
// 6/02/89: am - bptr deleted. Modulo-address W (5% perf improved)
//------------
// Intel is not responsible for use nor for misuse of this program.
//------------
// Do one entire stage (n/2 butterflies). Sample invocation:
//  call difstep(a,w,groups,offset,wincr,wlimit)
//=====================================================
// Inputs:
//   A= complex array of input, single-prec float
//        (complex stored as 4byte real, 4byte imag contiguously)
//   W= pointer to array of twiddle factors. Assuming W(k) is
//      CMPLX(cos(2pi*k/N)),-sin(2pi*k/N))  for k=0 to (N/2)-1.
//   offset = distance (except for scale-by-8byte sizeof(complex)) between
//      the 2 input values for each butterfly.
//      Offset also is the number of butterflies done per "group".
//   groups = N/(2*offset). The number of sub-DFTs this stage is split into.
//   wincr = distance (except for scale-by-8byte sizeof(complex)) between
//     successive w values for successive butterflies
//   wlimit =max index, in bytes, of W table.
//
// Outputs:
//   A= complex radix-2 butterflied version of input.
//------------------
define(astart, r16)  //input data base address
define(wstart,r17) //twiddle array ptr. Because w-contents depend on N,
// we will assume the caller has initialized w() array.
define(groups,r18) //groups=number of sub-DFTs this stage is split into.
define(offset,r19)  //offset (initially elements, mult by 8 to get bytes)
// between node and its dual (the 2 numbers to butterfly, ie. A and B)
define(wincr,r20) //increment between successive W values. Remains constant
// within a given stage. For Decimation in Freq, wincr addressing is:
   //  +8 for offset=N/2 (W0,W1,W2,W3,...W(n-1))
   //  +16    offset=N/4 (W0,   W2,   W4, ...  ) etc...
define(wlimit,r21) //max index, in bytes, of W table.
define(wind,r22) //current index, in bytes, of W table.
define(offset2,r23)  //offset*2

define(decrem,r24) //bla decrement
define(somecount,r25) // bla counter

define(FEtch, r26)  //pointer to 1st component of butterfly (load)
define(STore,r27)  // " " 1st component of butterfly (store)
```

```
// f4:f7 spare
define(AR, f12)  //element A, real component
define(AI, f13) // " ", imag
define(ARo,f14) // extra A value, for prefetch (o="odd")
define(AIo,f15)
define(BR, f16) //element B, real component
define(BI, f17)
define(BRo,f18) // extra B value, for prefetch
define(BIo,f19)

define(ER, f20) //A+B, real (ER = AR + BR)
define(EI, f21) // "   imag "
define(ERo,f22) //A+B, real, previous loop's value
define(EIo,f23) // "   imag "

define(FR, f24) //W*(A-B), real
define(FI, f25) // "       imag "
define(FRo,f26)
define(FIo,f27)

define(DR, f28) //Difference of A-B, real part
define(DI, f29) // " ", imag "
define(WR, f30) //W (twiddle factor), real part
define(WI, f31) // " " , imag
define(WRo,f10) //W (twiddle factor), real part (EXTRA copy)
define(WIo,f11) // " " , imag

.text
.align .quad
_difstep_::
 ld.l      0(groups),groups  //fix Fortran call-by-ref
 ld.l      0(offset),offset  //
 shl       3,offset,offset  // change from elements to bytes
 shl       1,offset,offset2

 fst.q   f8 ,-16(sp)++ //save "local" regs
 fst.q   f12,-16(sp)++ // " "

 adds      -1,groups,groups // pre-decrement for bnc usage, or bla usage
 adds      -16,r0,decrem //bla decrement

//  We code the last 2 stages as special cases:
//--------
 xor     8,offset,r0 //offset=1, special case, no complex mult, funny addressing
 bcoffset_1// (ASSUMING offset=1 means wincr=0, and no twiddle used)
 xor     16,offset,r0 //offset=2, special case, no complex mult, funny addressing
 bcoffset_2//(ASSUMING offset=2 means wincr=N/4)
//--------
 ld.l      0(wincr),wincr
 ld.l      0(wlimit),wlimit
```

```
 pfadd.ss f0,f0,f0
 pfadd.ss f0,f0,f0
 pfadd.ss f0,f0,f0  // init A1,A2,A3=0
 pfmul.ss f0,f0,f0
 pfmul.ss f0,f0,f0
 pfmul.ss f0,f0,f0
//--------
 // init pointers:
 shl       3,wincr,wincr     //scale for bytes.
 shl       1,wincr,wind      //init wind =2*wincr

 pfld.d 0 ( wstart),f0
 pfld.d wincr ( wstart),f0
 adds      -8,astart,FEtch
 pfld.d wind (wstart),f0
 adds      wincr,wind,wind //wind now 3*wincr
// here fetch first set of A,B,W before bla-loop
 pfld.d wind (wstart),WR
 adds      wincr,wind,wind
 and       wlimit,wind,wind //modulo-wlimit the w index
// We do modulo-addressing on W(), to keep the pfld pipeline full. We
//  never do a W-fetch beyond the end of the table.
// And the modulo-check needs to be done only every 4th pfld, as always
//  we use a multiple of 4 W() factors.


 fld.d 8 (FEtch)++,AR
 fld.d offset (FEtch),BR
d.r2apl.ss f0,f0,f0 //clear Treg.
 adds -32,offset,somecount // bla counter (predecrement by 4 elements)
// -----------
// Definitions for pipe diagram:
//  (the complex multiply product, F, broken into 4 real mult and 2 adds):
//   WR = cos(), WI=-sin().
//   DR = AR - BR; (diffence of Real components of A,B)
//   DI = AI - BI; (diffence of Imag components)
//   ER = AR + BR; EI = AI + BI;
//   FR = K - L; where K= WR*DR, L=WI*DI
//   FI = N + M; where M= WI*DR, N=WR*DI

// For 1st time thru inner_loop, don't have correct values to store.
// Must do 1 loop before the loop, sans the stores.

first_bfly:: //fill pipe
                    // KR...KI...M1....M2....M3   T    A1....A2....A3....Write
d.r2pt.ss WR,f0,f0   // WR0  -
 pfld.d wind (wstart),WRo
d.pfsub.ss AR,BR,f0  //           -     -     -     -   DR0   -     -
 fld.d 8 (FEtch)++,ARo
d.ratls2.ss AI,BI,f0 //           -     -     -     -   DI0   DR0   -     -
 fld.d offset (FEtch),BRo
d.i2st.ss  WI,f0,f0  //      WI0  -     -     -     -   -     DI0   DR0   -
 adds       wincr,wind,wind
```

6

```
d.ratlp2.ss AR,BR,DR   //          K0   -    -    -    ER0  -    DI0   DR0
 nop
d.ialp2.ss AI,BI,DI    //          L0   K0   -         EI0  ER0  -     DI0
 pfld.d  wind (wstart),WR
d.r2pt.ss WRo,DI,f0    // WR1  -    N0   L0   K0   -    -    EI0  ER0   -
 fld.d  8 (FEtch)++,AR
d.pfsub.ss ARo,BRo,ER  //          N0   L0   K0   -    DR1  -    EI0   ER0
 fld.d  offset (FEtch),BR
d.ratls2.ss AIo,BIo,EI //          -    N0   L0   K0   DI1  DR1  -     EI0
 adds       wincr,wind,wind
d.i2st.ss  WIo,DR,f0   //      WI1  M0   -    N0   K0   K-L  DI1  DR1   -
 and        wlimit ,wind,wind

quickstart::
d.ratlp2.ss ARo,BRo,DR //          K1   M0   -    N0   ER1  FR0  DI1   DR1
 bla        decrem,somecount,inner_loop //init LCC
d.ialp2.ss AIo,BIo,DI  //          L1   K1   M0   N0   EI1  ER1  FR0   DI1
 adds       -16,astart,STore  // ptrs init 16 low, for fst.q instructions
//-------------------
// Each butterfly = 1 complx multiply, 1 complx add, 1 complx subtract
// =        4 multiply,
//          3 add
//          3 subtract
//          3 8-byte fetches (A, B, W)
//          2 8-byte stores (A, B)
//
// 6 cycles per butterfly
//
// inner_loop: iterates "offset/2" times (eg, N/4 for stage 1, N/8 for stage2),
//   for each group. It does 2 butterflies per iteration

inner_loop::
                       // KR...KI...M1...M2..M3    T    A1..A2...A3..Write
                       // |    |    |    |    |    |    |    |    |    |
d.r2pt.ss WR,DI,FR     // WR2  -    N1   L1   K1   N0   N+M  EI1  ER1   FR0
 pfld.d  wind (wstart),WRo
d.pfsub.ss AR,BR,ERo   //          N1   L1   K1   N0   DR2  FI0  EI1   ER1
 fld.d  8 (FEtch)++,ARo
d.ratls2.ss AI,BI,EIo  //          -    N1   L1   K1   DI2  DR2  FI0   EI1
 fld.d offset (FEtch),BRo
d.i2st.ss WI,DR,FI     //      WI2  M1   -    N1   K1   K-L  DI2  DR2   FI0
 fst.q ER,16(STore)++ //update ER/EI/ERo/EIo
d.ratlp2.ss AR,BR,DR   //          K2   M1   -    N1   ER2  FR1  DI2   DR2
 adds       wincr,wind,wind
d.ialp2.ss AI,BI,DI    //          L2   K2   M1   N1   EI2  ER2  FR1   DI2
//no need for modulo-check ("and") here, as odd num of W's have been fetched.
 pfld.d wind (wstart),WR
//.............................................................
```

```
                    // KR...KI...M1....M2....M3   T    A1....A2....A3....Write
d.r2pt.ss WRo,DI,FRo // WR3  -    N2    L2    K2   N1   N+M   EI2   ER2   FR1
 adds       wincr,wind,wind
d.pfsub.ss ARo,BRo,ER//          N2    L2    K2   N1   DR3   FI1   EI2   ER2
 fld.d 8 (FEtch)++,AR
d.ratls2.ss AIo,BIo,EI//         -     N2    L2   K2   DI3   DR3   FI1   EI2
 fld.d  offset (FEtch),BR
d.i2st.ss  WIo,DR,FIo//      WI3 M2    -     N2   K2   K-L   DI3   DR3   FI1
 fst.q  FR, offset (STore)
 //update FR/FI/FRo/FIo
d.ratlp2.ss ARo,BRo,DR//         K3    M2    -    N2   ER3   FR2   DI3   DR3
 bla decrem,somecount, inner_loop
d.ialp2.ss AIo,BIo,DI//          L3    K3    M2   N2   EI3   ER3   FR2   DI3
 and        wlimit,wind,wind //modulo.

end_inner_loop:: //KEEP Pipelines full
// RE-init pointers for fetches
d.fiadd.ss f0,f0,f0
 adds       offset2,astart,astart //bump to next group
      //redo A,B fetches, with proper ptr.
d.fiadd.ss f0,f0,f0
 fld.d 0(astart)  ,AR //get first AR/AI in next group
d.fiadd.ss f0,f0,f0
 fld.d  offset (astart),BR
d.fiadd.ss f0,f0,f0
 adds       0,astart,FEtch

last_bfly:: //do final 2 butterflies, start next group
                    // KR...KI...M1....M2....M3   T    A1....A2....A3....Write
d.r2pt.ss WR,DI,FR   // WR4  -    N3    L3    K3   N2   N+M   EI3   ER3   FR2
 pfld.d  wind (wstart),WRo
d.pfsub.ss AR,BR,ERo //          N3    L3    K3   N2   DR4   FI2   EI3   ER3
 fld.d  8 (FEtch)++,ARo
d.ratls2.ss AI,BI,EIo//          -     N3    L3   K3   DI4   DR4   FI2   EI3
 fld.d  offset (FEtch),BRo
d.i2st.ss  WI,DR,FI  //      WI4 M3    -     N3   K3   K-L   DI4   DR4   FI2
 fst.q ER,16(STore)++
d.ratlp2.ss AR,BR,DR //          K4    M3    -    N3   ER4   FR3   DI4   DR4
 adds       wincr,wind,wind
d.ialp2.ss AI,BI,DI  //          L4    K4    M3   N3   EI4   ER4   FR3   DI4
 pfld.d  wind (wstart),WR
//.....................................................................
                    // KR...KI...M1....M2....M3   T    A1....A2....A3....Write
d.r2pt.ss WRo,DI,FRo // WR5  -    N4    L4    K4   N3   N+M   EI4   ER4   FR3
 fld.d  8 (FEtch)++,AR
d.pfsub.ss ARo,BRo,ER//          N4    L4    K4   N3   DR5   FI3   EI4   ER4
 adds -32,offset,somecount // reset bla counter
d.ratls2.ss AIo,BIo,EI//         -     N4    L4   K4   DI5   DR5   FI3   EI4
 adds       wincr,wind,wind
d.i2st.ss  WIo,DR,FIo//      WI5 M4    -     N4   K4   K-L   DI5   DR5   FI3
 adds -1,groups,groups
d.fnop
 fld.d  offset (FEtch),BR
d.fnop
 bnc.t quickstart //branch on value of groups
d.fnop
 fst.q  FR, offset (STore)
```

```
end_last_bfly::
d.fnop
 br endit
fiadd.ss f0,f0,f0
 fst.q  FR, offset (STore) //repeated for bnc.t untaken case
.align      .quad
//=============================================================
offset_1::
// want FEtch=0,2,4,6,8,... elements.  ASSUMING wincr=0,
//  and that w=(1,0), so that no complex mult needed, and NO W will be fetched.
//  E=A+B, F=A-B.  (Per double-butterfly loop:  8 pfadd,4 dword fld, 4 fst,
//  1 bla)(fld.q required, to reduce # flds to avoid pipe stalls)
//  Performance = 4 cyc/bfly best case.

//Redefine regs for fld.q,fst.q usage, when A and B adjacent:
define(AR3,f12)  //element A, real component
define(AI3,f13) // " ", imag
define(BR3,f14)  //element B, real component
define(BI3,f15)
define(AR4,f16) // extra A value, for prefetch
define(AI4,f17)
define(BR4,f18) // extra A value, for prefetch
define(BI4,f19)

define(ER3, f20) //A+B, real (ER = AR + BR)
define(EI3, f21) // "   imag "
define(FR3, f22) //(A-B), real
define(FI3, f23) // "    imag "

define(ER4,f24) //A+B, real, extra copy
define(EI4,f25) // "   imag

define(FR4,f26)
define(FI4,f27)
//=========================================
 adds       -16,astart,FEtch
 fld.q  16 (FEtch)++,AR4
 adds       -1,groups,somecount // bla counter (predecremented already by 1)
   //using groups=blacount on the offset_1 loop, intentionally.
 adds       -16,FEtch,STore
//startup the loop:
// --------------------// A1......A2......A3......Write:
d.pfadd.ss AR4,BR4,f0   // ARn+BRn -       -       -
 fld.q  16 (FEtch)++,AR3
d.pfadd.ss AI4,BI4,f0   // AIn+BIn ERn     -       -
 adds       -2,r0,decrem //2 bflies per loop
d.pfsub.ss AR4,BR4,f0   // ARn-BRn EIn     ERn     -
 bla decrem,somecount, offset1_loop  //init LCC
d.pfsub.ss AI4,BI4,ER4  // AIn-BIn FRn     EIn     ERnext
 nop
// --------------------// A1......A2......A3......Write:
offset1_loop::
```

```
d.pfadd.ss AR3,BR3,EI4  // AR+BR   FI-     FR-     EI-
 nop
d.pfadd.ss AI3,BI3,FR4  // AI+BI   ER      FI-     FR-
 fld.q 16 (FEtch)++,AR4
d.pfsub.ss AR3,BR3,FI4  // AR-BR   EI      ER      FI-
 fst.q      ER4,16(STore)++
d.pfsub.ss AI3,BI3,ER3  // AI-BI   FR      EI      ER
 nop
d.pfadd.ss AR4,BR4,EI3  // AR2+BR2 FI      FR      EI
 fld.q  16 (FEtch)++,AR3
d.pfadd.ss AI4,BI4,FR3  // AI2+BI2 ER2     FI      FR
 nop
d.pfsub.ss AR4,BR4,FI3  // AR2-BR2 EI2     ER2     FI
 bla decrem,somecount, offset1_loop
d.pfsub.ss AI4,BI4,ER4  // AI2-BI2 FR2     EI2     ERnext
 fst.q  ER3,16(STore)++
//-------------------------
end_offset1_loop::
d.fiadd.ss f0,f0,f0
 br endit
fiadd.ss f0,f0,f0
 nop
//-------------------------
.align .quad
offset_2::
// want FEtch=0,1;4,5;8,9;12,13;... elements.
// ASSUMING wincr=N/4  (W_addr=0,N/4,0,N/4,0,...). Trivial W() factors.
// USE bla loop, incrementing FEtch by 16 (2*offset).
//  Even-indexed elements identical to offset_1,W=W0, no complex mult.
//    So FReven=(AR-BR), FIeven=(AI-BI).
//  Odd components have W=(0,-1).  So FRodd=(AI-BI), FIodd=(BR-AR).
//  Each fld.q fetches AReven,AIeven,ARodd,AIodd.

//Assume ER,EI,ERo,EIo are 4 contiguous regs.
//Assume FR,FI,FRo,FIo are 4 contiguous regs.

 adds       -16,astart,FEtch
 fld.q  16 (FEtch)++,AR
 fld.q  16 (FEtch)++,BR
 adds       0,groups,somecount //bla counter
//startup the loop:
// ---------------------// A1......A2......A3......Write:
  pfadd.ss AR ,BR ,f0   // AR+BRe                  -
  pfadd.ss AI ,BI ,f0   // AI+BIe  ER              -
d.pfadd.ss ARo,BRo,f0   // ARo+BRo EI      ER      -
 nop
d.pfadd.ss AIo,BIo,ER   // AIo+BIo ERo     EI      ER
 nop
d.pfsub.ss AR ,BR ,EI   // AR-BRe  EIo     ERo     EI
 adds       -1,r0,decrem //2 bflies per loop,but groups is half desired value.
d.pfsub.ss AI ,BI ,ERo  // AI-BIe  FR      EIo     ERo
 adds       -16,astart,STore
d.pfsub.ss AIo,BIo,EIo  // AIo-BIo FI      FR      EIo
 bla decrem,somecount, offset2_loop //init LCC
d.pfsub.ss BRo,ARo,FR   // BRo-ARo FRo     FI      FR
 nop
```

```
offset2_loop::
d.fnop
 fld.q  16 (FEtch)++,AR //fetch AR,AI,ARo,AIo
d.fnop
 fld.q  16 (FEtch)++,BR //fetch BR,BI,BRo,BIo
// ---------------------// A1......A2......A3......Write:
d.pfadd.ss AR ,BR ,FI   // AR+BRe  FIo     FRo     FI
 nop
d.pfadd.ss AI ,BI ,FRo  // AI+BIe  ER      FIo     FRo
 nop
d.pfadd.ss ARo,BRo,FIo  // ARo+BRo EI      ER      FIo
 fst.q      ER ,16(STore)++
 //update ER ,EI ,ERo,EIo
d.pfadd.ss AIo,BIo,ER   // AIo+BIo ERo     EI      ER
 nop
d.pfsub.ss AR ,BR ,EI   // AR-BRe  EIo     ERo     EI
 nop
d.pfsub.ss AI ,BI ,ERo  // AI-BIe  FR      EIo     ERo
 fst.q  FR ,16(STore)++
d.pfsub.ss AIo,BIo,EIo  // AIo-BIo FI      FR      EIo
 bla decrem,somecount,offset2_loop
d.pfsub.ss BRo,ARo,FR   // BRo-ARo FRo     FI      FR
 nop

endit::
// restore regs
fiadd.ss f0,f0,f0 //exit DIM
 fld.q   0(sp),f12
fiadd.ss f0,f0,f0 //last DIM pair
 fld.q   16(sp),f8
adds    32,sp,sp
 bri r1
  nop
//---------------------------------
```

```
c-------------------------------------------------------------
c difstepf.f: do one stage of fft (DIF) butterflies
c (C) Copyright 1989 INTEL Corporation. ALL RIGHTS RESERVED.
c-------------------------------------------------------------
c Decimation in Freq, radix-2, inplace, 1-dimen
c 6/20/89

c Do one entire stage (n/2 butterflies). Sample invocation:
c  call difstep(a,w,groups,offset,wincr)

c Inputs:
c   A= complex array of input, single-prec float
c        (complex stored as 4byte real, 4byte imag contiguously)
c   W= pointer to array of twiddle factors. Assuming W(k) is
c        CMPLX(cos(2pi*k/N)),-sin(2pi*k/N)) for k=0 to (N/2)-1.
c   offset = distance (in "elements") between
c        the 2 input values for each butterfly
c   groups = number of sub-DFTs this stage is split into.
c        (groups*offset*2 = N)
c   wincr = distance between successive w values for successive butterflies
c
c Outputs:
c A= complex butterflied version of input.

   SUBROUTINE difstep(a,w,groups,offset,wincr)
   integer groups,offset,wincr
   integer i,j,indexl,iplus
   complex a(groups*offset*2),w(groups*offset),wtemp,temp
c-------------------------------------------------------------
c We implement a...
c Special case for offset=1(last stage): no complex multiplies, simple add
c  (Performance enhancement)
   IF (offset .eq. 1) THEN
CVD$ NODEPCHK
            DO 8 i = 1,(2*groups),2
              iplus = i + 1
              temp = a(iplus)
              a(iplus) = a(i) - temp
8             a(i) = a(i) + temp
   ELSE
C------------
C Special case for offset=2 (next-to-last stage): no complex multiplies,
cc    simple add.  (Performance enhancement)
cc For half the butterflies, W=(1,0). For the other half, W=(0,-1)
   IF (offset .eq. 2) THEN
CVD$       NODEPCHK
           DO 90 i = 1,(4*groups),4
             iplus = i + 2
             temp = a(iplus)
             a(iplus) = a(i) - temp
90           a(i) = a(i) + temp
C  2nd call to i-loop: w=cmplx(0,-1.)
CVD$       NODEPCHK
CVD$       NOVECTOR
   DO 92 i = 2,(4*groups),4
     iplus = i + 2
     temp = a(i) - a(iplus)
     a(i) = a(i) + a(iplus)
92   a(iplus) = CMPLX(AIMAG(temp),-REAL(temp))
```

```
   ELSE
C------------
c "DO 20" indexl-loop is "outer loop"
CVD$        VECTOR
CVD$        NODEPCHK
   DO 20 indexl = 1,(2*offset*groups),(2*offset)
         j = 1
CVD$       NODEPCHK
CVD$       ALTCODE
           DO 10 i = indexl,(indexl+offset-1)
             iplus = i + offset
             temp = a(i) - a(iplus)
             a(i) = a(i) + a(iplus)
             a(iplus) = w(j) * temp
10           j = j + wincr
20  CONTINUE
   ENDIF
   ENDIF
   RETURN
   END
ccccccccccccccccccccccccccccccc
   subroutine fetch(a,n)
   integer n
   complex a(n),temp
cc Kludge do-nothing prefetch.
   temp = a(1)
   RETURN
   END
ccccccccccccccccccccccccccccccc
   subroutine bitrev(a,dummy,n)
C Bit-Reverse
C Inputs:
C   A= complex array of input, single-prec float
C   dummy = %val(m).  Probably unusable from Fortran.
C   N = number of input points (and output points)

C Ouput:
C   A = original A data, but in bit-reversed order from A

   integer n,i,j,k,ndiv2
   complex a(n),temp
C------------
C "DO 7" loop to in-place-bit-reverse-shuffle output
   j=1
   ndiv2 = n / 2
   DO 7  i= 1, n-1
     IF (i .lt. j) THEN
           temp = a(j)
           a(j) = a(i)
           a(i) = temp
     ENDIF
     k = ndiv2
C  "While (j .gt. k)" /*decrease j by 2**something */
6    IF (j .gt. k) THEN
           j = j-k
           k = k / 2
           GOTO 6
     ENDIF
C  Add next lower power of 2 to j
7  j = j+k
   RETURN
   END
C------------
```

```
//--------------
// bitrev.ss
// (C) Copyright 1989 INTEL Corporation. ALL RIGHTS RESERVED.
//
//   BIT-reversal of 8byte array elements.
//   IN PLACE.
// (Allows arrays of 8,16,32,64,128,256,512, or 1024 elements)
//--------------
//  INTEL is not responsible for use nor misuse of this code.
//--------------
// 8/13/89
//=================================================
//  Invocation: (from Fortran)
// call bitrev(a,%VAL(m))

// Inputs:
//   a = r16 = pointer to array of 8byte elements
//   m = r17 (call by value)= base-2 log of total number of elements
//         (2**m = N)
// Outputs:
//         a= Bit-reversed ordered version of A
//
// Expected best-can-do performance, and measured performance=
//  approx 4*N clocks (0.06 mSec for 512 points)
//------------------
define(astart, r16)  //initial input data base address
define(m, r17)
define(logN,r17)
define(dest1,r19)
define(dest2,r20)
define(dest3,r21)
define(dest4,r22)
define(iptr, r23)  //index-array pointer

define(decrem,r24) //bla decrement
define(count,r25) // bla counter

.text
.align .quad
//=======================================
_bitrev_::
_bitr_::
//fetch base address for index table (rbasetab)
// base-addr-table elements = (baseaddr, number_of_swaps-2)
// base-addr-table indexed by logN.
 shl       3,logN,r30  //scale to 8-byte-entry length
 mov       rbasetab,r29
 ld.l      r29(r30), iptr
 addu      4,r29,r29
 ld.l      r29(r30), count //number of swaps required for this value N

 pfld.d 0(iptr),f0 //initiate fetch of first 2 bit-rev indices
 pfld.d 8(iptr)++,f0
 adds      -2,r0,decrem//2 swaps per loop
 pfld.d 8(iptr)++,f0

 bla       decrem,count, revloop  //init LCC
  pfld.d 8(iptr)++,f16       //get 2 indices, but don't cache the indices
```

```
revloop:: //2 swaps per loop
 //7.5 cycles consumed for each swap, best case.
 pfld.d 8(iptr)++,f18        //2 more indices
 fxfr f16,dest1  //transfer to integer index regs
 fxfr f17,dest2
 fld.d  dest1 (astart),f24 //fetch 2 elements to swap
 fld.d  dest2 (astart),f26
 fxfr f18,dest3
 fst.d     f24, dest2 (astart)
 fst.d     f26, dest1 (astart)
 fxfr f19,dest4
 fld.d  dest3 (astart),f28
 fld.d  dest4 (astart),f30
 pfld.d 8(iptr)++,f16        //2 more indices
 fst.d     f28, dest4 (astart)
 bla decrem,count, revloop  //
  fst.d    f30, dest3 (astart)

 bri       r1
  nop
//---------------
// _fetch8_: Touch all 32-byte lines in the 8k data bytes, to get them
//   into dcache. (ASSUMING .lte. 8Kbytes and .gte. 4Kbytes)
//
//  Invocation= fetch(astart,num8)
//  Inputs=
//    astart=r16=pointer to data which is to be touched.
//    num8=r17 (passed by VALUE, %VAL(), not by reference)
//------------
// Using RC and RB to improve dcache hit rates, for FFTs bigger than
//  1024 complex (8kB).
//  RC=10 causes replacement only of block denoted by RB lsbit. RC=11 disables
//  replacement.
//--------
define(num8,r17)
define(FEtch, r26)

_fetch8_::
_fetch_::
 ld.c dirbase,r30
 or 0x800,r30,r30 // Replace Dcache slot 0 only (RC=10,RB=00)
 st.c r30,dirbase
// Put 4Kbytes into Dcache slot 0. (The rest after 4kB goes to slot1).
 adds      -4,r0,decrem  //4 8-byte-groups per cache line
 adds      508,r0,count  //512, but pre-decremented for bla usage
 bla       decrem,count,floop
  adds     -32,astart,FEtch
floop::
 bla       decrem,count,floop
  fld.d    32(FEtch)++,f30 //dummy load.

 adds      -512,num8,count
 bc        fdone  //if data exhausted, quit
// ld.c dirbase,r30
 or 0x900,r30,r30 // Replace Dcache slot 1 only (RC=10,RB=01)
 st.c r30,dirbase
```

```
 adds       -8,count,count //predecr for bla
 bla        decrem,count,floop2 //set LCC
  fld.d     32(FEtch)++,f30
floop2::
 bla        decrem,count,floop2
  fld.d     32(FEtch)++,f30 //dummy load.
fdone::
// unlock dcache
 andnot 0xF00,r30,r30  //clear RC,RB (dirbase(11:8))
 st.c r30,dirbase
 bri        r1
  nop

.data
//---------------
// rbasetab:: (Table of bit-reversed indices for bitrev subroutine)
//  base-addr-table elements = (baseaddr, number_of_swaps-2)
//  base-addr-table indexed by logN.
.align .quad
rbasetab::
.long [6]0 //don't bother with log(n)=0,1,2
.long rev8,   0
.long rev16,   4
.long rev32,   10
.long rev64,   26
.long rev128,  54
.long rev256,  118
.long rev512,  238
.long rev1024, 494
//=====================

//number of swaps=240 for N=512 (ie, 32 symmetrical patterns
//  exist between 0 and 511.)
// rev512: array of bit-reversed indices, for N=512.
//  Each entry is ("i", and "bit-reversed-i"), shifted left by 3
//    to account for 8-byte-elements.
// NOTE: This listing DOES NOT SHOW all the table elements, to save paper.

.align .quad
rev512::
.long     8, 2048,   16, 1024
.long    24, 3072,   32,  512
.long    40, 2560,   48, 1536
// ETC..., ETC...., ETC...
//===============
 .align .quad
rev1024::
.long     8, 4096,   16, 2048
.long    24, 6144,   32, 1024
.long    40, 5120,   48, 3072
.long    56, 7168,   64,  512
// ETC..., ETC...., ETC...
```

6

```
//Number of swaps = 496
//N (Number of elements) = 1024
//=================
.align .quad
revl6::
   .long 1*8,8*8,2*8,4*8
   .long 3*8,12*8,5*8,10*8
   .long 7*8,14*8,11*8,13*8
rev8::
   .long 1*8,4*8,3*8,6*8
//=================
.align .quad
rev32::
.long     8, 128,16, 64, 24, 192, 40, 160, 48, 96, 56, 224
.long    72, 144, 88, 208, 104, 176, 120, 240, 152, 200, 184, 232
//=================
.align .quad
rev64::
.long     8,  256,   16,  128
.long    24,  384,   32,   64
.long    40,  320,   48,  192
.long    56,  448,   72,  288
// ETC..., ETC...., ETC...
//=================
.align .quad
revl28::
.long     8,  512,   16,  256
.long    24,  768,   32,  128
.long    40,  640,   48,  384
.long    56,  896,   72,  576
// ETC..., ETC...., ETC...
//Number of swaps = 56 (Number of elements) =128
//=================
.align .quad
rev256::
.long     8, 1024,   16,  512
.long    24, 1536,   32,  256
.long    40, 1280,   48,  768
.long    56, 1792,   64,  128
// ETC..., ETC...., ETC...
//Number of swaps = 120, N (Number of elements) = 256
```

```
      PROGRAM FFTTEST
C
C   1-D FFT TEST PROGRAM
C
C Intel assumes no responsibility for use or misuse of this code.
C
C 7/20/89
C------------------
C
      character*8 REALLY
      PARAMETER (IREV=0)
      PARAMETER (REALLY='complex')
      PARAMETER (TIMEIT=1, CACHETIME=0)
      DATA IT/200000/
c      PARAMETER (N=1024,M=10)
       PARAMETER (N=512,M= 9)
c      PARAMETER (N=256,M= 8)
c      PARAMETER (N=128,M= 7)
c      PARAMETER (N=64,M= 6)
c      PARAMETER (N=32,M= 5)
c     PARAMETER (N=16, M=4)
      PARAMETER (PI=3.1415926536)
      COMPLEX X(N),X1(N),X2(N),X3(N), W(N/2)
c Fortran complex values stored R,I, R,I for arrays.
    Real  ASQR(N),ASQR2(N),XR(N)
    complex wtemp
    real rtemp
C

    PRINT *,' FFT test program (ffttest.f) ....'
    print *, '================================='
    IF (IREV .eq. 0) THEN
     print *,'NOT counting time for bit-reversal.'
     print *,'DO NOT expect matching answers,without bit-rev'
    ELSE
     print *, 'Time for bit-reversal included.'
    ENDIF

     print *, 'Time for cache writeback and fills...'
    IF (CACHETIME .eq. 0) THEN
     print *,' NOT included, if iterating.'
    ELSE
     print *,' ... included.'
    ENDIF

     print *, '================================='
     print *, 'If iterating... Number of Iterations =',IT
     print *, '================================='
     print *, 'Number of Points     = ', N
     print *, '(',REALLY,' data)'
     print *, '================================='
```

```
C------------------
C Init twiddle factor array w(k) with (cos,-sin) of 2pi*k/N
C (Should just declare this as constant, if N is non-variable)
C (OR could have one constant 512-entry W (for N=1024), adjust wincr accordingly
C   in diff.f for smaller N)
   rtemp = 2.0*pi/N
   wtemp= CMPLX(cos(rtemp), -sin(rtemp))
   w(1) = (1.0, 0.0)
   DO  200  k  =  2,N/2
200        w(k) = wtemp * w(k-1)
cc print *,' W (twiddle) initialization completed......'
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   INITIALIZE input data
C
      PIN = (4*PI)/ N
      DO 100 I = 1, N
c  For testing with sinewave input data:
c       Treal = COS( I*PIN)
c       Timag = SIN( I*PIN)

c  For testing with squarewave input:
cc IF (I .lt. N/2) THEN

cc   Treal = 1.0
cc   Timag = 0.5
cc ELSE
cc   Treal = 0.0
cc   Timag = 0.0
cc ENDIF
C  For testing with ramp function input data:
     Treal = I - 1.0
     Timag = Treal + 0.5
     X(I) = CMPLX (Treal, Timag)
     X1(I) = CMPLX (Treal, Timag)
     X2(I) = CMPLX (Treal, Timag)
     X3(I) = CMPLX (Treal, Timag)
100   CONTINUE
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
   IF (TIMEIT .ne. 0) THEN

     CALL fft (X2, M, N)
cc Subroutine fft is Decimation-In-Time, Fortran version.

c        CALL ditt(X, M, N,W,IREV)
         CALL diff(X, M, N,W,IREV)
   ENDIF

ccccccccccccccccccccccccccccccccccccc
   IF (IREV .ne. 0) THEN
   IF (TIMEIT .eq. 0) THEN
     call vcompare(X,X2,2*N)
     call cmags(X,N,ASQR)
c cmags to take squared magnitude of complex values
     call cmags(X2,N,ASQR2)
```

```
c----------------------c
C print non-zero results:
   J=0
   DO 700 I = 1,N
   IF ((ASQR(I) .GT. 1.0) .OR. (ASQR2(I) .GT. 1.0)) THEN
     WRITE (6,22) (I-1), ASQR(I), ASQR2(I)
22   FORMAT (' I-1=',I4,' ASQR(I)= ',F14.2, ' ASQR2(I)= ',F14.2//)
   J = J+1
   IF (J .GT. 32) GOTO 725
   ENDIF
700 CONTINUE

725     CALL TIME
   ENDIF
   ENDIF

   IF (TIMEIT .ne. 0) THEN
ccccccccccccccccccccccccccccccccccccccc
cc- Timing loop follows:

     print *,' Start Ass.FFT'
   IF (CACHETIME .eq. 0) THEN
     DO 500 I = 1, IT,4
C Reuse same array, so cache fill and writeback time NOT included.
          CALL diff(X, M, N,W,IREV)
          CALL diff(X, M, N,W,IREV)
          CALL diff(X, M, N,W,IREV)
500       CALL diff(X, M, N,W,IREV)
   ELSE
     DO 504 I = 1, IT,4
C Alternating between X,X1,X2,X3 should provide cache misses.
          CALL diff(X, M, N,W,IREV)
          CALL diff(X1, M, N,W,IREV)
          CALL diff(X2, M, N,W,IREV)
504       CALL diff(X3, M, N,W,IREV)
   ENDIF
     print *,' END Ass. FFT'
ccccccccccccccccccccccccccccccccccccccc
   ENDIF
     STOP
     END
```

6

```
c----------------------c
      subroutine vcompare(res,exp,n)
c VCOMPARE compares 2 REAL vectors, prints out 1st few miscompares
c
      integer n, errcnt
      real res(n), exp(n)

      write(6,12)
12    format('*** VCOMPARE: vector comparison beginning ***')

   data errcnt/0/
        do 30 i = 1,n
             if(AINT(res(i)) .ne. AINT(exp(i))) then
c  {print out error, exit if alot already}
120                  print *,'*** Error in compares ***'
                     write(6,121) i
121                  format('  Item number = ',I6)
                     write(6,124) res(i), exp(i)
124                  format('  Res_=',F14.2,' Expected_=',F14.2)
                     errcnt = errcnt + 1
                     if (errcnt .gt. 19) then
                          return
                     end if
               end if
30      continue

   if (errcnt .eq. 0) then
190   print *,' *** vector compares SUCCESSFUL ***'
   end if

99 return
   end
c----------------------c
```

```
C---------------
C File: ditt.f
C  6/15/89

C Intel assumes no responsibility for use or misuse of this code.

C  FFT - Decimation in TIME, radix-2, inplace, 1-dimen
C Inputs:
C   A= complex array of input, up to 1024 pts, single-prec float
C   M= log of number of pts
C    = (Number of stages of FFT)
C   N = number of points. ie, N= 2**M = number of pts
C   W= complex array of twiddle factors, length=N/2.
C   REV= ignored parameter.
C
C Outputs:
C  A= complex fft of input A. Correct order (bit-reversal done).
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

   subroutine ditt(a,m,N,W,REV)
   integer m,N, i, REV,wlimit
   integer offset, stage, groups, wincr,powers2(0:10)
   complex a(n),w(N/2),temp

   data powers2 /1,2,4,8,16,32,64,128,256,512,1024/
C Powers2 to avoid calls to POW, DIV

C Twiddle factor array w(i) has (cos,-sin) of 2pi*i/N
CC Assume the caller provides w(i) constants ALREADY initialized
C------------
C Pre-touch data, lock into cache, for 8kByte fft:
   IF (N .gt. 513) THEN
           call fetch(a,%VAL(n))
   ENDIF
C------------
   call bitrev(a,%VAL(M),n)
C Bitreversal of input needed for in-place decim in time FFT, to avoid
C  fetching twiddle-factors in bitrev order.
   wlimit = 8*((N/2) - 1)

DO 20 stage = 1,m
           groups = powers2(m-stage)
C groups=number of times the twiddle factors are used, ie, the number of
C  smaller DFTs the stage is split into.

C   offset gets 1,2,4,8,...N/2
           offset = powers2(stage-1)
           wincr = groups
           call ditstep(a,w,groups,offset,wincr,wlimit)
20 CONTINUE

   RETURN
   END
C------------
```

6

```
//--------------
// ditstep.ss: do one stage of fft butterflies
// DIT = Decimation in Time, radix-2, inplace, 1-dimension
// (C) Copyright 1989 INTEL Corporation. ALL RIGHTS RESERVED.
// 7/15/89
//------------
// Intel is not responsible for use nor for misuse of this program.
//------------
// Do one entire stage (n/2 butterflies). Sample invocation:
//  call ditstep(a,w,groups,offset,wincr,wlimit)
//================================================
// Inputs:
//   A= complex array of input, single-prec float
//        (complex stored as 4byte real, 4byte imag contiguously)
//   W= pointer to array of twiddle factors. Assuming W(k) is
//      CMPLX(cos(2pi*k/N)),-sin(2pi*k/N)) for k=0 to (N/2)-1.
//   offset = distance (except for scale-by-8byte sizeof(complex)) between
//      the 2 input values for each butterfly.
//      Offset also is the number of butterflies done per "group".
//   groups = N/(2*offset). The number of sub-DFTs this stage is split into.
//   wincr = distance (except for scale-by-8byte sizeof(complex)) between
//      successive w values for successive butterflies
//   wlimit =max index, in bytes, of W table.
//
// Outputs:
//   A= complex radix-2 butterflied version of input.
//
//------------------
define(astart, r16)  // input data base address
define(wstart,r17) //twiddle array ptr. Because w-contents depend on N,
// we will assume the caller has initialized w() array.
define(groups,r18) //groups=number of sub-DFTs this stage is split into.
define(offset,r19)  //offset (initially elements, mult by 8 to get bytes)
// between node and its dual (the 2 numbers to butterfly, ie. A and B)
define(wincr,r20) //increment between successive W values. Remains constant
// within a given stage.
define(wlimit,r21) //max index, in bytes, of W table.
define(wind,r22) //current index, in bytes, of W table.
define(offset2,r23)  //offset*2

define(decrem,r24) //bla decrement
define(somecount,r25) // bla counter

define(FEtch, r26)  //pointer to 1st component of butterfly (load)
define(STore,r27)  // " " 1st component of butterfly (store)

define(offsetp8,r28) //offset+8
```

```
// f4:f7 spare
define(ARe,f12)  //element A, real component
define(AIe,f13) // " ", imag
define(ARo,f14) // extra A value, for prefetch (o="odd")
define(AIo,f15)
define(BRe,f16)  //element B, real component
define(BIe,f17)
define(BRo,f18) // extra B value, for prefetch
define(BIo,f19)

define(ERe,f20) //A+(B*W), real (ER = AR + BR)
define(EIe,f21) // "   imag "
define(ERo,f22) // previous loop's value
define(EIo,f23) // "   imag "

define(FRe,f24) //A-(B*W), real
define(FIe,f25) // "       imag "
define(FRo,f26) // previous loop's value
define(FIo,f27) // "   imag "

define(PR, f28) //(B*W), real
define(PI, f29) //(B*W), imag

define(WRe,f30)  //W (twiddle factor), real part
define(WIe,f31)  // " " , imag

define(WRo,f10)  //W (twiddle factor), real part (EXTRA copy)
define(WIo,f11)  // " " , imag

.text
.align .quad
_ditstep_::
 ld.l      0(groups),groups  //fix Fortran call-by-ref
 ld.l      0(offset),offset  //
 shl       3,offset,offset  // change from elements to bytes
 shl       1,offset,offset2
 adds      8,offset,offsetp8

 fst.q   f8 ,-16(sp)++ //save "local" regs
 fst.q   f12,-16(sp)++ // " "

 adds      -1,groups,groups // pre-decrement for bnc usage, or bla usage
 adds      -16,r0,decrem //bla decrement

//  We code the last 2 stages as special cases:
//--------
 xor       8,offset,r0 //offset=1, special case, no complex mult, funny addressing
 bc        offset_1// (ASSUMING offset=1 means wincr=0, and no twiddle used)
 xor       16,offset,r0 //offset=2, special case, no complex mult
 bc        offset_2
//--------
 ld.l      0(wincr),wincr
 ld.l      0(wlimit),wlimit
```

6

```
 pfadd.ss f0,f0,f0
 pfadd.ss f0,f0,f0
 pfadd.ss f0,f0,f0  // init A1,A2,A3=0
 pfmul.ss f0,f0,f0
 pfmul.ss f0,f0,f0
 pfmul.ss f0,f0,f0
//--------
 // init pointers:
 shl       3,wincr,wincr   //scale for bytes.
 shl       1,wincr,wind //init wind =2*wincr

 pfld.d 0 ( wstart),f0
 pfld.d wincr ( wstart),f0
 adds      -8,astart,FEtch
 pfld.d wind (wstart),f0
 adds      wincr,wind,wind //wind now 3*wincr
// here fetch first set of B,W before bla-loop
 pfld.d  wind (wstart),WRe
 adds      wincr,wind,wind
//first Bfetch from offset, then 1st afetch from 0.
 fld.d  offsetp8 (FEtch),BRe //first B value

 and       wlimit,wind,wind //modulo-wlimit the w index
// We do modulo=addressing on W(), to keep the pfld pipeline full. We
//  never do a W-fetch beyond the end of the table.
// And the modulo-check needs to be done only every 4th pfld, as always
//  we use a multiple of 4 W() factors.

d.r2apl.ss f0,f0,f0  //clear Treg.
 adds -32,offset,somecount // bla counter (predecrement by 4 elements)
// -----------
// Definitions for pipe diagram:
//   Anew = E = A+(B*W)
//   Bnew = F = A-(B*W)
//     Let P=(B*W).
//--------------
//  (the complex multiply product, P, broken into 4 real mult and 2 adds):
//   WR = cos(), WI=-sin().
//   PR = K - L; where K= WR*BR, L=WI*BI
//   PI = N + M; where N= WI*BR, M=WR*BI
//   ER = AR + PR (Overwrites AR)
//   EI = AI + PI (   "     AI)
//   FR = AR - PR (   "     BR)
//   FI = AI - PI (   "     BI)

// For 1st time thru inner--loop, don't have correct values to store.
// Must do 1 loop before the loop, sans the stores.
//----------------
first_bfly::  //fill pipe
```

```
                        // KR...KI...M1....M2....M3    T   A1....A2....A3....Write
d.r2pt.ss WRe,f0,f0     // WRe  -    -     -     -     -   -     -     -     -
 pfld.d  wind (wstart),WRo
d.i2st.ss  WIe,f0,f0    //       WIe
 adds        wincr,wind,wind
d.r2apl.ss f0 ,BRe,f0 //           K0    -     -     -   -     -     -     -
 fld.d  8 (FEtch)++,ARe //first A value
d.pfmul.ss WIe,BIe,f0 //           L0    K0    -     -   -     -     -     -
 pfld.d  wind (wstart),WRe
d.r2pt.ss WRo,BIe,f0  // WRo       M0    L0    K0    -   -     -     -     -
 fld.d  offsetp8 (FEtch),BRo
d.ratls2.ss f0 ,PR ,f0//           -     M0    L0    K0  -     -     -     -
 adds        wincr,wind,wind
d.i2st.ss  WIo,BRe,f0 //       WIo  N0    -     M0    K0  K-L0  -     -     -
 nop
//.....................................................................
d.r2apl.ss f0 ,BRo,f0 //           K1    N0    -     M0  -     PR0
 and         wlimit,wind,wind
d.pfsub.ss f0 ,PI ,f0 //           K1    N0    -     M0  -     -     PR0
 fld.d  8 (FEtch)++,ARo
d.pfadd.ss ARe,PR ,PR //           K1    N0    -     M0  ER0   -     -     PR0
 fld.d  offsetp8 (FEtch),BRe
d.pfmul.ss WIo,BIo,f0 //           L1    K1    N0    M0  ER0   -     -     -
 nop
d.r2pt.ss WRe,BIo,f0  // WRe       M1    L1    K1    M0  M+N0  ER0   -     -
 bla         decrem,somecount,restart //init LCC
d.ratls2.ss ARe,PR ,f0//           -     M1    L1    K1  FR0   PI0   ER0   -
 nop
restart::
d.i2st.ss WIe,BRo,ERe//       WIe  N1    -     M1    K1  K-L1  FR0   PI0   ER0
 adds        -16,astart,STore  // ptrs init 16 low, for fst.q instructions
//-------------------
// Each butterfly = 1 complx multiply, 1 complx add, 1 complx subtract
// =         4 multiply, 3 add, 3 subtract
//           3 8-byte fetches (A, B, W)
//           2 8-byte stores (A, B)
//
// 7 cycles per butterfly
//
// inner_loop: iterates "offset/2" times
//   for each group. It does 2 butterflies per iteration

// AR/AI fetches need to be a cycle behind BR/BI fetches here. So we
//  must index with offset+8 into B.
// AR is used 1/2 loop before AI.
// Pattern= AI0,AR1,BR2,BI2;AI1,AR2,BR3,BI3.

inner_loop::          // KR...KI...M1....M2....M3    T   A1....A2....A 3....Write
d.r2apl.ss AIe,BRe,PI //           K2    N1    -     M1  EI0   PR1   FR0   PI0
 pfld.d  wind (wstart),WRo
d.pfsub.ss AIe,PI ,FRe//          K2    N1    -     M1  FI0   EI0   PR1   FR0
 fld.d  8(FEtch)++,ARe
d.pfadd.ss ARo,PR ,PR //          K2    N1    -     M1  ER1   FI0   EI0   PR1
 fld.d  offsetp8 (FEtch),BRo
d.pfmul.ss WIe,BIe,f0 //          L2    K2    N1    M1  ER1   FI0   EI0   -
 adds   wincr,wind,wind
```

```
d.r2pt.ss WRo,BIe,EIe // WRo        M2    L2    K2        M+N1  ER1   FI0   EI0
 pfld.d wind (wstart),WRe
d.ratls2.ss ARo,PR ,FIe//          -     M2    L2    K2  FR1   PI1   ER1   FI0
 adds   wincr,wind,wind
d.i2st.ss  WIo,BRe,ERo//      WIo  N2    -     M2    K2  K-L2  FR1   PI1   ER1
 and    wlimit,wind,wind //modulo.
                      // KR...KI...M1....M2....M3    T   A1....A2....A3....Write
d.r2apl.ss AIo,BRo,PI //           K3    N2    -     M2  EI1   PR2   FR1   PI1
 nop
d.pfsub.ss AIo,PI ,FRo//           K3    N2    -     M2  FI1   EI1   PR2   FR1
 fld.d  8 (FEtch)++,ARo
d.pfadd.ss ARe,PR ,PR //           K3    N2    -     M2  ER2   FI1   EI1   PR2

 fld.d offsetp8 (FEtch),BRe
d.pfmul.ss WIo,BIo,f0 //           L3    K3    N2    M2  ER2   FI1   EI1   -
 nop
d.r2pt.ss WRe,BIo,EIo // WRe       M3    L3    K3        M+N2  ER2   FI1   EI1
 fst.q ERe,16(STore)++ //update ERe/EIe/ERo/EIo
d.ratls2.ss ARe,PR ,FIo//          -     M3    L3    K3  FR2   PI2   ER2   FI1
 bla decrem,somecount, inner_loop
d.i2st.ss  WIe,BRo,ERe//      WIe  N3    -     M3    K3  K-L3  FR2   PI2   ER2
 fst.q  FRe, offset (STore)
 //update FRe/FIe/FRo/FIo

end_inner_loop:: //KEEP Pipelines full
// RE-init pointers for fetches
d.fiadd.ss f0,f0,f0
 adds   offset2,astart,astart //bump to next group
      //redo A,B fetches, with proper ptr.
d.fiadd.ss f0,f0,f0
 fld.d  offset (astart),BRe //get first BR/BI in next group
d.fiadd.ss f0,f0,f0
 adds   -8,astart,FEtch

last_bfly:: //do final 2 butterflies, start next group
                      // KR...KI...M1....M2....M3    T   A1....A2....A3....Write
d.r2apl.ss AIe,BRe,PI //           K0    N3    -     M3  EI2   PR3   FR2   PI2
 pfld.d wind (wstart),WRo
d.pfsub.ss AIe,PI ,FRe//           K0    N3    -     M3  FI2   EI2   PR3   FR2
 fld.d  8(FEtch)++,ARe
d.pfadd.ss ARo,PR ,PR //           K0    N3    -     M3  ER3   FI2   EI2   PR3
 fld.d  offsetp8 (FEtch),BRo
d.pfmul.ss WIe,BIe,f0 //           L0    K0    N3    M3  ER3   FI2   EI2   -
 adds       wincr,wind,wind
d.r2pt.ss WRo,BIe,EIe // WRo       M0    L0    K0        M+N3  ER3   FI2   EI2
 pfld.d wind (wstart),WRe
d.ratls2.ss ARo,PR ,FIe//          -     M0    L0    K0  FR3   PI3   ER3   FI2
 adds       wincr,wind,wind
d.i2st.ss  WIo,BRe,ERo//      WIo  N0    -     M0    K0  K-L0  FR3   PI3   ER3
 and        wlimit,wind,wind //modulo
//.....................................................
d.r2apl.ss AIo,BRo,PI //           K1    N0    -     M0  EI3   PR0   FR3   PI3
 adds -32,offset,somecount // reset bla counter
d.pfsub.ss AIo,PI ,FRo//           K1    N0    -     M0  FI3   EI3   PR0   FR3
 fld.d  8 (FEtch)++,ARo
```

```
d.pfadd.ss ARe,PR ,PR //          K1    N0    -     M0  ER0   FI3   EI3   PR0
 fld.d  offsetp8 (FEtch),BRe
d.pfmul.ss WIo,BIo,f0 //          L1    K1    N0    M0  ER0   FI3   EI3   -
 bla        decrem,somecount,nowhere //re-init LCC=1
d.r2pt.ss WRe,BIo,EIo // WRe      M1    L1    K1        M+N0  ER0   FI3   EI3
 adds -1,groups,groups
nowhere::
d.ratls2.ss ARe,PR ,FIo//         -     M1    L1    K1  FR0   PI0   ER0   FI3
 fst.q  ERe,16(STore)++
d.fnop
 bnc.t restart //branch on value of groups
d.fnop
 fst.q  FRe, offset (STore)

end_last_bfly::
d.fnop
 br endit
fiadd.ss f0,f0,f0
 fst.q  FRe, offset (STore) //repeated for bnc.t untaken case
.align      .quad
//=============================================================
offset_1::
// want FEtch=0,2,4,6,8,... elements. ASSUMING wincr=0,
//  and that w=(1,0), so that no complex mult needed.
//  E=A+B, F=A-B. (Per double-butterfly loop:  8 pfadd,4 dword fld, 4 fst,
//  1 bla)(fld.q used to reduce # flds)
//  Performance = 4 cyc/bfly best case.

//Redefine regs for fld.q,fst.q usage, when A and B adjacent:
define(AR3,f12)  //element A, real component
define(AI3,f13) // " ", imag

define(BR3,f14) //element B, real component
define(BI3,f15)
define(AR4,f16) // extra A value, for prefetch
define(AI4,f17)
define(BR4,f18)
define(BI4,f19)

define(ER3, f20) //A+B, real (ER = AR + BR)
define(EI3, f21) // "   imag "
define(FR3, f22) //(A-B), real
define(FI3, f23) // "     imag

define(ER4,f24) //A+B, real
define(EI4,f25) // "   imag
define(FR4,f26) //(A-B), real
define(FI4,f27) // "   imag
//=========================================
 adds       -16,astart,FEtch
 fld.q  16 (FEtch)++,AR4
 adds       -1,groups,somecount // bla counter (predecremented already by 1)
   //using groups=blacount on the offset_1 loop, intentionally.
 adds       -16,FEtch,STore
//startup the loop:
```

```
// ---------------------// A1......A2......A3......Write:
d.pfadd.ss AR4,BR4,f0   // ARn+BRn -       -       -
 fld.q  16 (FEtch)++,AR3
d.pfadd.ss AI4,BI4,f0   // AIn+BIn ERn     -       -
 adds       -2,r0,decrem //2 bflies per loop
d.pfsub.ss AR4,BR4,f0   // ARn-BRn EIn     ERn     -
 bla decrem,somecount, offset1_loop  //init LCC
d.pfsub.ss AI4,BI4,ER4  // AIn-BIn FRn     EIn     ERnext
 nop
// ---------------------// A1......A2......A3......Write:
offset1_loop::
d.pfadd.ss AR3,BR3,EI4  // AR+BR   FI-     FR-     EI-
 nop
d.pfadd.ss AI3,BI3,FR4  // AI+BI   ER      FI-     FR-
 fld.q  16 (FEtch)++,AR4
d.pfsub.ss AR3,BR3,FI4  // AR-BR   EI      ER      FI-
 fst.q      ER4,16(STore)++
d.pfsub.ss AI3,BI3,ER3  // AI-BI   FR      EI      ER
 nop
d.pfadd.ss AR4,BR4,EI3  // AR2+BR2 FI      FR      EI
 fld.q  16 (FEtch)++,AR3
d.pfadd.ss AI4,BI4,FR3  // AI2+BI2 ER2     FI      FR
 nop
d.pfsub.ss AR4,BR4,FI3  // AR2-BR2 EI2     ER2     FI
 bla decrem,somecount, offset1_loop
d.pfsub.ss AI4,BI4,ER4  // AI2-BI2 FR2     EI2     ERnext
 fst.q  ER3,16(STore)++
//---------
end_offset1_loop::
d.fiadd.ss f0,f0,f0
 br endit
fiadd.ss f0,f0,f0
  nop
//---------
.align .quad
offset_2::
// want FEtch=0,1;4,5;8,9;12,13;... elements.
// ASSUMING wincr=N/4  (W_addr=0,N/4,0,N/4,0,...). Trivial W() factors.
//  Even-indexed elements identical to offset_1,W=W0, no complex mult.
//    So              EReven=(AR+BR), EIeven=(AI+BI).
//    So              FReven=(AR-BR), FIeven=(AI-BI).

//  Odd components have W=(0,-1).  So B*W = (BI,-BR).
//   So ERodd=Re(A+(B*W)) = (AR+BI)    EIodd=(AI-BR).
///  So FRodd=Re(A-(B*W)) = (AR-BI)    FIodd=(AI+BR).
//  Each fld.q fetches AReven,AIeven,ARodd,AIodd.

//Assume ERe,EIe,ERo,EIo are 4 contiguous regs.
//Assume FRe,FIe,FRo,FIo are 4 contiguous regs.
//Assume ARe,AIe,ARo,AIo are 4 contiguous regs.
```

```
 adds       -16,astart,FEtch
 fld.q  16 (FEtch)++,ARe
 fld.q  16 (FEtch)++,BRe
 adds       0,groups,somecount //bla counter
//startup the loop:
// --------------------// A1......A2......A3......Write:
  pfadd.ss ARe,BRe,f0   // AR+BRe                    -
  pfadd.ss AIe,BIe,f0   // AI+BIe  ER                -
d.pfadd.ss ARo,BIo,f0   // ARo+BIo EI      ER        -
 nop
d.pfsub.ss AIo,BRo,ERe  // AIo-BRo ERo     EI        ER
 nop
d.pfsub.ss ARe,BRe,EIe  // AR-BRe  EIo     ERo       EI
 ads       -1,r0,decrem //2 bflies per loop,but groups is half desired value.
d.pfsub.ss AIe,BIe,ERo  // AI-BIe  FR      EIo       ERo
 adds       -16,astart,STore
d.pfsub.ss ARo,BIo,EIo  // ARo-BIo FI      FR        EIo
 bla decrem,somecount, offset2_loop  //init LCC
d.pfadd.ss AIo,BRo,FRe  // AIo+BRo FRo     FI        FR
 nop
offset2_loop::
d.fnop
 fld.q  16 (FEtch)++,ARe//fetch AR,AI,ARo,AIo

d.fnop
 fld.q  16 (FEtch)++,BRe
// -------              // A1......A2......A3......Write:
d.pfadd.ss ARe,BRe,FIe  // AR+BRe  FIo     FRo       FI
 nop
d.pfadd.ss AIe,BIe,FRo  // AI+BIe  ER      FIo       FRo
 nop
d.pfadd.ss ARo,BIo,FIo  // ARo+BIo EI      ER        FIo
 fst.q      ERe,16(STore)++ //update ER ,EI ,ERo,EIo
d.pfsub.ss AIo,BRo,ERe  // AIo-BRo ERo     EI        ER
 nop
d.pfsub.ss ARe,BRe,EIe  // AR-BRe  EIo     ERo       EI
®nop
d.pfsub.ss AIe,BIe,ERo  // AI-BIe  FR      EIo       ERo
 fst.q  FRe,16(STore)++
d.pfsub.ss ARo,BIo,EIo  // ARo-BIo FI      FR        EIo
 bla decrem,somecount,offset2_loop
d.pfadd.ss AIo,BRo,FRe  // AIo+BRo FRo     FI        FR
 nop
endit::
// restore regs
fiadd.ss f0,f0,f0 //exit DIM
 fld.q   0(sp),f12
fiadd.ss f0,f0,f0 //last DIM pair
 fld.q   16(sp),f8
adds    32,sp,sp
 bri r1
  nop
//=============================================
```

```
C---------------
C File: dirr.f
C  FFT - Decimation in Freq, radix-2, inplace, 1-dimen,
C        REAL input
C   Intel is not responsible for use nor misuse of this code.

C  8/14/89

C Inputs:
C   A= REAL array of input, up to 1024 pts, single-prec float
C   M= log of number of pts
C    = (Number of stages of FFT)
C   N = number of points. ie, N= 2**M = number of pts
C   W= complex array of twiddle factors, length N/2.
C   REV= 0 if bitreversed output ok. 1=must re-order output
C     (REV will be ignored, and output will be properly ordered. Bit
C      reversal WILL be done.)
C
C Outputs:
C  A= complex fft of input A, but only the positive frequency half.
C     Length = N/2+1 complex numbers. A(0:n/2)
C
   subroutine dirr(a,m,N,W,REV)
   integer m,N, i, j,k, REV,wlimit
   integer offset, stage, groups, wincr,powers2(0:10)
   real a(N)
   complex w(N/2),temp

   data powers2 /1,2,4,8,16,32,64,128,256,512,1024/
C Powers2 to avoid calls to POW, DIV

C Twiddle factor array w(k) has (cos,-sin) of 2pi*k/N
CC Assume the caller provides w(k) constants ALREADY initialized
C------------
C Pre-touch data, for 8kByte fft: (2048 points real)
   IF (N .gt. 1025) THEN
           call fetch(a,%VAL(n/2))
   ENDIF
C------------
   wlimit = 8*((N/2) - 1)

C "DO 20" stage-loop: doing Complex FFT on length N/2 array. Twiddles are
C   for a length N array, so wincr gets scaled by 2.
   DO 20 stage = 1,m-1
           groups = powers2(stage-1)
C groups=number of times the twiddle factors are used, ie, the number of
C  smaller DFTs the stage is split into.

C  offset gets N/4,N/8,N/16,...
           offset = powers2(m-1-stage)
           wincr = groups * 2
           call difstep(a,w,groups,offset,wincr,wlimit)
20 CONTINUE

   call bitrev(a,%VAL(M-1),n/2)
   call realfix(a,w,%VAL(n))

   RETURN
   END
C------------
```

```
// realfix.ss:  This is i860(tm) CPU assembly code to revise data from an
//   N/2 length Complex FFT.
//   (assumes the input data fed to Complex FFT was N real values)
//
//  INTEL is not responsible for use nor misuse of this code.
//
// 8/14/89
//   This 18-cycle-butterfly loop may be sub-optimal.
//
//   output = overwrite the data array used for input. Results are
//    complex.     Re0,Im0,Rel,Iml,..., Re(N/2),Im(N/2).
//     NOTE that output array is 1 element longer than input.
//
//  Input is H(k), output is F(k)...
//   F(k)=.5*( H(k)+ Hconj(N/2-k) -j*(H(k) -Hconj(N/2-k))*Wconj(k))
//
//  Algorithm from "Numerical Recipes in C", by Flannery, Press, Teukolsky, and
//    Vetterling, Cambridge Univ. Press 1988, p.417.
//*************************/

//* The C-version of realfix: */ void realfix_(a,w,n)
///*Input =
// a(0:n+1): length n/2+1 complex array. Entries 0:n/2-1 are the complex FFT
// *   result, in correct (NON BIT REVERSED) order. Entry n/2 is undefined.
// * w: length n/2 complex array of twiddles. (cos,-sin(2pi*k/n))
// * n: call-by-value, number of REAL input samples

// *Output =
// *  a(0:n+1): length n/2+1 complex array.
// *    Format is Re0,Im0,Rel,Iml,..., Re(N/2),Im(N/2).
// *    NOTE: To generate entire N-length complex output spectrum, you can copy
// *     conjugate of element(i) to element(N-i).
// */
//float a[], w[]; int n;   { int aptr,bptr, wptr; float half=0.5,
//      AR,AI,BR,BI, /* input values for A,B*/
//      PR,PI,SR,SI,DR,DI, /*temporary differences,sums,products*/
//      K,L,M,N, /*temporary products */
//      ER,EI,ERD,EID,
//      FR,FI,FRD,FID,
//      WR,WI;

///*We do first and last elements as special case(Imag=0, W=(1,0))*/
// AR = a[0];       AI = a[1];
// a[0] = AR + AI; a[1] = 0;
// a[n] = AR - AI; a[n+1] = 0;
```

```
//for(aptr=2, bptr=(n-2), wptr=2; aptr < n/2; aptr +=2, bptr -=2, wptr +=2)
//{WR = w[wptr];              WI = w[wptr+1];
// AR = a[aptr];    AI = a[aptr+1];
// BR = a[bptr];    BI = a[bptr+1];
// /* aptr =2,4,6...,14;  bptr=30,28,26,...,18 (if n=32) */
// /* Note that there is no need to revise the value at the middle of the
//    list, as it is already correct. (.5*(H(n/4)+Hconj(n/4)) */
// SI = (AI + BI);
// DR = (BR - AR);
// K = WR*SI; L= WI*DR;     PR = K-L;
// M = WR*DR; N= WI*SI;     PI = M+N;
// SR = (AR + BR);
// DI = (AI - BI);

// ERD = SR+PR; ER = half*ERD;
//  a[aptr] = ER;
// EID = DI+PI; EI = half*EID;
//  a[aptr+1]= EI;
// FRD = SR-PR; FR = half*FRD;
//  a[bptr] = FR;
// FID = PI-DI; FI = half*FID;
//  a[bptr+1]= FI;  } /*end of for-loop */ }
//************* End of C-code for realfix.**********************
.text
.align .quad
//--------------
define(astart, r16)  //input data base address

define(wptr,r17) // pointer to W table. Because w-contents depend on N,
// we will assume the caller has initialized w() array.
define(N,r18) //
define(aptr, r20)  //pointer to 1st component of butterfly (load)
define(bptr, r21)  //pointer to 2nd component of bfly (load); DOWNCOUNTER

define(decrem,r24) //bla decrement
define(count,r25) // bla counter

define(WR, f18)  //W (twiddle factor), real part
define(WI, f19)  // " " , imag

define(AR, f12)  //element A, real component
define(AI, f13) // " ", imag
define(ARo,f14) // extra A value, for prefetch (o="odd")
define(AIo,f15)
define(BR, f16)  //element B, real component
define(BI, f17)

define(ER, f20) //Result of butterfly which overwrites AR
define(EI, f21) // "   " " " AI

define(half,f22)  //constant 0.5

define(FR, f24) //Result of butterfly which overwrites BR
define(FI, f25)
define(PR,f26)
define(PI,f27)

define(DR, f28)
define(DI, f29)
```

```
define(SR, f30) //Sum of A+B, real part
define(SI, f31) // " ", imag "

.data
.align .double
halfloc::  .float 0.5
//--------
.text
.align .quad
_realfix_::
 fst.q   f12,-16(sp)++ //save "local" regs
 adds      -4,r0,decrem //bla decrement
//--------
// We do not bother to initialize FP pipes to zero here, as we assume
//  this routine is called after another,"safe", pipelined FP routine.

pfld.l halfloc,f0
pfld.d 8( wptr)++,f0 //skip W(0) intentionally. Is a trivial (1,0) value
// init pointers:
 adds       0,astart,aptr
pfld.d 8( wptr)++,f0
 shl        2,N,bptr  //bptr=total # bytes of input data
pfld.d 8( wptr)++,half  //0.5 into an fpr
 adds       bptr,astart,bptr  // bptr points to a(N)

// here fetch first set of A,B,W before bla-loop
pfld.d 8( wptr)++,WR
 fld.d 0 (aptr),AR //for 1st and last elements
 adds -8,N,count // bla counter (predecrement by 2 butterflies worth)
// -----------
// Do n/4 butterflies: (computing only N/2 elements of complex output, because
//    the second N/2 are just complex conjugates of the 1st N/2)

// Definitions for pipe diagram:
//   WR = cos(), WI=-sin().
//   DR = BR - AR; (diffence of Real components of A,B)
//   DI = AI - BI; (diffence of Imag components)
//   SR,SI = sum of A,B
//   PR = K - L; where K= WR*SI, L=WI*DR
//   PI = M + N; where M= WR*DR, N=WI*SI
//  (ER,EI)=complex result to overwrite A.
//  (FR,FI)=" " " " B.

first_fly::  //fill pipe.
// For 0th butterfly:
// AR = a[0];       AI = a[1];
// a[0] = AR + AI;  a[1] = 0;
// a[n] = AR - AI;  a[n+1] = 0;

                    // KR..KI..M1....M2....M3   T   A1....A2....A3....Write
  r2pt.ss  f0,f0,f0  // 0       0
  mrm1p2.ss AR,AI,f0 //         0     0     -        ER0   -     -     -
  mrm1s2.ss AR,AI,f0 //         0     0     0        FR    ER    -     -
 fld.d  8 (aptr)++,AR
 fld.d  -8(bptr)++,BR
d.pfadd.ss f0,f0,f0  //         0     0     0        0     FR    ER    -
d.pfadd.ss  f0,f0,ER //         0     0     0        0     0     FR    ER0
```

```
d.ralp2.ss  AI ,BI ,FR //          -     0     0     -     SI1   -     -     FR0
 nop
d.mrmls2.ss BR ,AR ,EI //          -     -     0     -     DR1   SI1   -     EI0
 fst.d ER,-8(aptr)
d.mr2pt.ss  WR ,f0, FI // WR       -     -     -     -     -     DR1   SI1   FI0
 fst.d FR, 8(bptr)
d.ralp2.ss  BR ,AR ,SI //         K1     -     -     -     SR1   -     DR1   SI1
 andh 0x8000,count,r0  //check for negative
d.ml2tpm.ss WI ,DR ,DR //         L1     K1    -     -     -     SR1   -     DR1
 bnc  endfix
d.r2pt.ss   half,DR, f0 //half    M1     L1    K1    -     -     -     SR1   -
 nop
d.ml2ttpa.ss WI ,SI ,SR//         N1     M1    L1    K1    -     -     -     SR1
 nop
d.i2st.ss     f0 ,f0 ,f0//     f0  -     N1    M1    K1    PR1   -     -     -
 nop
                      // KR..KI..M1....M2....M3   T     A1....A2....A3....Write
d.ratls2.ss AI ,BI ,f0 //          -     -     N1    M1    DI1   PR1   -     -
 nop
d.i2pt.ss    f0 ,f0,  f0//     f0  -     -     -     M1    PI1   DI1   PR1   -
 fld.d  8 (aptr)++,AR
d.r2apl.ss  SR ,f0,  PR//          -     -     -     -     ERD   PI1   DI1   PR1
 fld.d  -8(bptr)++,BR
d.rals2.ss  SR ,PR, DI //          -     -     -     -     FRD   ERD   PI1   DI1
 pfld.d  8( wptr)++,WR
d.r2apl.ss  DI ,f0,  PI//          -     -     -     -     EID   FRD   ERD   PI1
 nop
d.rals2.ss  PI ,DI ,f0 //         ER1    -     -     -     FID   EID   FRD   -
 nop
d.ralp2.ss  f0 ,f0 ,f0 //         FR1   ER1    -     -     -     FID   EID   -
 nop
d.rals2.ss  f0 ,f0 ,f0 //         EI1   FR1   ER1    -     -     -     FID   -
 bla decrem,count,fix_loop
d.pfadd.ss  f0 ,f0 ,FI //         EI1   FR1   ER1    -     -     -     -FID
 nop
//-------------------
// Each butterfly = 1 complx multiply, 3 complx add, 1 real multiply
//  =        8 multiply, 10 add/subtract
//           3 8-byte fetches (A, B, W)
//           2 8-byte stores (E, F)
//
// approx. 18 cycles per butterfly
//
```

```
fix_loop::              // KR..KI..M1....M2....M3   T   A1....A2....A3....Write
d.mr2pt.ss  f0 ,FI ,ER // 0        FI1   EI1   FR1   -   -     -     -      ER1
 nop
d.mrmlp2.ss AI ,BI ,FR //          -     FI1   EI1   -   SI2   -     -      FR1
 nop
d.mrmls2.ss BR ,AR ,EI //          -     -     FI1   -   DR2   SI2   -      EI1
 fst.d ER,-8(aptr)
d.mr2pt.ss  WR ,f0, FI // WR       -     -     -     -   -     DR2   SI2    FI1
 fst.d FR, 8(bptr)
d.ralp2.ss  BR ,AR ,SI //          K2    -     -     -   SR2   -     DR2    SI2
 andh 0x8000,count,r0  //check for negative
d.ml2tpm.ss WI ,DR ,DR //          L2    K2    -     -   -     SR2   -      DR2
 bnc  endfix
d.r2pt.ss  half,DR, f0 //half      M2    L2    K2    -   -     -     SR2    -
 nop
d.ml2ttpa.ss WI ,SI ,SR//          N2    M2    L2    K2  -     -     -      SR2
 nop
d.i2st.ss   f0  ,f0 ,f0//     f0   -     N2    M2    K2  PR2   -     -      -
 nop
                        // KR..KI..M1....M2....M3   T   A1....A2....A3....Write
d.ratls2.ss AI ,BI , f0//          -     -     N2    M2  DI2   PR2   -      -
 nop
d.i2pt.ss   f0 ,f0,  f0//     f0   -     -     -     M2  PI2   DI2   PR2    -
 fld.d  8 (aptr)++,AR
d.r2apl.ss  SR ,f0,  PR//          -     -     -     -   ERD   PI2   DI2    PR2
 fld.d -8(bptr)++,BR
d.rals2.ss  SR ,PR,  DI//          -     -     -     -   FRD   ERD   PI2    DI2
 pfld.d  8( wptr)++,WR
d.r2apl.ss  DI ,f0,  PI//          -     -     -     -   EID   FRD   ERD    PI2
 nop
d.rals2.ss  PI ,DI ,f0 //          ER2   -     -     -   FID   EID   FRD    -
 nop
d.ralp2.ss  f0 ,f0 ,f0 //          FR2   ER2   -     -   -     FID   EID    -
 nop
d.rals2.ss  f0 ,f0 ,f0 //          EI2   FR2   ER2   -   -     -     FID    -
 bla decrem,count,fix_loop
d.pfadd.ss  f0 ,f0 ,FI //          EI2   FR2   ER2   -   -     -     -      FID
 nop
//---------
endfix::
// restore regs
fiadd.ss f0,f0,f0 //exit DIM
 fld.q   0(sp),f12
fiadd.ss f0,f0,f0 //last DIM pair
adds    16,sp,sp
 bri r1
  nop
//------------
```

```
      PROGRAM FFTTEST
c  file = real.f
C
C   1-D FFT TEST PROGRAM
C
C 8/14/89

C Intel assumes no responsibility for use or misuse of this code.
C-----------------
      PARAMETER (IREV=1)
      character*8 really
      PARAMETER (REALLY='real')
c     PARAMETER (REALLY='complex')
      PARAMETER (TIMEIT=0, CACHETIME=0)
c REALLY='real' means real-only input, otherwise assume complex input
      DATA IT/200000/
c      PARAMETER (N=2048,M=11)
      PARAMETER (N=1024,M=10)
c      PARAMETER (N=512,M= 9)
c      PARAMETER (N=256,M= 8)
c      PARAMETER (N=128,M= 7)
c      PARAMETER (N=64,M= 6)
c      PARAMETER (N=32,M= 5)
c       PARAMETER (N=16, M=4)
     PARAMETER (PI=3.1415926536)
     COMPLEX X2(N),X(N),X3(N), W(N/2)

   Real  ASQR(N),ASQR2(N),XR(N+2),XR1(N+2),XR2(N+2),XR3(N +2)
   complex wtemp
   real rtemp
C
   PRINT *,' FFT test program ....'
   print *,'================================'
   IF (IREV .eq. 0) THEN
    print *,'NOT counting time for bit-reversal.'
    print *,'DO NOT expect matching answers,without bit-rev'
   ELSE
    print *, 'Time for bit-reversal included.'
   ENDIF

    print *, 'Time for cache writeback and fills...'
   IF (CACHETIME .eq. 0) THEN
    print *,' NOT included, if iterating.'
   ELSE
    print *,' ... included.'
   ENDIF

    print *, '================================'
    print *, 'If iterating... Number of Iterations =',IT
    print *, '================================'
    print *, 'Number of Points     = ', N
    print *, '(',REALLY,' data)'
    print *, '================================'
```

```
 C-----------------
C Init twiddle factor array w(k) with (cos,-sin) of 2pi*k/N
   rtemp = 2.0*pi/N
   wtemp= CMPLX(cos(rtemp), -sin(rtemp))
   w(1) = (1.0, 0.0)
   DO  200  k  =  2,N/2
200        w(k) = wtemp * w(k-1)
cc print *,' W (twiddle) initialization completed.....'
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C   INITIALIZE input data
C
      DO 100 I = 1, N
c:constant:
c     Treal = 1.0
c     Timag = 0.0

c:squarewave:
cc IF (I .lt. N/2) THEN
cc   Treal = 1.0
cc   Timag = 0.5

cc ELSE
cc   Treal = 0.0
cc   Timag = 0.0
cc ENDIF
C: ramp function:
     Treal = I - 1.0
     Timag = Treal + 0.5
   IF (REALLY .ne. 'real') THEN
         X(I) = CMPLX (Treal, Timag)
         X2(I) = CMPLX (Treal, Timag)
         X3(I) = CMPLX (Treal, Timag)
   ELSE
     X(I)  = CMPLX (Treal,0.0)
     X2(I) = CMPLX (Treal,0.0)
     XR(I) = Treal
     XR1(I) = Treal
     XR2(I) = Treal
     XR3(I) = Treal
   ENDIF
100   CONTINUE
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      CALL fft (X2, M, N)
cc Subroutine fft is Decimation-In-Time, Fortran version.
      CALL dirr(XR,M,N,W,1)
c   (Assuming dirr produces inplace result, items 0:N/2 complex results)
```

6

```
ccccccccccccccccccccccccccccccccccccc
   IF (IREV .ne. 0) THEN
   IF (TIMEIT .eq. 0) THEN
   call vcompare(XR,X2,N/2+2)
   call cmags(XR,N/2+1,ASQR)
c cmags to take squared magnitude of complex values in X
   call cmags(X2,N,ASQR2)
c----------------------c
C print non-zero results:
   J=0

   DO 700 I = 1,N/2+1
   IF ((ASQR(I) .GT. 1.0) .OR. (ASQR2(I) .GT. 1.0)) THEN
      WRITE (6,22)  (I-1), ASQR(I), ASQR2(I)
22    FORMAT (' I-1=',I4,'  ASQR(I)= ',F14.2, '  ASQR2(I)= ',F14.2//)
   J = J+1
   IF (J .GT. 32) GOTO 725
   ENDIF
700        CONTINUE

725     CALL TIME
   ENDIF
   ENDIF

   IF (TIMEIT .ne. 0) THEN
ccccccccccccccccccccccccccccccccccccc
cc- Timing loop follows:

     print *,' Start Ass.FFT'
   IF (CACHETIME .eq. 0) THEN
     DO 500 I = 1, IT,4
C Reuse same array, so cache fill and writeback time NOT included.
          CALL dirr(XR, M, N,W,IREV)
          CALL dirr(XR, M, N,W,IREV)
          CALL dirr(XR, M, N,W,IREV)
500       CALL dirr(XR, M, N,W,IREV)
   ELSE
     DO 504 I = 1, IT,4
C Alternating between XR,XR1,XR2,XR3 should provide cache misses.
          CALL dirr(XR, M, N,W,IREV)
          CALL dirr(XR1, M, N,W,IREV)
          CALL dirr(XR2, M, N,W,IREV)
504       CALL dirr(XR3, M, N,W,IREV)
   ENDIF
     print *,' END Ass. FFT'
ccccccccccccccccccccccccccccccccccccc
```

```
   ENDIF

      STOP
      END
c----------------------c
     subroutine vcompare(res,exp,n)
c VCOMPARE compares 2 vectors, prints out 1st few miscompares
c
     integer n, errcnt
     real res(n), exp(n)

     write(6,12)
12   format('*** VCOMPARE: vector comparison beginning ***')

   data errcnt/0/
        do 30 i = 1,n
           if(AINT(res(i)) .ne. AINT(exp(i))) then
c  {print out error, exit if alot already}
120                print *,'*** Error in compares ***'
                   write(6,121) i
121                format('  Item number = ',I6)
                   write(6,124) res(i), exp(i)
124                format('  Res_=',F14.2,'  Expected_=',F14.2)
                   errcnt = errcnt + 1
                   if (errcnt .gt. 19) then
                          return
                   end if
           end if
30      continue

   if (errcnt .eq. 0) then
190  print *,' *** vector compares SUCCESSFUL ***'
   end if

99 return
   end
c----------------------c
```

6

```
C---------------
C file: fft.f

C FFT routine from Rabiner & Gold, 1975, who copied it
C  from Cooley, Lewis, Welch
C 6/02/89
C
C Decimation in Time, radix-2, inplace, 1-dimen
C Inputs:
C   A= complex array of input, up to 1024 pts, single-prec float
C        (maybe more than 1024, uncertain what limit is)
C  M= log of number of pts
C    = (Number of stages of FFT)
C  N = number of points.  ie, N= 2**M = number of pts
C
C Outputs:
C  A= complex fft of input A, in NON-bit-reversed order.
C
C w (twiddle factor) calculated by recursion. Supposedly takes 15% more
C  operations than keeping entire twiddle array as constants pre-allocated.
C
   subroutine fft(a,m,n)
   integer m,n, i, j,k, ndiv2,powers2(0:10)
   integer iplus,offset, stage, indexl, groups
   complex a(n),wtemp(2),w(11),temp

C Init twiddle factor array w() with (cos,-sin) of pi,pi/2,pi/4,...
       data w(1) /(-1.0,0.0) /
       data w(2) /(0.0,-1.0) /
       data w(3) /(0.7071068,-0.7071068)/
       data w(4) /(0.9238795,-0.3826834)/
       data w(5) /(0.9807853,-0.1950903)/
       data w(6) /(0.9951847,-0.0980171)/
       data w(7) /(0.9987955,-0.0490677)/
       data w(8) /(0.9996988,-0.0245412)/
       data w(9) /(0.9999247,-0.0122715)/
       data w(10) /(0.9999812,-0.0061359) /
       data w(11) /(0.9999953,-0.003068) /

   data powers2 /1,2,4,8,16,32,64,128,256,512,1024/
C Powers2 to avoid calls to POW, DIV

C Setup for bit-reversal loop:
   ndiv2 = n / 2
   j =1
C-----------
C "DO 7" loop to in-place-bit-reverse-shuffle input
   DO 7  i= 1, n-1
     IF (i .lt. j) THEN
           temp = a(j)
           a(j) = a(i)
           a(i) = temp
     ENDIF
     k = ndiv2
```

```
C  "While (j .gt. k)" /*decrease j by 2**something */
6    IF (j .gt. k) THEN
           j = j-k
           k = k / 2
           GOTO 6
     ENDIF
C  Add next lower power of 2 to j
7  j = j+k
C------------
C Special case for stage 1: no complex multiplies, simple add
C  (Performance enhancement)
   groups = 2
   offset = 1
   indexl = 1
C  i-loop iterates N/2 times for 1st stage (and would do twice N/4 x for 2nd)
CVD$        NODEPCHK
   DO 8 i = 1,n,2
     iplus = i + 1

     temp = a(iplus)
     a(iplus) = a(i) - temp
8    a(i) = a(i) + temp
C------------
C Special case for stage 2: no complex multiplies, simple add
C  (Performance enhancement)
   groups = 4
   offset = 2
   indexl = 1

C  i-loop iterates N/4 times for 2nd stage
C  1st call to i-loop,in stage2: indexl=1, wtemp(1)=(1,0)
CVD$        NODEPCHK
   DO 90 i = 1,n,4
     iplus = i + 2
     temp = a(iplus)
     a(iplus) = a(i) - temp
90   a(i) = a(i) + temp

   indexl = 2
CVD$       NODEPCHK
CVD$       NOVECTOR
   DO 92 i = 2,n,4
     iplus = i + 2
          temp = CMPLX(AIMAG(a(iplus)),-REAL(a(iplus)))
     a(iplus) = a(i) - temp
92   a(i) = a(i) + temp
CVD$       VECTOR
C------------
C "DO 20" stage-loop executed once for each of the (m) stages of FFT
C  (Except 1st and 2nd stage)
C  offset gets 4,8,16,32,64,128,256...
   DO 20 stage = 3,m
           groups = powers2(stage)
           offset = groups/2
           wtemp(1) =(1.0, 0.0)
C   One twiddle seed (W) calc per stage.
C   We pre-allocated w(12)-array with those values, avoid cos/sin calls
```

6

```
C-------------------
   DO 20 indexl = 1,offset

C "DO 10" i-loop does each butterfly of each stage, with varying twiddles
C    i-loop iterates N/2 times for 1st stage, N/4 x for 2nd, N/8 x for 3rd
C     stage, N/16 x for 4th stage,... 1 time for last stage.

CVD$       NODEPCHK
CVD$       ALTCODE
           DO 10 i = indexl,n,groups
             iplus = i + offset
             temp = a(iplus) * wtemp(1)
             a(iplus) = a(i) - temp
10           a(i) = a(i) + temp
20 wtemp(1) = wtemp(1) * w(stage)
   RETURN
   END
C------------
   subroutine cmags(a,n,asqr)
C Complex magnitude squared.
C Inputs:
C   A= complex array of input, single-prec float
C  N = number of input points (and output points)
C Ouput:
C   asqr = real squared magnitude (R*R + I*I), N elements, single-prec float

   integer n,i
   real asqr(n)
   complex a(n)

   DO 100 i = 1, n
     asqr(i) = (REAL(a(i))*REAL(a(i))) + (AIMAG(a(i))*AIMAG(a(i)))
100        CONTINUE
   RETURN
   END
```

```
## makefile for i860(tm) CPU FFTs (for Unix V/386 programming environment)
##   8/7/89
##
GH=/usr/i860/bin
GHL=/usr/i860/lib
CC=$(GH)/c860
FC=$(GH)/f860

CFLAGS= -OLM -X393 -X405 -X188 -X370

FFLAGS= -OLM -X370 -X393 -X71 -X422
## -X71 uses single-precision math routines

FLFLAGS= -Mx map -e start

LFLAGS= -Mx map -e _main
CLIB=$(GHL)/libc.a
MLIBPSR=$(GHL)/860mtlib.a

MLIB=$(GHL)/libm.a
FLIB=$(GHL)/libf.a

ASM=$(GH)/as860

FLINK=$(GH)/ld860 $(FLFLAGS)

RT=$(GHL)/s5lib.a

LIBS= $(FLIB) $(MLIBPSR) $(MLIB) $(CLIB) $(RT)

LIBCC= $(MLIB) $(CLIB) $(RT)
## NOTE: Order of linked files is CRUCIAL, other orders may give errors

.SUFFIXES:
.SUFFIXES: .f .c .s .ss .o .8

.IGNORE:
## .ignore causes make to ignore error codes from compilers

## To test Fortran plus assembler-fft-stage version:
FILE= ffttest.o fft.o diff.o bitrev.o difstep.o start.o time.o

## To test all-Fortran version of fft:
##FILE= ffttest.o fft.o diff.o difstepf.o start.o time.o

## To test REAL-input version of fft:
RFILE= real.o fft.o dirr.o realfix.o difstep.o bitrev.o start.o time.o

.f.o:
   $(FC) $(FFLAGS) $*.f
   $(ASM) -x -o $*.o $*.s

.c.o:
   $(CC) $(CFLAGS) $*.c
   $(ASM) -x -o $*.o $*.s
```

```
.s.o:
   m4 $*.s temp2.s
   $(ASM) -x -o $*.o temp2.s
ffttest.8: $(FILE)
   $(FLINK)  -o ffttest.8 $(FILE) $(LIBS)
real.8:    $(RFILE)
   $(FLINK)  -o real.8 $(RFILE) $(LIBS)

clean:
   rm -f *.o *.8

.ss.o:
   m4 $*.ss temp.s
   $(ASM) -x -o $*.o temp.s
```

```
//start.ss
// 8/18/89
// Fortran runtime startoff routine
//
   .text
   .globl  start
   .globl  finish
start::
   orh     h%_stack+262128+262144,r0,sp
   or     l%_stack+262128+262144,sp,sp
   adds    -16,sp,sp
   st.l    r1,12(sp)
   call    _main
   nop
finish::
   call    _exit
   nop
   .file   "start.c"

   .data
   .align  .quad
   .lcomm  _stack,262144+262144
   .end
//================================================================

/* file: time.c.  Purpose: establish a label to use for breakpoints */
long       time_(x)
long       *x;
{  x = x+4;
   return((long) x);
}
long       timestop_(x)
long       *x;
{  x = x+4;
   return((long) x);
}
```

i750™ Video Processor Family 7

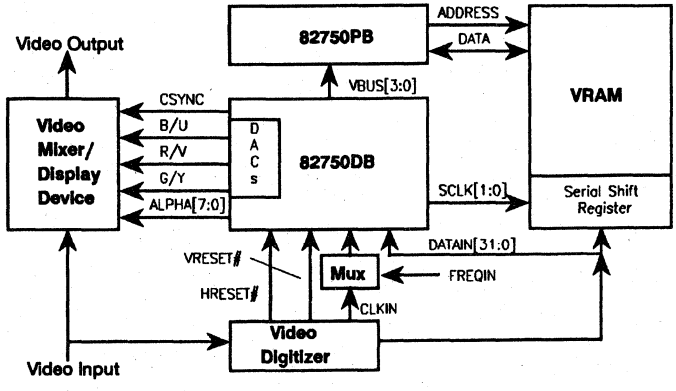
82750PB
PIXEL PROCESSOR

- 25 MHz Clock with Single Cycle Execution
- Zero Branch Delay
- Wide Instruction Word Processor
- 512 x 48-Bit Instruction RAM
- 512 x 16-Bit Data RAM
- Two Internal 16-Bit Buses
- ALU with Dual-Add-With-Saturation Mode
- Variable Length Sequence Decoder
- Pixel Interpolator
- High Performance Memory Interface
  - 32-Bit Memory Data Bus
  - 50 MBytes per Second Maximum
  - 25 MBytes per Second with Standard VRAMs or DRAMs
- 16 General-Purpose Registers
- 4 Gbyte Linear Address Space
- 132-Pin PQFP
- Compatible with the 82750PA

The 82750PB is a 25 MHz wide instruction processor that generates and manipulates pixels. When paired with its companion chip, the 82750DB, and used to implement a DVI Technology video subsystem, the 82750PB provides real time (30 images/sec) pixel processing, real time video compression, interactive motion video playback and real time video effects.

Real time pixel manipulations, including 30 images/sec video compression, are supported by the 25 MHz instruction rate. On-chip instruction RAM provides programmability for execution of a wide range of algorithms that support motion video decompression, text, and 2D and 3D graphics. Inner loops are optimized with the integration of sixteen 16-bit quad ported registers, on-chip DRAM, and two loop counters that provide zero delay two-way branching "free" in any instruction. Two, 16-bit internal buses enable two parallel register transfers on each 82750PB instruction, contributing to the real time performance of the video processing. Another feature that adds to the processing power of the 82750PB is the 16-bit ALU, which includes an 8-bit dual-add-with-saturate operation critical for pixel arithmetic. Other specialized features for pixel processing include a 2D pixel interpolator for image processing functions and a variable length sequence decoder for decoding compressed data.

The 82750PB is implemented using Intel's low-power CHMOS IV Technology and is packaged in a 132-lead space-saving, plastic quad flat pack (PQFP) package.



240854-1

82750PB Subsystem Diagram

For the complete data sheet on this device, contact Intel's Literature Distribution Dept., (800) 548-4725.

November 1990
Order Number: 240854-001

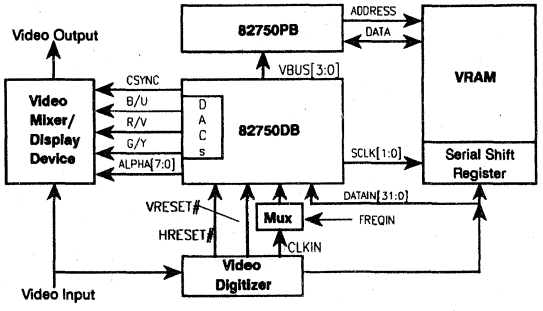
82750DB
DISPLAY PROCESSOR

- **Programmable Video Timing**
  - **— 28 MHz Operating Frequency**
  - **— Pixel/Line Address Range to 4096**
  - **— Fully Programmable Sync, Equalization, and Serration Components**
  - **— Fully Programmable Blanking and Active Display Start and Stop Times**
  - **— Genlocking Capability**
- **Flexible Display Characteristics**
  - **— 8-, Pseudo 16-, 16-, and 32-Bit/Pixel Modes**
  - **— Selectable Pixel Widths of 1.0, 1.5, 2.0, 2.5, through 14 Periods of the Input Frequency**
  - **— Support Popular Display Resolutions: VGA, NTSC, PAL, and SECAM**
  - **— On-Chip Triple DAC for Analog RGB/ YUV Output**
  - **— Mix Graphics and Video Images on a Pixel by Pixel Basis**
  - **— Real Time Expansion of the Reduced Sample Density Video Color Components (U, V) to Full Resolution**
  - **— Three Independently Addressable Color Palettes**
  - **— Programmable 2X Horizontal Interpolation of Y Channel**
  - **— 16 x 16 x 2-Bit Cursor Map with Independently Programmable 2X Expansion Factors in X and Y Dimensions**
  - **— YUV to RGB Color Space Conversion**
  - **— 2X Vertical Replication of Y, U, and V Data for Displaying Full Motion Video on VGA Monitor**
  - **— Register and Function Compatible with the 82750DA**

The 82750DB is a custom designed VLSI chip used for processing and displaying video graphic information. It is register and function compatible with the 82750DA.

Reset inputs allow the 82750DB to be genlocked to an external sync source. By programming internal control registers, this sync can be modified to accommodate a wide variety of scanning frequencies. A large selection of bits/pixel, pixels/line, and pixel widths are programmable, allowing a wide latitude in trading-off image quality vs update rate and VRAM requirements.

The 82750DB can operate in a digitizing mode, wherein it generates timing and control signals to the 82750PB and VRAM, but does not output display information. Besides digitizer support signals and video synchronization, the 82750DB outputs digital and analog RGB or YUV information and an 8-bit digital word of alpha data. This alpha channel data may be used to obtain a fractional mix of 82750DB outputs with another video source.



240855-1

82750DB Subsystem Diagram

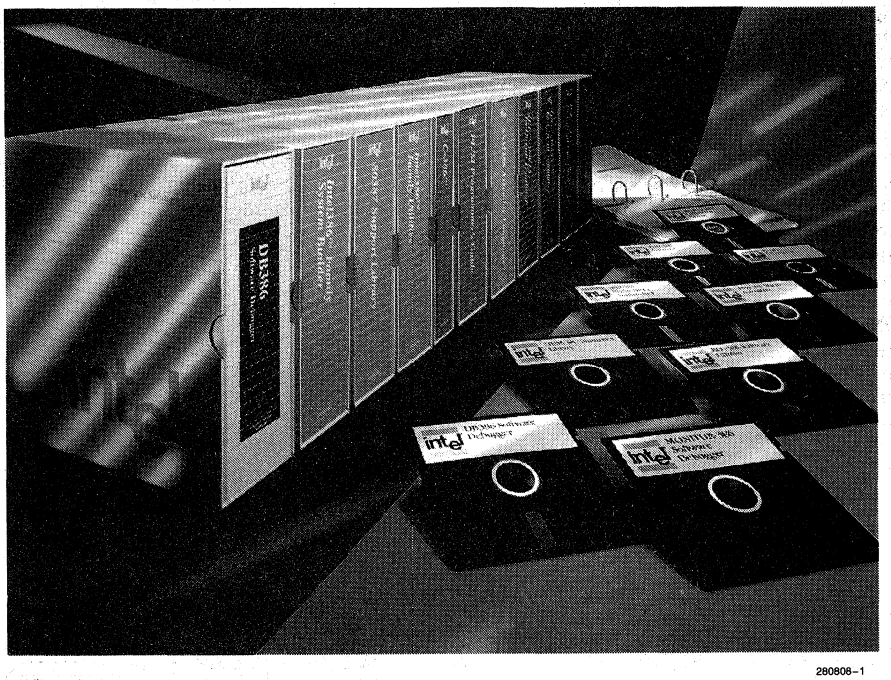
For the complete data sheet on this device, contact Intel's Literature Distribution Dept., (800) 548-4725.

November 1990
Order Number: 240855-001

Development Tools for the 80386 and 80486

8

INTEL386™/i486™ FAMILY DEVELOPMENT SUPPORT



280808-1

COMPREHENSIVE DEVELOPMENT SUPPORT FOR THE INTEL386™/i486™ FAMILIES OF MICROPROCESSORS

The perfect complement to the Intel386™ and i486™ microprocessor family is the optimum development solution. From a single source, Intel, comes a complete, synergistic hardware and software development toolset, delivering full access to the power of the Intel386 and i486 microprocessor family architectures.

Intel development tools are easy to use, yet powerful, with contemporary user interface techniques and productivity boosting features such as symbolic debugging. And you'll find Intel first to market with the tools needed to start development, and with lasting product quality and comprehensive support to keep development on-track.

If what interests you is getting the best product to market in as little time as possible, Intel is the choice.

8

*AboveBoard, i486, Intel386, 386 DX, 386 SX, 376, 387, ICE, and iPAT are trademarks of Intel Corporation.
VAX, MicroVAX and VMS are registered trademarks of Digital Equipment Corporation.

November 1990
Order Number: 280808-005

FEATURES

- Comprehensive support for the full 32 bit Intel 386 and i486 microprocessor architectures—includes protected mode, 4 gigabyte physical memory addressing, and i486 microprocessor on-chip cache and numerics
- Standard windowed interface that is common across Intel debug tools and architectures
- Source line display and symbolics allow debugging in the context of the original program
- Intel high-level languages provide architectural extensions for manipulating hardware directly without assembly language routines
- A common object code format (Intel OMF386™) supports symbolic debug and permits the intermixing of modules written in various languages—Intel's assembler, C, PL/M, and FORTRAN
- A common OMF386 permits compilers and assembler to seamlessly operate with in-circuit and software debug tools
- ROM-able code is output directly from the language tools, significantly reducing the effort necessary to integrate software into the final target system
- Extensive support for the Intel family of math coprocessors
- Operation in DOS IBM PC AT*, PS/2 Model 60 and 80, or compatible) and VAX/VMS* hosted environments

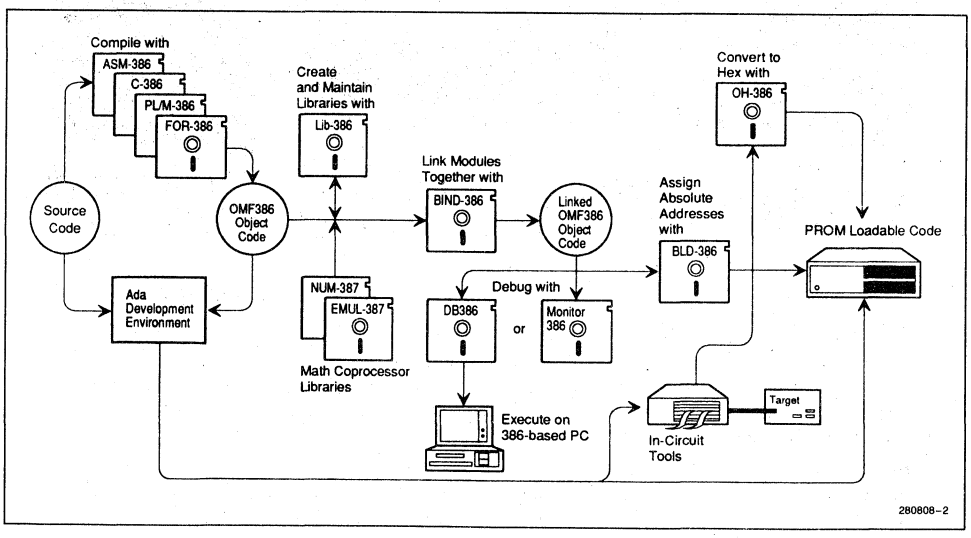


Figure 1: Intel Microprocessor Development Environment

intel

FEATURES

ASM 386™/i486™ MACRO ASSEMBLER

Intel's ASM 386™ is a "high-level" macro assembler for the Intel386 Family. ASM 386 offers many features normally found only in high-level languages. The macro facility in ASM 386 saves development time by allowing common program sequences to be coded only once. The assembly language is strongly typed, performing extensive checks on the usage of variables and labels.

Other Intel ASM 386 features include:

- "High-level" assembler mnemonics to simplify the language
- Structures and records for data representation
- Support for Intel's standard object code format for source-level symbolic debug, and for linking object modules from other Intel386 and i486 microprocessor languages
- Full support for processor and math coprocessor instruction sets
- A "MOD486" switch for support of the i486 microprocessor instructions
- 16 bit or 32 bit address overrides
- Supports development for Virtual 86, Real, 286 Protected, and 386 Protected modes

iC386™/i486™ COMPILER

Intel's iC-386 compiler provides special features for architectural support and code efficiency, for ease of use, and for compatibility with other Intel development tools.

The iC-386 compiler produces code for Intel386 and i486™ processors from C source files, and conforms to the 1989 ANSI standard (ANS X3.159-1989) for the C programming language.

Key Intel iC-386 features include:

- Controls to tailor the compilation for each step of your application development process
- In-line versions of many ANSI-standard library functions
- Uses expanded memory (LIM Version 3.0 and higher)
- Object code (including supplied run-time libraries) suitable for ROM
- Three different levels of optimization
- A choice of three segmentation memory models (small, compact, and flat) to create compact and efficient code
- Object code that takes advantage of the on-chip cache of the i486 processor
- In-line processor-specific functions and time-saving macros that provide access to the special features of the Intel386 and i486 processors
- In-line floating-point instructions for the 387™ numerics coprocessor and i486 processor floating-point unit
- Time-saving macros and functions to help assembly language routines interface with Intel's high-level programming languages
- The standard C run-time library plus libraries for floating-point support and the iRMX® III C interface library
- An easy interface to Intel's non-C programming languages, along with object module compatibility between Intel C and non-C compilers
- Support for source-level debugging using the Intel DB-386 Software Debugger
- Programming with subsystems, allowing mixed segmentation memory models
- Extensions to the 1989 ANSI C standard for compatibility with previous versions of Intel C

The iC-386 libraries contain over 200 functions for use in iC-386 programs. The libraries and header files make development of iC-386 applications easier by providing:

- Fast and efficient functions for common programming tasks
- Interfaces to standard and custom execution environments
- Built-in versions of some functions

PL/M386™/i486™ COMPILER

Intel's PL/M-386 is a structured high-level system implementation language for the Intel386/i486 Families. PL/M-386 supports the implementation of protected operating system software by providing built-in procedures and variables to access the Intel386/i486 architectures.

For efficient code generation, PL/M-386 features four levels of optimization, a virtual symbol table, and four models of program size and memory usage.

8

FEATURES

Other Intel PL/M-386 features include:

- The ability to define a procedure as an interrupt handler as well as facilities for generating interrupts
- Direct support of byte, half-word, and word input and output from microprocessor ports
- Upward compatibility with Intel PL/M-286 and PL/M-86 source code
- A "MOD486" compiler switch for i486 microprocessor instruction generation

PL/M-386 combines the benefits of a high-level language with the ability to access the Intel386 architecture. For the development of systems software, PL/M-386 is a costeffective alternative to assembly language programming.

FORTRAN 386™/i486™ COMPILER

Intel's FORTRAN-386 compiler is a cross-compiler that supports the entire Intel386 family of components and i486 (when operating in the 386 chip mode) microprocessors.

FORTRAN-386 features high-level support for floating-point calculations, transcendentals, interrupt procedures, and run-time exception handling. Specifically, the FORTRAN-386 language is a superset of the language described in the ANSI Fortran 77 standard. The additions to that standard include the Department of Defense (DOD) extensions, extensions that support programs written for the ANSI Fortran 66 standard, and extensions that support the 386 microprocessor and 80387/80387DX/80387SX math coprocessors.

To aid in the development and debugging process, the compiler generates warning and error messages and an optional listing file. The listing file can include symbol cross-reference tables and a listing of the generated 386 microprocessor assembly-language instructions. Library routines are reentrant and ROMable.

Other Intel FORTRAN-386 compiler features include:

- Object code can be configured to reside in either RAM or ROM
- The program code can be optimized for execution speed or memory size
- Source-level debugging is supported via the rich symbolics provided in the object module format (Intel OMF386)
- Support for the proposed REALMATH IEEE floating point standard

RLL 386™/i486™ RELOCATION, LINKAGE, AND LIBRARY TOOLS

The RLL 386™ relocation, linkage, and library tools are a cohesive set of utilities featuring comprehensive support of the full Intel386™/i486™ architectures. RLL-386 provides for a variety of functions—from linking separate modules, building an object library, or linking in 387™ support, to building a task to execute under protected mode or the multi-tasking, memory protected system software itself. Specifically, RLL-386 supports loadable, linkable, and bootloadable Intel object module formats; and supports all segmentation models, including FLAT. Map, librarian, and conversion (for outputting hex format code for PROM programming) utilities are included.

EMUL387, NUM387 NUMERICS SUPPORT LIBRARIES

Intel's EMUL-387 and NUM-387 Numerics Libraries fully support the 80387/80387DX/80387SX math coprocessors and the i486 internal math coprocessor—whether an actual math coprocessor is used in the final system or not.

For 386 microprocessor based applications without a math coprocessor, EMUL-387, a numerics software emulator, will execute instructions as though the coprocessor were present. Its functionality is identical to that of the math coprocessor. It is ideal for prototyping and debugging floating-point application software independent of hardware. Further, this permits portability of application code regardless of the presence of math coprocessor hardware in target systems.

For applications with a math coprocessor, NUM-387 numerics support library provides Intel's ASM 386, C-386, PL/M-386, and FORTRAN-386 language users with enhanced numeric data processing capability. With the library, it is easy for programs to do floating point arithmetic. Programmers can bind in library modules to do trigonometric, logarithmic and other numeric functions, and the user is guaranteed accurate, reliable results for all appropriate inputs.

FEATURES

Intel's NUM-387 support library is a collection of four functionally distinct libraries:

- Common elementary function library routines perform algebraic, logarithmic, exponential, trigonometric, and hyperbolic operations on real and complex numbers, as well as real-to-integer conversions; the routines extend the ranges of the coprocessor instructions
- Initialization library routines set up the numerics processing environment for 80386 microprocessor based systems with an 80387/80387DX/80387SX or true software emulator
- Decimal conversion library routines convert floating-point numbers from one 80387/80387DX/80387SX binary storage format to another, or from ASCII decimal strings to 80387/80387DX/80387SX binary floating-point format and vice versa
- Exception handling library routines make writing numerics exception handlers easier

All support library modules are in 80386 microprocessor object module format (Intel OMF-386) so they can be linked with the object output of any Intel language. All routines are reentrant and ROMable.

By using Intel's NUM-387, the user not only saves software development time, but is guaranteed that the numeric software meets industry standard (ANSI/IEEE standard for binary floating point arithmetic, 754-1985) and is portable—software investment is maintained.

FEATURES

ONCE-386

If you have a surface mount Intel386 SX microprocessor design using 100 pin PQFP parts, Intel ICE emulators now have "On-Circuit Emulation" (ONCE™) capability. With your part surface mounted, the ICE-386 SX emulator cabling clamps over the part, tri-stating the component, and allowing the emulator to operate. This allows you to debug manufactured boards without resoldering.

REM-386

Designed to enhance your existing ICE-386 DX 25 and ICE-386 DX 33 emulators, the REM-386 DX Expansion board adds 2 MB of expanded memory.

INTEL386™/i486™ FAMILY IN-CIRCUIT TOOLS

In-Circuit Emulators

Intel386 Family in-circuit emulators embody exclusive technology that gives access to internal processor states that are accessible in no other way. Intel386 microprocessors fetch and execute instructions in parallel, with fetched instructions not necessarily executing in order of input. Because of this, an emulator without this access to internal processor states is prone to error in determining what actually occurred inside the microprocessor. With Intel's exclusive technology, Intel386 Family emulators are one hundred percent accurate. In addition, internal access comes without signal buffer interference of processor timing. Operation is non-intrusive (zero wait-state).

Other features of Intel386 Family in-circuit emulators include:

- Unparalleled support of the Intel386 architecture, notably the native protected mode
- Emulation at clock speeds to 33 MHz, and full featured trigger and trace capabilities
- Convertible using removable probes to support any of the Intel386 microprocessors—80386DX, 80386SX, and 80376 microprocessors

With symbolic debugging, memory locations can be examined or modified using symbolic references to the original program, such as procedure or a variable names, line numbers, or program labels. Source code associated with a given line number can be displayed, as can the type information of variables, such as byte, word, record, or array. Microprocessor data structures, such as registers, descriptor tables, and page tables, can also be examined and modified using symbolic names. The symbolic debugging information for use with Intel development tools is produced by Intel OMF386 compatible languages.

ICE™-486 IN-CIRCUIT EMULATOR

The ICE-486 In-circuit Emulator is the world's leading tool for debugging software and hardware designs based on the Intel i486 family of microprocessors. The ICE-486 emulator features real-time emulation at speeds up to 33 MHz. The standard high- level symbolic debug capability saves valuable development time. The flexible breakpoint capability and 8K deep trace buffer provide power to identify and solve even the toughest hardware and software bugs. The emulator also provides 2 MB expansion memory to debug large programs. It is designed to work with the rich array of software development tools optimized for creating 32-bit applications.

In-Circuit Debugger

Intel's ICD-486 represents a new generation of in-circuit emulation technology. From the inventor of the microprocessor comes a development tool that delivers complete access to the i486 architecture. ICD-486 is the first development tool which allows users to debug high speed, cached applications at the full speed of the target processor. ICD-486 embodies exclusive technology, giving users symbolic access to the internal processor states that would not be accessible in any other way. With Intel;s exclusive technology, users can be assured that the ICD-486 provides complete accuracy when debugging cached applications in real-time.

Other Intel ICD-486 features include:

- Real-time emulation at the full speed of the i486 microprocessor
- Full support for the i486 on-chip caching and numerics
- Ability to set up to 16 software breakpoints and four hardware breakpoints on execution addresses, data writes, or data accesses
- Full symbolic information to display and modify all registers of the i486 microprocessor

FEATURES

SOFTWARE DEBUGGER

Intel's DB386™ is an on-host software execution environment with source-level symbolic debug capabilities for object modules produced by Intel's assembler and high-level language compilers. For the DOS hosted version, this software debug environment allows 386 microprocessor code to be executed and debugged directly on a 386 DX or 386 SX microprocessor based PC, without any additional target hardware required. With Intel's standard windowed human interface, users can focus their efforts on finding bugs rather than spending time learning and manipulating the debug environment.

For the VMS* hosted version, the debugger works in conjunction with an extensive 386 microprocessor software instruction simulator included with the product. This simulator simulates the 386 microprocessor in "flat" mode, 387, 8259A and 8254 interrupt controller and timer chips, supports map memory up to 4 gigabytes, and provides complex break, trace, and profiling support.

FEATURES

Other Intel DB386™ features include:

- A run-time interface allows protected-mode 386 microprocessor programs to be executed directly on a 386 DX or 386 SX microprocessor based PC
- Drop-down menus make the tool easy to learn for new or casual users. A command line interface is also provided for more complex problems
- Watch windows (which display user-specified variables), trace points, and breakpoints (including fixed, temporary, and conditional) can be set and modified as needed, even during a debug session
- The user can browse source and callstacks, observe processor registers, and access watch window variables by either the pull down menu or by a single keystroke using the function keys
- An easy-to-use disassembler and single-line assembler speeds the debug process
- The user need not know whether a variable is an unsigned integer, a real, or a structure—the debugger uses the wealth of typing information available in Intel languages to display program variables in their respective type formats
- DB-386 supports the i486 microprocessor when operated in the 386 microprocessor mode

MON386 TARGET RESIDENT SOFTWARE DEBUGGER

Intel's MON-386 is a hosted or unhosted target resident software debugger for the 386 DX and 386 SX-based systems. MON-386 provides program execution control and symbolic processor and memory interrogation and modification. Hardware and software breakpoints can be set at symbolic addresses and program execution can be single-stepped through assembly level or high-level language instructions.

Other Intel MON-386 features include:

- Debug procedures (user-definable sequences of MON-386 commands) enable users to define macro commands that would otherwise take several lines of command entries to perform the same function
- A disassembler/single line assembler allows users to display memory and patch memory with 80386/80387 mnemonics

MON-386, used in conjunction with Intel single board computers iSBC® 386/22 and iSBC 386/116, or other customer designed systems, can debug software before a functional prototype of the target system is available.

Intel's MON-386 can be used for i486 microprocessor development when the component is run in the 386 microprocessor mode of operation.

iPAT-386™ PERFORMANCE ANALYSIS TOOL

Intel's iPAT-386™ performance analysis tool provides analysis of real-time software executing on a 386-based target system. With iPAT-386, it is possible to speed-tune applications, optimize use of operating systems, determine response characteristics, and identify code execution coverage.

By examining iPAT-386 histogram and tabular information about procedure usage for critical functions (with the option of including interaction with other procedures, hardware, the operating system, or interrupt service routines) performance bottlenecks can be identified. With iPAT-386 code execution coverage information, the completeness of testing can be confirmed.

Intel's iPAT-386 provides real-time analysis up to 20 MHz, performance profiles of up to 125 partitions, and code execution coverage analysis over 252K. The iPAT-386 target probe is used with the same iPAT base module supporting 80286, 80186, and 8086 development. The iPAT-386 system can be used independently or piggy-backed with Intel386 in-circuit emulator tools.

WORLDWIDE SERVICE, SUPPORT, AND TRAINING

To augment its development tools, Intel offers a full array of seminars, classes, and workshops, field application engineering expertise, hotline technical support and on-site service.

Intel also offers a Software Support package which includes technical software information, telephone support, automatic distribution of software and documentation updates, access to the "ToolTalk" electronic bulletin board, "*iComments*" publication, remote diagnostic software, and a development tools troubleshooting guide.

FEATURES

PRODUCT SUPPORT MATRIX

Product	Component				Host	
	i486™	386™ DX	386™ SX	376™	DOS 3.x	VMS 5.1+
ASM-386 Macro Assembler	✓	✓	✓	✓	✓	✓
iC-386 Compiler	✓	✓	✓	✓	✓	✓
PL/M-386 Compiler	✓	✓	✓	✓	✓	✓
FORTRAN-386 Compiler	✓	✓	✓	✓	✓	✓
RLL-386 Relocation, Linkage, Library, Support Tools	✓	✓	✓	✓	✓	✓
NUM-387 Libraries	✓	✓	✓	✓	✓	✓
EMUL-387 Libraries	NA	✓	✓	✓	✓	✓
In-circuit Emulators		✓	✓	✓	✓	
In-circuit Debugger	✓				✓	
DB-386 Software Debugger	✓	✓	✓	✓	✓	
MON-386 Target Level Software Debugger		✓	✓	✓	✓	
iPAT-386 Performance Analysis Tool		✓			✓	

ORDERING INFORMATION

386™/i486™ FAMILY DOS HOSTED DEVELOPMENT KIT ORDER CODES

DKIT386C	Compiler Software Development Kit (see following content list). Also supports i486 microprocessor
DKIT386CS	C Compiler Software Development Kit w/ one year Gold Software Support. Also supports i486 microprocessor.
DKIT386CIDX	C Compiler Software Development Kit w/ ICE386 DX 33 MHz In-circuit Emulator and 2 MB AboveBoard™
DKIT386CIDXS	Same as above w/ one year Hardware and Gold Software Support
pDKIT386CISX	C Compiler Software Development Kit w/ ICE386 SX 20 MHz In-circuit Emulator and 2 MB Above Board
pDKIT386CISXS	Same as above w/ one year Hardware and Gold Software Support
pDKIT386CI376	C Compiler Software Development Kit w/ ICE376 16 MHz In-circuit Emulator and 2 MB Above Board
pDKIT386CI376S	Same as above w/ one year Hardware and Gold Software Support

The Intel Basic Software Development Kit for the DOS hosted environment includes:

iC386 compiler
ASM386 assembler
RLL386 relocation linker and locator (builder/binder)
NUM387 numerics library
EMUL387 math coprocessor emulator library
DB386 software debugger
OMF386LOAD loader development object module format documentation

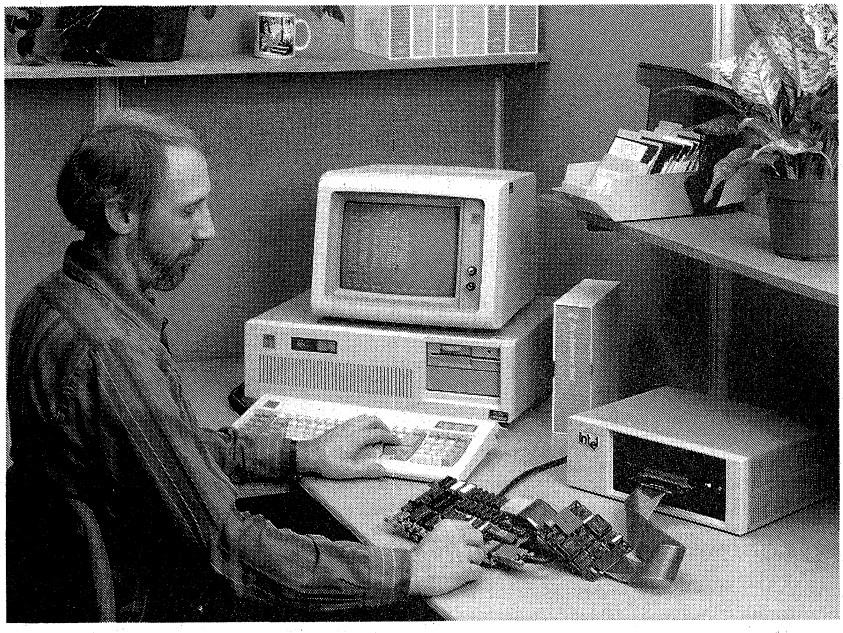
386™/i486™ FAMILY VAX AND MICROVAX/VMS HOSTED DEVELOPMENT KIT ORDER CODES*

MVVSC386KIT	MicroVAX/VMS C386 compiler, RLL386 relocation linker and locator, ASM386 assembler, DB386 software debugger
MVVSP386KIT	MicroVAX/VMS PL/M386 compiler, RLL386, ASM386, DB386
MVVSF386KIT	MicroVAX/VMS FORTRAN386 compiler, RLL386, ASM386, DB386
VVSC386KIT	VAX/VMS C386, RLL386, ASM386, DB386
VVSP386KIT	VAX/VMS PL/M386, RLL386, ASM386, DB386
VVSF386KIT	VAX/VMS FORTRAN386, RLL386, ASM386, DB386

ADDITIONAL 386™/i486™ FAMILY DEVELOPMENT TOOL ORDER CODES

ICD48625D	25 MHz In-circuit Debugger for the i486 microprocessor
ICD486CON33D	ICD48625D with a prepaid upgrade to 33 MHz
iPATCORE	iPAT Performance Analysis Tool base unit
iPAT386DOS	iPAT 80386 probe kit including PC-DOS 3.x software, requires iPATCORE

INTEL 376™ FAMILY DEVELOPMENT SUPPORT



280903-1

COMPREHENSIVE DEVELOPMENT SUPPORT FOR THE INTEL 376™ EMBEDDED PROCESSORS

The perfect complement to the Intel 376™ embedded processor is the optimum development solution. From a single source, Intel, comes a complete, synergistic hardware and software development toolset, delivering full access to the power of the Intel 376 architecture.

Intel development tools are easy to use, yet powerful, with ease-of-use user interface techniques and productivity boosting features such as symbolic debugging. And you'll find Intel first to market with the tools needed to start development, and with lasting product quality and comprehensive support to keep development on-track.

If what interests you is getting the best product to market in as little time as possible, Intel is the choice.

FEATURES

- Full speed emulation up to 20 MHz
- Source line display and symbolics allow debugging in the context of the original program
- Intel high-level languages provide architectural extensions for manipulating hardware directly without assembly language routines
- A common object code format (Intel OMF386) supports symbolic debug and permits the intermixing of modules written in various languages—Intel's assembler, C, PL/M, and FORTRAN
- A common OMF386 permits compilers and assembler to seamlessly operate with in-circuit and software debug tools
- ROM-able code is output directly from the language tools, significantly reducing the effort necessary to integrate software into the final target system
- Extensive support for the Intel 80387SX math coprocessor
- Operation in DOS IBM PC AT*, PS/2 Model 60 and 80, or compatible) and VAX/VMS* hosted environments

8

November 1990
Order Number: 280903-001

INTEL 376™ FAMILY DEVELOPMENT SUPPORT

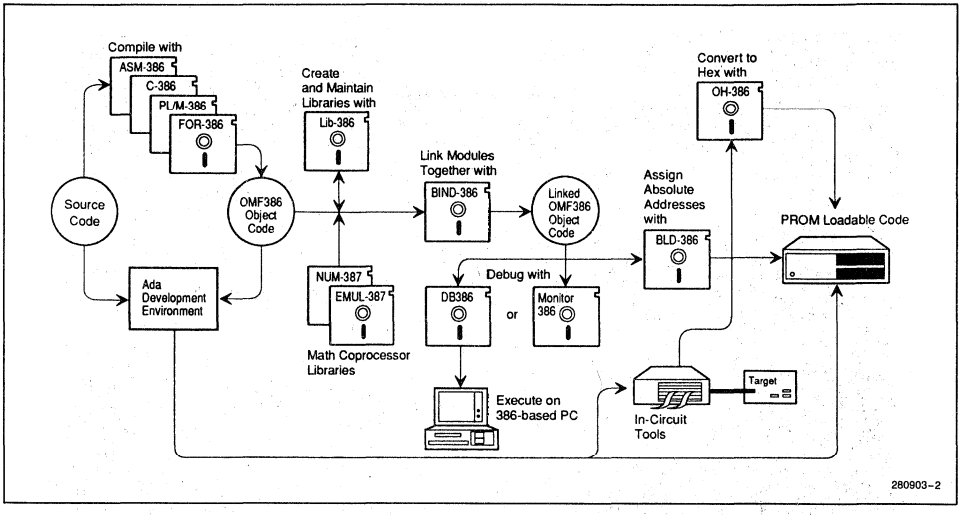


Figure 1: Intel Microprocessor Development Environment

ASM 386™ MACRO ASSEMBLER

Intel's ASM 386™ is a "high-level" macro assembler for developing 376 based embedded system. ASM 386 offers many features normally found only in high-level languages. The macro facility in ASM 386 saves development time by allowing common program sequences to be coded only once. The assembly language is strongly typed, performing extensive checks on the usage of variables and labels.

Other Intel ASM 386 features include:

- "High-level" assembler mnemonics to simplify the language
- Structures and records for data representation
- Support for Intel's standard object code format for source-level symbolic debug, and for linking object modules from other Intel386/376 languages
- Full support for processor and math coprocessor instruction sets
- 16 bit or 32 bit address overrides
- Supports development for Virtual 86, Real, 286 Protected, and 386 Protected modes

iC-386 COMPILER

Intel's iC-386 compiler provides special features for Intel 376 architectural support and code efficiency, for ease of use, and for compatibility with other Intel development tools.

The iC-386 compiler produces code for Intel 376 processor from C source files, and conforms to the ANSI standard (ANS X3.159-1989) for the C programming language.

Key Intel iC-386 features include:

- Controls to tailor the compilation for each step of your application development process
- In-line versions of many ANSI-standard library functions
- Uses expanded memory (LIM Version 3.0 and higher)
- Object code (including supplied run-time libraries) suitable for ROM
- Three different levels of optimization
- A choice of three segmentation memory models (small, compact, and flat) to create compact and efficient code
- In-line processor-specific functions and time-saving macros that provide access to the special features of the Intel 376 embedded processor

INTEL 376™ FAMILY DEVELOPMENT SUPPORT

- In-line floating-point instructions for the 387™ SX™ numerics coprocessor
- Time-saving macros and functions to help assembly language routines interface with Intel's high-level programming languages
- The standard C run-time library plus libraries for floating-point support
- An easy interface to Intel's non-C programming languages, along with object module compatibility between Intel C and non-C compilers
- Support for source-level debugging using the Intel DB-386 Software Debugger
- Programming with subsystems, allowing mixed segmentation memory models
- Extensions to the ANSI C standard for compatibility with previous versions of Intel C

The iC-386 libraries contain over 200 functions. The libraries and header files make development of iC-386 applications easier by providing:

- Fast and efficient functions for common programming tasks
- Interfaces to standard and custom execution environments
- Built-in versions of some functions

PL/M-386 COMPILER

Intel's PL/M-386 is a structured high-level system implementation language for the Intel 376 embedded processor. PL/M-386 supports the implementation of protected operating system software by providing built-in procedures and variables to access the Intel 376 architecture.

For efficient code generation, PL/M-386 features four levels of optimization, a virtual symbol table, and four models of program size and memory usage.

Other Intel PL/M-386 features include:

- The ability to define a procedure as an interrupt handler as well as facilities for generating interrupts
- Direct support of byte, half-word, and word input and output from microprocessor ports
- Upward compatibility with Intel PL/M-286 and PL/M-86 source code

PL/M-386 combines the benefits of a high-level language with the ability to access the Intel386 architecture. For the development of systems software, PL/M-386 is a cost-effective alternative to assembly language programming.

FORTRAN-386 COMPILER

Intel's FORTRAN-386 compiler is a cross-compiler that supports the entire Intel 376 embedded processor.

FORTRAN-386 features high-level support for floating-point calculations, transcendentals, interrupt procedures, and run-time exception handling. Specifically, the FORTRAN-386 language is a superset of the language described in the ANSI Fortran 77 standard. The additions to that standard include the Department of Defense (DOD) extensions, extensions that support programs written for the ANSI Fortran 66 standard, and extensions that support the 376 processor and 80387SX math coprocessors.

To aid in the development and debugging process, the compiler generates warning and error messages and an optional listing file. The listing file can include symbol cross-reference tables and a listing of the generated 386 microprocessor assembly-language instructions. Library routines are reentrant and ROMable.

Other Intel FORTRAN-386 compiler features include:

- Object code can be configured to reside in either RAM or ROM
- The program code can be optimized for execution speed or memory size
- Source-level debugging is supported via the rich symbolics provided in the object module format (Intel OMF386)
- Support for the proposed REALMATH IEEE floating point standard

RLL-386 RELOCATION, LINKAGE, AND LIBRARY TOOLS

The RLL-386 relocation, linkage, and library tools are a cohesive set of utilities featuring comprehensive support of the full Intel 376 architecture. RLL-386 provides for a variety of

functions—from linking separate modules, building an object library, or linking in 387 SX support, to building a task to execute under protected mode or the multi-tasking, memory protected system software itself. Specifically, RLL-386 supports loadable, linkable, and bootloadable Intel object module formats; and supports all segmentation models, including FLAT. Map, librarian, and conversion (for outputting hex format code for PROM programming) utilities are included.

EMUL-387, NUM-387 NUMERICS SUPPORT LIBRARIES

Intel's EMUL-387 and NUM-387 Numerics Libraries fully support the 80387SX math coprocessor whether an actual math coprocessor is used in the final system or not.

For 376 microprocessor based applications without a math coprocessor, EMUL-387, a numerics software emulator, will execute instructions as though the coprocessor were present. Its functionality is identical to that of the math coprocessor. It is ideal for prototyping and debugging floating-point application software independent of hardware. Further, this permits portability of application code regardless of the presence of math coprocessor hardware in target systems.

For applications with a math coprocessor, NUM-387 numerics support library provides Intel's ASM 386, C-386, PL/M-386, and FORTRAN-386 language users with enhanced numeric data processing capability. With the library, it is easy for programs to do floating point arithmetic. Programmers can bind in library modules to do trigonometric, logarithmic and other numeric functions, and the user is guaranteed accurate, reliable results for all appropriate inputs.

Intel's NUM-387 support library is a collection of four functionally distinct libraries:

- Common elementary function library routines perform algebraic, logarithmic, exponential, trigonometric, and hyperbolic operations on real and complex numbers, as well as real-to-integer conversions; the routines extend the ranges of the coprocessor instructions
- Initialization library routines set up the numerics processing environment for 80376 microprocessor based systems with an 80387SX or true software emulator
- Decimal conversion library routines convert floating-point numbers from 80387SX binary storage format to ASCII decimal strings to 80387SX binary and vice versa
- Exception handling library routines make writing numerics exception handlers easier

All support library modules are in 80386 microprocessor object module format (Intel OMF386) so they can be linked with the object output of any Intel 80386 language. All routines are reentrant and ROMable.

By using Intel's NUM-387, the user not only saves software development time, but is guaranteed that the numeric software meets industry standard (ANSI/IEEE standard for binary floating point arithmetic, 754-1985) and is portable—software investment is maintained.

INTEL 376™ FAMILY DEVELOPMENT SUPPORT

INTEL 376™ IN-CIRCUIT EMULATORS

Intel 376 In-circuit Emulators embody exclusive technology that gives access to internal processor states that are accessible in no other way. Intel 376 processor fetch and execute instructions in parallel, with fetched instructions not necessarily executing in order of input. Because of this, an emulator without this access to internal processor states is prone to error in determining what actually occurred inside the microprocessor. With Intel's exclusive technology, Intel 376 emulator is one hundred percent accurate. In addition, internal access comes without signal buffer interference of processor timing. Operation is non-intrusive (zero wait-state).

Opening the Door to Protected Mode

The Intel 376 In-circuit Emulator opens the door to the full potential of the architecture with unparalleled support of protected mode. The emulator can display and modify task state segments and global, local, and interrupt descriptor tables (with symbolic access to all descriptor components like privilege level and segment type). Emulation memory of 128 Kbytes or the optional 2 Mbytes of relocatable expansion memory can be used instead of target memory for code debugging.

With symbolic debugging, memory locations can be examined or modified using symbolic references to the original program, such as procedure or a variable names, line numbers, or program labels. Source code associated with a given line number can be displayed, as can the type information of variables, such as byte, word, record, or array. Processor data structures, such as registers, descriptor tables, and page tables, can also be examined and modified using symbolic names. The symbolic debugging information for use with Intel development tools is produced by Intel OMF386 compatible languages.

Flexible and Versatile Event Recognition

Flexibility and versatility in event recognition makes short work of uncovering the most complex bugs. Bus even recognition circuitry may be used to trigger on specific or masked data input, output, read, written, or fetched at a physical address or range of addresses. Or on-chip debug registers may be used to trigger on virtual, linear, or symbolic addresses being executed, accessed, or written.

Versatility shows in other triggering options—upon a task switch, an external signal from another emulator or a logic analyzer, multiple occurrences of an event, a full trace buffer, halt or shutdown cycles, or interrupt acknowledge. And up to four sequential event triggers can be combined with a high-level construct.

The Intel 376 In-circuit Emulator continuously captures all bus activity and, as an option, execution information, into a trace buffer of 4K frames with PRE, POST, and CENTERED collection modes. The contents of the trace buffer can be displayed during full speed emulation in either execution cycle or machine-level instruction formats.

Accessing the Power

The power of the Intel 376 In-circuit Emulator is reflected in the sophisticated user interface. Refined for ease-of-use, the command line interface contains many features to boost productivity and customize functionality.

On-line help, a syntax menu, command line editing, command history, and error message query promote ease of learning and use. I/O redirection and the ability to escape the host operating system provide versatility for the power user. Customized procedures with variables and literal definitions can be created to assist in debugging or for manufacturing test or field service applications.

SOFTWARE DEBUGGER

Intel's DB386 is a useful tool for early software algorithm debug. It is an on-host software execution environment with source-level symbolic debug capabilities for object modules produced by Intel's assembler and high-level language compilers. For the DOS hosted version, this software debug environment allows 386 microprocessor code to be executed and debugged directly on a 386 DX or 386 SX™ microprocessor based PC, without any additional target hardware required. With Intel's standard windowed human interface, users can focus their efforts on finding bugs rather than spending time learning and manipulating the debug environment.

INTEL 376™ FAMILY DEVELOPMENT SUPPORT

Other Intel DB386 features include:

- A run-time interface allows protected-mode 376 microprocessor programs to be executed directly on a 386 DX or 386 SX microprocessor based PC
- Drop-down menus make the tool easy to learn for new or casual users. A command line interface is also provided for more complex problems
- Watch windows (which display user-specified variables), trace points, and breakpoints (including fixed, temporary, and conditional) can be set and modified as needed, even during a debug session
- The user can browse source and callstacks, observe processor registers, and access watch window variables by either the pull down menu or by a single keystroke using the function keys
- An easy-to-use disassembler and single-line assembler speeds the debug process
- The user need not know whether a variable is an unsigned integer, a real, or a structure—the debugger uses the wealth of typing information available in Intel languages to display program variables in their respective type formats

ICE™-376 SPECIFICATIONS AND REQUIREMENTS

HOST SYSTEM REQUIREMENTS

The user supplied host system can be either an IBM® PC AT® or Personal System/2® Model 60. Host system requirements to run the emulator include the following:

- DOS version 4.0, or Hewlett-Packard HP9000 UNIX
- 640 Kbytes of RAM in conventional memory
- An Above™ board with 1 megabyte of RAM configured in expanded memory mode, EMM.SYS software version 3.2
- A 20 MB hard disk
- A serial port or the National Instruments GPIB-PCII™, GPIB-PCIIA™, or MC-GPIB™ board
- A math coprocessor if either the optional time tag board is used or if a math coprocessor resides on the target system

ELECTRICAL CHARACTERISTICS

100-120V or 220-240V selectable
50-60 Hz
2 amps (AC max) @ 120V
1 amp (AC max) @ 240V

ENVIRONMENTAL CHARACTERISTICS

Operating Temperature: +10°C to +40°C (50°F to 104°F)

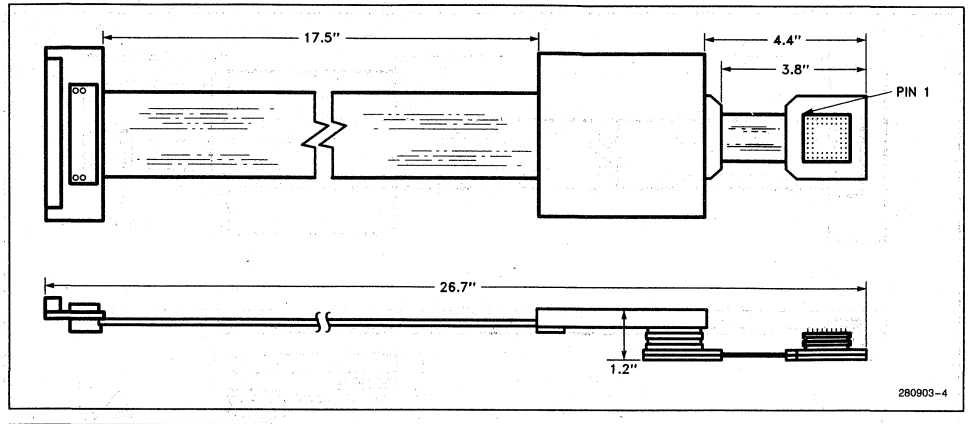
Operating Humidity: Maximum of 85% relative humidity, non-condensing

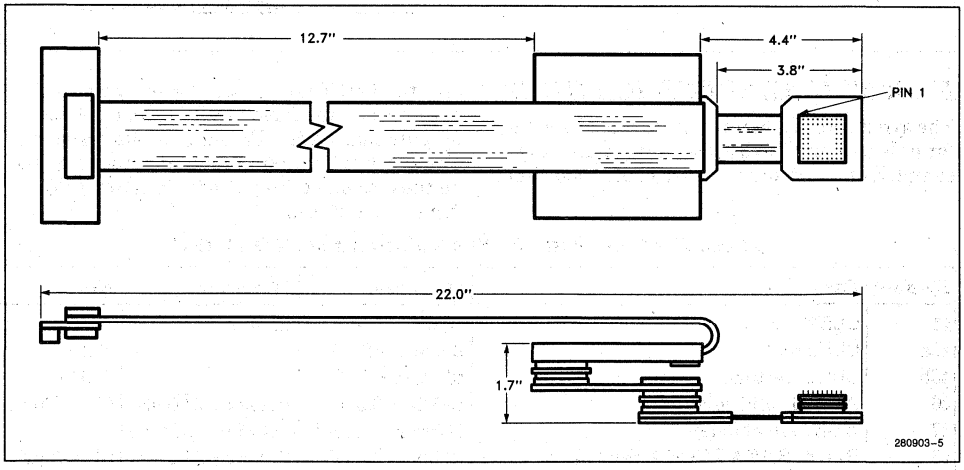
The Emulator's Physical Characteristics

Unit	Width		Height		Length	
	inches	cm	inches	cm	inches	cm
Base Unit	13.4	34.0	4.6	11.7	11.0	27.9
Processor Module	3.8	9.7	0.7	1.8	4.4	11.2
Optional Isolation Board	3.8	9.7	0.5	1.3	4.4	11.2
Power Supply	7.7	19.6	4.1	10.4	11.0	27.9
User Cable	1.9	4.8			17.3	43.9
100-Pin Target-Adapter Cable	2.3	5.3	0.5	1.3	5.1	13.0
88-Pin Target-Adapter Cable	2.3	5.3	0.5	1.3	5.8	14.7
Serial Cable					144	366
Optional Clips Pod	3.3	8.4	0.8	2.0	6.0	15.2

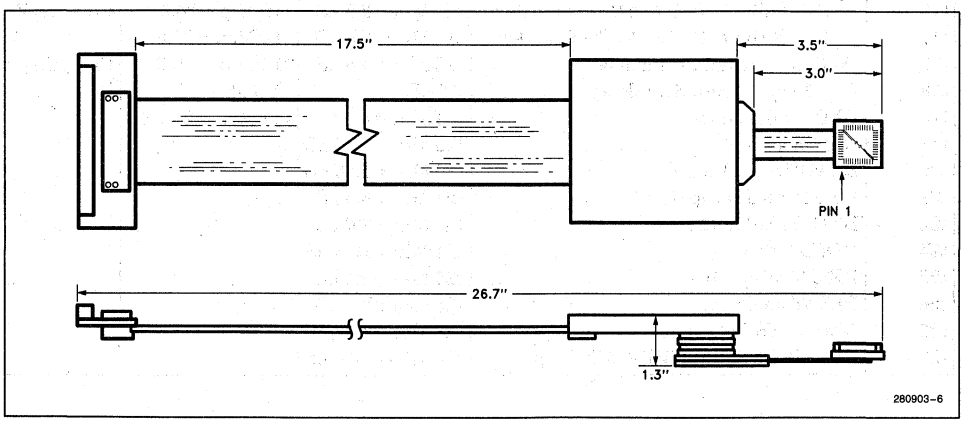
ICE™-376 SPECIFICATIONS AND REQUIREMENTS

The Processor Module and Bus Isolation Board Dimensions (88 Pin PGA)





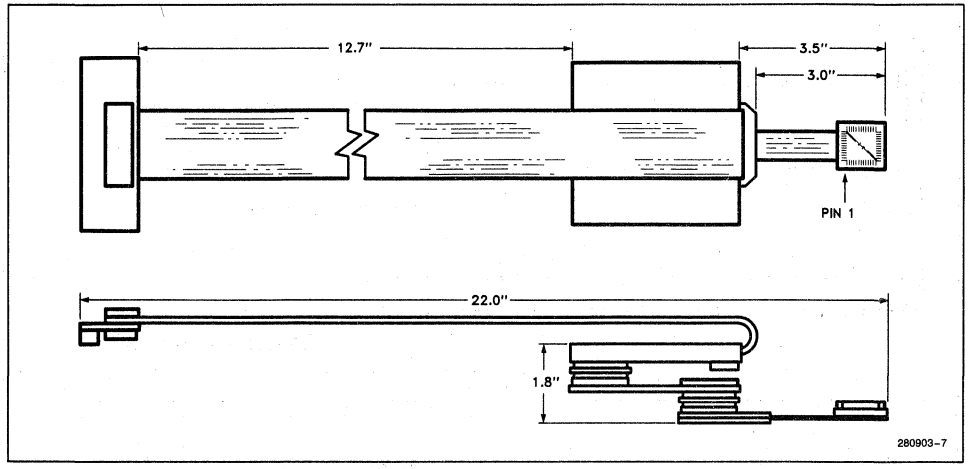
The Processor Module and Bus Isolation Board Dimensions (100 Pin PQFP)



8

ICE™-376 SPECIFICATIONS AND REQUIREMENTS

The Processor Module and Bus Isolation Board Dimensions (100 Pin PQFP) *(Continued)*



ELECTRICAL SPECIFICATIONS

The synchronization input lines must be valid for at least four CLK2 cycles as they are only sampled on every other cycle. These input lines are standard TTL inputs. The synchronization output lines are driven by TTL open collector outputs that have 4.7K-ohm pull-up resistors. The synchronization input and output signals on the optional clips pod are standard TTL input and outputs.

AC Specifications With the Bus Isolation Board Installed.

Symbol	Parameter	Minimum	Maximum	Notes
t1	CLK2 period	50 nS	t1 Max	
t2a	CLK2 high time	t2a Min + 2 nS		@ 2V
t3b	CLK2 low time	t3b Min + 2 nS		@ 0.8v
t6	A1-A23 valid delay	t6 Min + 3.5 nS	t6 Max + 24.6 nS	CL = 120 pF
t7	A1-A23 float delay	t14 Min + 5.5 nS	t14 Max + 37.6 nS	
t8	BLE#, BHE# LOCK# valid delay	t8 Min + 3.5 nS	t8 Max + 24.6	CL = 75pF
t9	BLE#, BHE# LOCK# float delay	t14 Min + 5.5 nS	t14 Max + 37.6	
t10	W/R#, M/IO#, D/C#, ADS# valid delay	t10 Min + 3.5 nS	t10 Min + 24.6	CL = 75 pF
t11	W/R#, M/IO#, D/C#, ADS# float delay	t14 Min + 5.5 nS	t14 Max + 37.6	
t12	D0-D15 write data valid delay	t12 Min + 4.5 nS	t12 Max + 20.6	CL = 120 pF
t13	D0-D15 write data float delay	7.5 nS	45.6 nS	
t14	HLDA valid delay	t14 Min = 3 nS	t14 Max + 21.2 nS	
t16	NA# hold time	t16 Min + 10.6 nS		
t20	READY# hold time	t20 Min + 10.6 nS		
t21	D0-D15 read setup time	t21 Min + 8.5 nS		
t22	D0-D15 read hold time	t22 Min + 7.6 nS		
t24	HOLD hold time	t24 Min + 10.6 nS		
t25	RESET setup time	t25 Min + 2.1 nS		
t26	RESET hold time	t26 Min + 2.1 nS		
t28	NMI, INTR hold time	t28 Min + 10.6 nS		
t30	PEREQ, ERROR#, BUSY# hold time	t30 Min + 10.6 nS		

SPECIFICATIONS

Emulator Capacitance Specifications With Target-Adapter Cable Installed

Symbol	Description	Typical (Note 1)
C_{IN}	Input Capacitance	
	CLK2	55pf
	READY#, ERROR#	35pf
	HOLD, BUSY#, PEREQ, NA#,	
	INTR, NMI	20pf
	RESET	30pf
C_{OUT}	Output or I/O Capacitance	
	D15-D0	50pf
	A15-A1, BLE#	40pf
	A23-A16, BHE#, D/C#	30pf
	HLDA, W/R#	55pf
	ADS#, M/IO#, LOCK#	35pf

Note 1: Not tested. These specifications include the 80376 component and all additional emulator loading.

Emulator DC Specifications Without the Bus Isolation Board Installed

Item	Description	Max.	Notes
PM-I_{CC}	Processor Module Supply Current	376-I_{CC} + 940 mA	
I_{IH}	Input High Leakage Current		
	A23-A1, BLE#, BHE#, D/C#, HLDA	0.02 mA	1
	D15-D0	0.06 mA	1
	ADS#, M/IO#, LOCK#, READY#, ERROR#	0.01 mA	1
	W/R#	0.03 mA	1
	CLK2	0.04 mA	1
	RESET	0.06 mA	2
I_{IL}	Input Low Leakage Current		
	A23-A1, BLE#, BHE#, D/C#	0.6 mA	1
	D15-D0	0.06 mA	1
	ADS#, M/IO#, LOCK#, READY#, ERROR#	0.01 mA	1
	W/R#	0.51 mA	1
	CLK2	0.62 mA	1
	RESET	0.6 mA	2
	HLDA	0.02 mA	1

Note 1: This specification is the DC input loading of the emulator circuitry only and does not include any 80376 leakage current.
Note 2: This specification replaces the 80376 specification for this signal.

SPECIFICATIONS

Emulator DC Specifications With the Bus Isolation Board Installed

Item	Description	Min.	Max.
BIB-I_{CC}	Bus Isolation Board Supply Current		PM-I_{CC}+ 350 mA
V_{OL}	Output Low Voltage (I_{OL} = 48 mA)		
	A23-A1, BLE#, BHE#, D/C#, ADS#		0.5 v
	D15-D0, M/IO#, LOCK#, W/R#		0.5 v
	HLDA (I_{OL} = 24 mA)		0.44. v
	Output High Voltage (I_{OH} = 3 mA)		
	A23-A1, BLE#, BHE#, D/C#, ADS#	2.4 v	
	D15-D0, M/IO#, LOCK#, W/R#	2.4 v	
	HLDA (I_{OH} = 24 mA)	3.8 v	
I_{IH}	Input High Current		
	CLK2, RESET		1.0 μA
	READY#		25 μA
I_{IL}	Input Low Current		
	CLK2, RESET		1.0 μA
	READY#		250 μA
I_{IO}	Output Leakage Current		
	A23-A1, BLE#, BHE#, D/C#, ADS#		±20 μA
	D15-D0, M/IO#, LOCK#, W/R#		±20 μA

PROCESSOR MODULE INTERFACE CONSIDERATIONS

With the processor module directly attached to the target system without using the bus isolation board, the target system must meet the following requirements.

- The user bus controller must only drive the data bus during a valid read cycle of the emulator processor or while the emulator processor is in a hold state (the emulator processor uses the data bus to communicate with the emulator hardware).
- Before driving the address bus, the user system must gain control by asserting HOLD and receiving HLDA.
- The user reset signal is disabled during the interrogation mode. It is enabled in emulation, but is delayed by 2 or 4 CLK2 cycles.
- The user system must be able to drive one additional TTL load on all signals that go to the emulation processor.

When the target system does not satisfy the first two restrictions, the bus isolation board is used to isolate the emulation processor from the target system. With the isolation board installed, the processor CLK2 is restricted to running at 20 MHz.

The processor module derives its DC power from the target system through the 80376 socket. It requires 1400mA, including the 80376 current. The isolation board requires an additional 350mA.

The processor must be socketed, for example using Textool 2-0100-07243-000 or AMP 821949-4 sockets.

The printed circuit board design should locate the processor socket at the physical ends of the printed circuit board traces that connect the processor to the other logic of the target system. This reduces transmission line noise. Additionally, if the target system is enclosed in a box, pin one of the processor socket should be oriented away from the target system's box opening to make connecting the target-adapter cable easier.

ORDERING INFORMATION

SERVICE, SUPPORT, AND TRAINING

To augment its development tools, Intel offers a full array of seminars, classes, and workshops, field application engineering expertise, hotline technical support, and on-site service.

Intel also offers a Gold Software Support package which includes:

Technical software information phone support,
Automatic distribution of software and documentation updates
Access to the "ToolTalk" electronic bulletin board

Intel's Hardware Support package includes:

Technical hardware information phone support,
Warrantee on parts/labor/material
On-site hardware support
One Customer Training Course of choise, plus discounts on additional customer training and SE consulting

SOFTWARE DEVELOPMENT TOOL

Order Code	Description
D86ASM386NL	DOS Macro Assembler supports 80386/80376
D86C386NL	DOS C Compiler supports 80386/80376
D86PLM386NL	DOS PL/M Compiler supports 80386/80376
D86FOR386NL	DOS FORTRAN Compiler supports 80386/80376
D86RLL386NL	DOS S/W DEV Package Builder/Binder/Mapper/ Librarian. Supports 80386/ 80376
DASM386PLUS	DOS ASM Developers Kit, Inc. ASM, NUM, RLL and EMUL
DB386	DOS S/W Debugger for 80386
D86NUM387NL	DOS 80387 Numerics Libraries
EMUL387SU	387 Numerics Coprocessor S/W Emulator object code
EMUL387RO	EMUL-387 to form derivative works. Requires incorporation fee Requires prior purchase of EMUL-387RF
EMUL387RF	EMUL-387 one time incorp fee

IN-CIRCUIT EMULATOR

pICE376D	ICE376 In-circuit Emulator for 80376 component. Operates to 16 MHz. Includes control unit, power supply, 376 Processor Module with PQFP adaptor, Stand-Alone Self-Test board, bus Isolation Board, and DOS 3.x host software and interface cable.
ICE37620D	ICE376 In-circuit Emulator for 80376 component. Operates to 20 MHz. Includes control unit, power supply, 376 Processor Module with PQFP adaptor, Stand-Alone Self-Test board, bus Isolation Board, and DOS 3.x host software and interface cable.
REM386SX376	2 Mbytes relocatable expansion memory
pICE386TO376D	Conversion kit to adapt ICE386 25 MHz emulator to support the 80376 component. Operates to 16 MHz. Includes ICE376 emulator Processor Module and DOS 3.x host software.
pICE386SXTO376D	Conversion kit to adapt ICE386SX 16 or 20 MHz emulator to support the 80376 component. Operates to 16 MHz. Includes ICE376 emulator Processor Module and DOS 3.x host software.

intel

ORDERING INFORMATION

p88PGAADAPT	Adaptor for ICE376 emulator to support 88 pin PGA component packaging.
pICE3XXCPO	Clips Pod Option for ICE376, ICE386SX 16 or 20 MHz, ICE386 25 MHz, and ICE386DX 33 MHz emulators.
pICE3XXTTB	Time Tag Board Option for ICE376, ICE386SX 16 or 20 MHz, ICE386 25 MHz, and ICE386DX 33 MHz emulators.
DT0AB	2 MB Intel Above Board.

ICD-486/25 IN-CIRCUIT DEBUGGER



280872-1

i486™ MICROPROCESSOR IN-CIRCUIT DEBUGGER

Intel's ICD-486/25, the in-circuit debugger for the 25 MHz i486™ microprocessor, represents a new generation of in-circuit emulation technology. From the inventor of the microprocessor comes a development tool that delivers complete access to the i486 architecture. ICD-486/25 is the first development tool which allows users to debug high-speed, cached applications at the full speed of the target processor. ICD-486/25 embodies exclusive technology, giving users symbolic access to the internal processor states that would not be accessible in any other way. With Intel's exclusive technology, users can be assured that the ICD-486/25 provides complete accuracy when debugging cached applications in real-time.

FEATURES

- Real-time emulation at the full speed of the i486 microprocessor
- Full development and debug support for the i486 microprocessor on-chip caching and numerics
- Programming support for the i486 microprocessor real mode **and** native protected modes
- Non-intrusive operation, allowing the target system to be debugged without modification
- Ability to set up to sixteen software breakpoints and four hardware breakpoints on execution addresses, data writes, or data accesses
- Sync in and out lines for connecting an ICD-486/25 to a high-speed logic analyzer to provide trace information and bus breakpoints
- Provides full symbolic information to display and modify all registers of the i486 microprocessor

8

November 1990
Order Number: 280872-002

intel

FEATURES

FULL-SPEED DEBUG AND DEVELOPMENT

The ICD-486/25 In-circuit Debugger provides sophisticated real-time hardware and software debug capabilities for i486 microprocessor based designs. The user can run at the full speed of the target processor, ensuring that elusive timing bugs will be found. And, because the ICD-486/25 is non-intrusive, your target system being developed can be the same as your final target system.

DEBUG CACHED APPLICATIONS

Until now, it has been extremely difficult to accurately debug high-speed, cached microprocessor applications. However, by incorporating Intel's exclusive technology, the ICD-486/25 allows users to debug applications which use the on-chip caching features of the i486 microprocessor. ICD-486/25 provides complete debugging accuracy whether the cache is on or off.

IDEAL FOR ALL STAGES OF DEVELOPMENT

The ICD-486/25 can be used by both hardware and software developers, at any stage of design. Early in the development process the ICD-486/25 allows prototype development and software debugging when using the optional REM board. Later in the design cycle, the ICD-486/25 can be used to integrate hardware and software modules.

SPEEDING DEVELOPMENT WITH SYMBOLICS

With symbolic debugging, memory locations can be examined or modified using symbolic references to the original program, such as a procedure or variable name, line number, or program label. Microprocessor data structures, such as registers, descriptor tables, and page tables, can also be examined and modified using symbolic names rather than cumbersome linear or physical addresses. Optimal symbolic debugging can be achieved when using the ICD-486/25 with Intel languages.

THE COMPLETE STORY

For advanced hardware debugging, the ICD-486/25 has been designed to work with high-speed logic analyzers. The standard ICD-486/25 ships with a Logic Analyzer Interface (LAI) board providing access to all chip signals which may be used to trigger a logic analyzer. With a user-supplied interface, the ICD-486/25 and logic analyzer can work in combination to monitor and recognize bus activity.

SOFTWARE COMPLETES THE SYSTEM

Intel provides a comprehensive software development environment to complement the ICD-486/25, delivering the most complete 32-bit microprocessor development environment available from a single vendor. Intel's i486 software development tools for the 386™ and i486 microprocessor families include 32-bit ANSI C, FORTRAN 77, and PL/M compilers, as well as 32-bit assembly language, linkage, IEEE math, run-time libraries and system software builders with full access to all aspects of the i486 microprocessor. In addition, all translators are object code compatible. Architectural extensions in the high-level languages allow hardware features such as interrupts, input/output or flags to be controlled directly, avoiding the maintenance of assembly routines.

Intel's software environment includes the sophisticated source-level DOS DB-386 software debugger and execution environment, allowing i486 software applications to be tested and debugged directly on a standard 386 microprocessor-based PC.

To provide full access to the power of the i486 architecture, the software portfolio incorporates a unique, sophisticated, and very powerful system builder, simplifying the generation of protected mode systems. To further reduce the task of integrating software into the final target configuration, Intel i486 microprocessor development tools produce code which can be directly downloaded target system ROM or converted into standard hex code.

intel

FEATURES

THE RIGHT TOOL FOR THE JOB

The ICD-486/25, representing a new generation of in-circuit emulation technology, is the right tool to use when your product development schedules are tight and your product quality requirements are high. Intel's exclusive technology allows you to debug cached applications at the full speed of the i486 microprocessor, and the symbolic debug information can vastly improve your productivity.

THE TOOL FOR THE FUTURE

The ICD-486/25 was designed to be easily, and rapidly, convertible to support the newest speeds of the i486 microprocessor. You can be assured that your investment in the ICD-486/25 today will put you squarely on the upgrade path to higher speed components when they are made available.

WORLDWIDE, WORLD CLASS SERVICES

Augmenting Intel i486 microprocessor development tools is a full array of seminars, classes, and workshops; on-site consulting services; field application engineering expertise; telephone hotline support; and software and hardware maintenance contracts.

ICD-486/25 SPECIFICATIONS AND REQUIREMENTS

HOST SYSTEM REQUIREMENTS

The user supplied host system can be an IBM® PC/AT® or Personal System/2® Model 60, or Model 80 or fully compatible system. Host system requirements to run the in-circuit debugger include the following:

- DOS version 3.2 or later
- 640K bytes of RAM in conventional memory
- An Above™ board with 1 megabyte of RAM configured in expanded memory mode, EMM.SYS version 3.2, or later, or
- One megabyte of RAM configured as expanded memory using QEMM.SYS or 386MAX
- A hard disk with 2 megabytes of free space
- A serial port

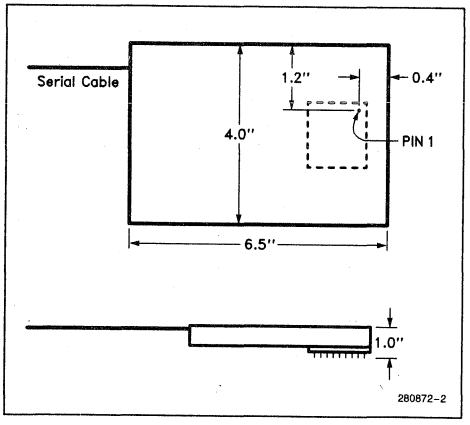


Figure 1: In-circuit Board (ICB)

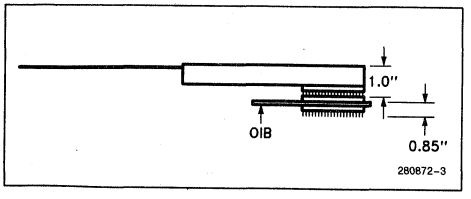


Figure 2: ICB with Optional Isolation Board (OIB) installed

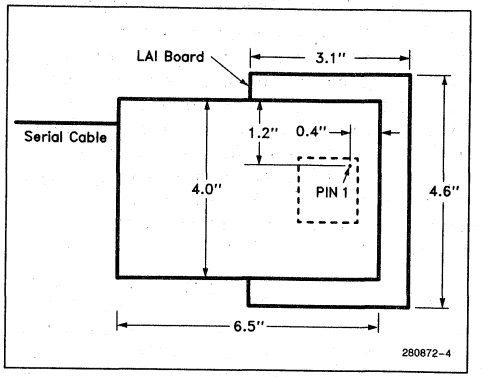


Figure 3: ICD with Logic Analyzer Interface (LAI) board installed

ELECTRICAL CHARACTERISTICS

100-120V or 220-240V selectable
50-60 Hz
2 amps (AC max) @ 120V
1 amp (AC max) @ 240V

ENVIRONMENTAL CHARACTERISTICS

Operating Temperature: +10° C to +40° C (50 to 104° F)
Operating Humidity: Maximum of 85% relative humidity, non-condensing

ELECTRICAL SPECIFICATIONS

The synchronization input line must be valid for at least two CLK cycles. The synchronization input and output signals are standard TTL input and outputs.

ICD-486/25 SPECIFICATIONS AND REQUIREMENTS

ICD-486/25 INTERFACE CONSIDERATIONS

With the ICB directly attached to the target system without using the optional isolation board, the target system must meet the following requirements:

- The bus controller must only enable data transceivers onto the bus during valid read cycles of the 486 CPU.
- READY# cannot be used with BREQ to terminate outstanding bus requests. (i.e., when using the ICD-486/25, BREQ will be asserted when there is not a corresponding assertion of ADS#).
- Before another bus master drives the local processor address bus, the other bus master must gain access to the address bus through the use of HOLD HLDA, AHOLD or BOFF#.
- The user system must be able to drive one additional CMOS load (approximately 25pF) on all signals that go to the emulation processor.

If the target system does not satisfy these restrictions, the optional isolation board should be used to isolate the emulation processor from the target system. To guarantee proper operation with the optional isolation board, the clock period should be increased by the round trip buffer delay (10ns) unless the target system design already has enough timing margin.

The processor module derives its DC power from the target system through the i486 CPU socket. It requires 1300mA, including the i486 microprocessor current. The optional isolation board requires an additional 500mA.

The processor must be socketed. The printed circuit board design should locate the processor socket at the physical ends of the printed circuit board traces that connect the processor to the other logic of the target system. This reduces transmission line noise. Additionally, if the target system is enclosed in a box, pin one of the processor socket should be oriented to make connecting the hinge cable easier. The ICD-486/25 hinge cable adds an additional 10pF of capacitive loading and approximately 0.5ns of propagation delay to each i486 CPU signal.

Pins specified as N.C. in the i486 CPU pin description must be left unconnected. Connection of any of these pins to power, ground or any other signal may cause the processor or the ICD-486/25 debugger to malfunction.

Symbol	Parameter	Minimum	Maximum	Notes
t6	A2-A31 valid delay	t6 Min + 1.5ns	t6 Max + 5ns	
t6	BE0-3#, M/IO#, W/R#, ADS#, HLDA valid delay	t6 Min + 2.5ns	t6 Max + 8ns	
t6	A2-A31 valid delay	t6 Min + 1.5ns	t6 Max + 11ns	1
t6	BE0-3#, M/IO#, W/R#, ADS# valid delay	t6 Min + 2.5ns	t6 Max + 14ns	1
t8a	BLAST# valid delay	t8a Min + 2.5ns	t8a Max + 8ns	
t10	D0-D31 write data valid delay	t10 Min + 1.5ns	t10 Max + 5ns	
t22	A4-A31, D0-D31 input set-up time	t22 Min + 5ns		

Note 1: Use these specifications for any bus cycle that begins on the same clock that HLDA is de-asserted.

ICD-486/25 SPECIFICATIONS AND REQUIREMENTS

DC SPECIFICATIONS WITH OPERATION ISOLATION BOARD INSTALLED

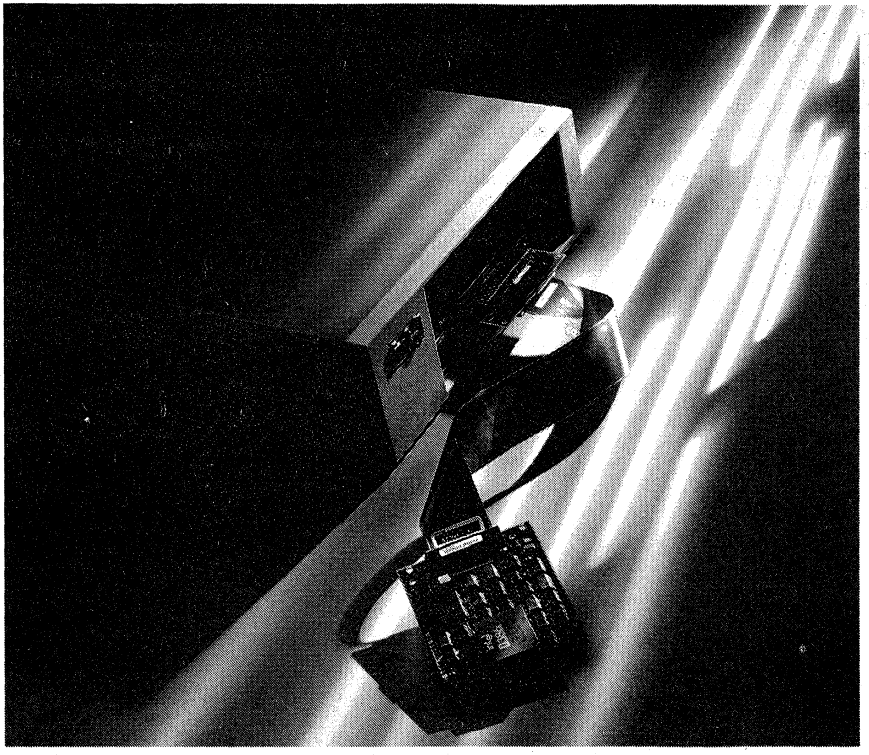
Item	Description		Minimum	Maximum	Notes
V_{OH}	Output High Voltage				
	A2-A31,D0-D31	(I_{OH} = 15 mA)	2.4V		
	BE0-3#, M/IO#,	(I_{OH} = 3 mA)	2.4V		
	W/R#, ADS#, BLAST#	(I_{OL} = 3 mA)	2.4V		
	HLDA	(I_{OH} = 3.2 mA)	2.4V		
V_{OL}	Output Low Voltage				
	A2-A31,D0-D31	(I_{OL} = 64 mA)		0.55V	
	BE0-3#, M/IO#,	(I_{OH} = 64 mA)		0.55V	
	W/R#, ADS#, BLAST#	(I_{OL} = 64 mA)		0.55V	
	HLDA	(I_{OL} = 24 mA)		0.5V	
I_{LI}	Input Leakage Current				
	A2-A31, D0-D31			±15uA	
I_{IL}	Input High Current				
	CLK, RESET, RDY#, BRDY#, BOFF#, AHOLD			25uA	1
I_{IH}	Input Low Current				
	CLK, RESET, RDY#, BRDY#, BOFF#, AHOLD			−250uA	1

Note 1: These specifications are for the OIB only and do not include any processor module loading.

ORDERING INFORMATION

Order Code	Description
ICD48625D	In-circuit debugger for the i486 microprocessor. Operates to 25MHz. Includes hardware debug module, power supply, isolation board, stand alone self-test chassis, flexible hinge cable, socket accessory assortment, user documentation, DOS host software and interface cable.
ICD486CON33D	Identical to the ICD48625D, but includes a prepaid upgrade to 33 MHz i486 microprocessor support when available.
ICD486CON33DS	Identical to the ICD486CON33D, but includes an additional 12 months of hardware and software maintenance and support.
ICD486LAI	Additional Logic Analyzer Interface board.
ICD48625DS	Identical to ICD48625D, but includes an additional 12 months of hardware and software maintenance and support.
486HNGCBLA	Additional flexible, hinge cable assembly.
ICD486ACC	Additional ICD-486 socket accessory assortment and board separator.

INTEL386™ FAMILY IN-CIRCUIT EMULATORS



280850-1

ACCURATE AND SOPHISTICATED EMULATION FOR THE INTEL386™ FAMILY OF MICROPROCESSORS

Intel386™ In-circuit Emulators are the cornerstone of the optimum development solution for the Intel386 Family of microprocessors. From the inventor of the microprocessor comes a development tool that delivers absolute access to the sophistication of the architecture in a way that only Intel can.

Productivity boosting features such as symbolic debugging make Intel386 emulators easy to use and powerful. Intel product quality and world class technical support and service minimizes the "downtime" incurred in resolving problems. And your investment in development tools is protected via interchangeable probes for the 80386 DX, 80386 SX, and 80376 microprocessors.

Maximize your productivity with Intel development tools. Reduced time to market and increased market acceptance for your microprocessor-based product are the benefits when Intel is the choice.

8

*HP9000 is a trademark of Hewlett Packard.
ICE, iPAT, Above Board, Intel386, 386 DX, 386 SX, and 376 are trademarks of Intel Corporation.
IBM, PC AT, PS/2 are registered trademarks of International Business Machines Corporation.
GPIB-PCII, GPIB-PCIIA, and MC-GPIB are trademarks of National Instruments Corporation.

November 1990
Order Number: 280850-004

intel

FEATURES

INTEL ICE™-386 FAMILY IN-CIRCUIT EMULATOR FEATURES

- Unparalleled support of all of the Intel386 operating modes opens the door to the full potential of the Intel386 architecture
- Non-intrusive (zero wait-state) emulation to processor speeds of 33 MHz
- Versatile event recognition makes short work of uncovering complex bugs
- Dynamic trace display of bus and execution information during emulation
- A comprehensive software development system creates the most complete development environment available from a single vendor
- A companion performance analysis tool provides analysis of software for optimized performance and reliability
- Available on Hewlett-Packard HP9000 UNIX*.

FEATURES

100% ACCURATE EMULATION

Intel386 Family In-circuit Emulators embody technology that accesses internal processor states that are otherwise invisible. Intel386 microprocessors fetch and execute instructions in parallel; fetched instructions are not necessarily executed in any order. Because of this, an emulator without this capability is prone to error in determining what actually occurs inside the microprocessor. With Intel's technology, an Intel386 In-circuit Emulator displays execution history with one hundred percent accuracy and in real-time.

OPENING THE DOOR TO PROTECTED MODE

The Intel386 family of In-circuit Emulators opens the door to the full potential of the architecture with unparalleled support of protected mode. Not only does the emulator display and modify task state segments and global, local, and interrupt descriptor tables (with symbolic access to all descriptor components like privilege level and segment type), but emulator functions are sensitive to the operating mode of the processor, greatly improving ease of use.

The Intel386 family of In-circuit Emulators supports all aspects of protected mode addressing, including paged virtual memory. Processor tables are used to automatically translate virtual addresses to linear and physical addresses. Physical addresses can be translated to symbolic references to indicate the module, procedure, or data segment accessed. And when debugging a memory management system, components of the page table and directory can be displayed and modified.

FLEXIBLE AND VERSATILE EVENT RECOGNITION

Flexibility and versatility in event recognition makes short work of uncovering the most complex bugs. Bus event recognition circuitry may be used to trigger on specific or masked data input, output, read, written, or fetched at a physical address or range of addresses. Or on-chip debug registers may be used to trigger on virtual, linear, or symbolic addresses being executed, accessed, or written.

Versatility shows in other triggering options—upon a task switch, an external signal from another emulator or a logic analyzer, multiple occurrences of an event, a full trace buffer, halt or shutdown cycles, or interrupt acknowledge. And up to four sequential event triggers can be combined with a high-level construct.

The Intel386 family of In-circuit Emulators continuously captures all bus activity and, as an option, execution information, into a trace buffer of 4 K frames with PRE, POST, and CENTERED collection modes. The contents of the trace buffer can be displayed during full speed emulation in either execution cycle or machine-level instruction formats. Symbolic information can optionally be included in the trace display. A third trace display, the current chain of procedure calls, can be displayed when emulating high-level language programs.

SPEEDING DEVELOPMENT WITH SYMBOLICS

Intel386 processor data structures, such as registers, descriptor tables, and page tables, can be examined and modified using symbolic names. And with the symbolic debugging information that is a feature of Intel languages, memory locations can be accessed using symbolic references to the source program (such as a procedure and variable names, line numbers, or program labels) rather than via cumbersome virtual, linear, or physical addresses. The type information of variables (such as byte, word, record, or array) can also be displayed.

ACCESSING THE POWER

The power of the Intel386 In-circuit Emulator is reflected in the sophisticated user interface. Refined for ease-of-use, the command line interface contains many features to boost productivity and customize functionality.

On-line help, a syntax menu, command line editing, command history, and error message query promote ease of learning and use. I/O redirection and the ability to escape to the host operating system provide versatility for the power user. Customized procedures with variables and literal definitions can be created to assist in debugging or for manufacturing test or field service applications.

FEATURES

ADDITIONAL FEATURES

The Intel386 In-circuit Emulator can be combined with a variety of devices. I/O lines synchronize emulation starts and triggers with external tools such as a logic analyzer or another emulator. An optional Time Tag Board synchronizes multiple Intel386 emulators and records timestamp information in the trace buffer with 20 nanosecond resolution. An Optional Clips Pod allows 8 user defined data lines to be captured and displayed in the trace. The bus isolation board buffers the emulation processor from faults in an untested target. And with the Stand-Alone Self-Test board the emulator can be used to debug software before the target system is functional, as well as execute confidence tests.

THE INVESTMENT PICTURE

As designs move from one Intel386 Family processor to another, the reinvestment cost is limited to probes that adapt the emulator base to the specific processor. Beside cost savings, migration from one processor to another is accomplished with minimum disruption in the engineering environment, as the same command language applies to the entire emulator family.

intel

FEATURES

SOFTWARE COMPLETES THE SYSTEM

Intel wraps a comprehensive software development system around the emulator to deliver the most complete development environment available from a single vendor. Like the emulator, Intel's software development system supports every aspect of the Intel386 architecture.

Overlooked at times is the fact that a significant part of developing a system is making sure the code works. Intel languages and software debugger integrate seamlessly with the Intel386 emulator and provide the symbolics so important for efficient debugging. By using Intel software tools with the Intel386 emulator the full power of Intel development solution can be utilized.

The software development system offers a broad choice of languages with object code compatibility so performance can be maximized by using different languages for specialized, performance critical modules. Architectural extensions in the high-level languages allows hardware features such as interrupts, input/output, or flags to be controlled directly, avoiding the tediousness of coding assembly language routines.

Intel's software portfolio includes a unique, sophisticated, and very powerful system builder, simplifying the generation of protected mode systems. To further reduce the effort necessary to integrate software into the final target configuration, Intel tools produce ROM-able code directly from the development system.

OPTIMIZING PERFORMANCE AND RELIABILITY

A companion performance analysis tool, iPAT™-386, provides analysis of real-time software executing on 80386-based target systems. With iPAT-386, it is possible to speed-tune applications, optimize use of operating systems, determine response characteristics, and identify code execution coverage. And iPAT-386 can be used in conjunction with an Intel386 in-circuit emulator to control test conditions.

WORLD CLASS, WORLDWIDE SERVICES

Augmenting the Intel386 Family development tools is a full array of seminars, classes and workshops; on-site consulting services; field application engineering expertise; telephone hotline support; and software and hardware maintenance contracts.

ICE™-386 DX 33 MHz SPECIFICATIONS AND REQUIREMENTS

HOST SYSTEM REQUIREMENTS

The user supplied host system can be either an IBM® PC AT® or Personal System/2® Model 60. Host system requirements to run the emulator include the following:

- DOS version 4.0 or Hewlett Packard HP9000 UNIX
- 640 Kbytes of RAM in conventional memory
- An Above™ board with 1 megabyte of RAM configured in expanded memory mode, EMM.SYS software version 3.2
- A 20 MB hard disk
- A serial port or the National Instruments GPIB-PCII™, GPIB-PCIIA™, or MC-GPIB™ board
- A math coprocessor if either the optional time tag board is used or if a math coprocessor resides on the target system

ELECTRICAL CHARACTERISTICS

100–120V or 220–240V selectable
50–60 Hz
2 amps (AC max) @ 120V
1 amp (AC max) @ 240V

ENVIRONMENTAL CHARACTERISTICS

Operating temperature:	+10°C to +40°C (50°F to 104°F)
Operating Humidity:	Maximum of 85% relative humidity, non-condensing

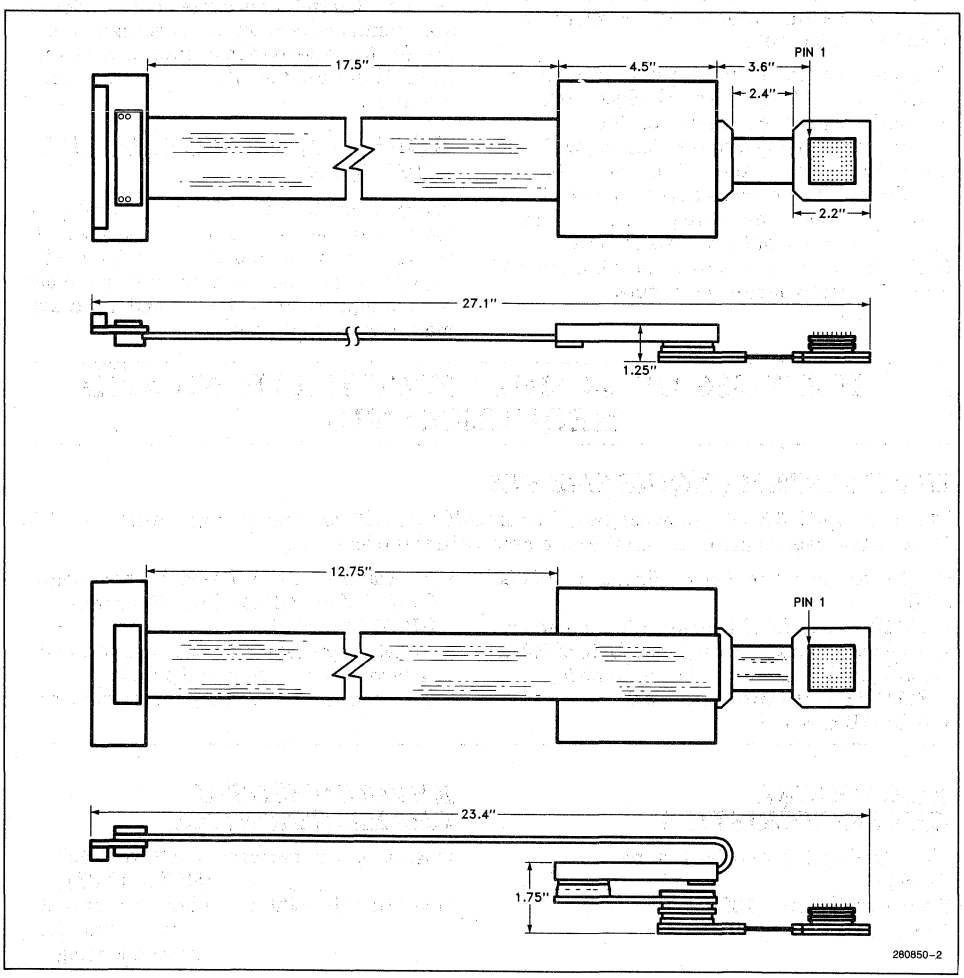
8

ICE™-386 DX 33 MHz SPECIFICATIONS AND REQUIREMENTS

The Emulator's Physical Characteristics

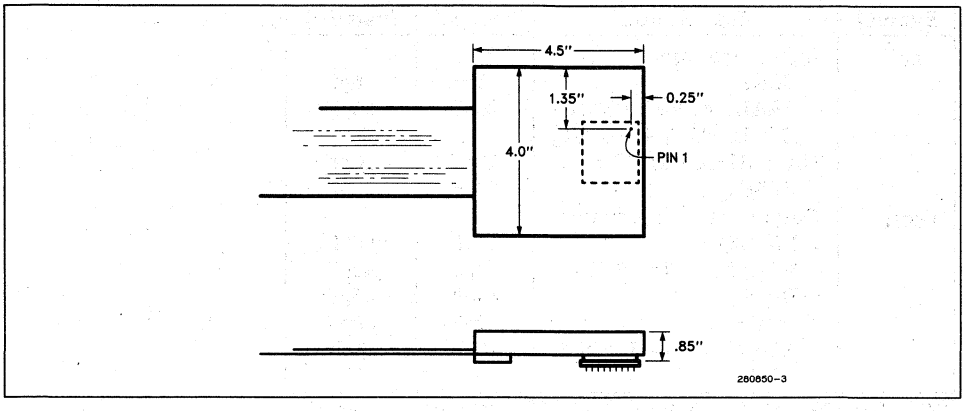
Unit	Width		Height		Length	
	inches	cm	inches	cm	inches	cm
Base Unit	13.4	34.0	4.6	11.7	11.0	27.9
Processor Module	3.8	9.7	0.7	1.8	4.4	11.2
Optional Isolation Board	3.8	9.7	0.5	1.3	4.4	11.2
Power Supply	7.7	19.6	4.1	10.4	11.0	27.9
User Cable	1.9	4.8			17.3	43.9
Target-Adapter Cable	2.3	5.3	0.5	1.3	5.8	14.7
Serial Cable					144	366
Optional Clips Pod	3.3	8.4	0.8	2.0	6.0	15.2

The Processor Module and Bus Isolation Board Dimensions



ICE™-386 DX 33 MHz SPECIFICATIONS AND REQUIREMENTS

The Processor Module and Bus Isolation Board Dimensions



ELECTRICAL SPECIFICATIONS

The synchronization input lines must be valid for at least four CLK2 cycles as they are only sampled on every other cycle. These input lines are standard TTL inputs. The synchronization output lines are driven by TTL open collector outputs that have 4.7K-ohm pull-up resistors. The synchronization input and output signals on the optional clips pod are standard TTL input and outputs.

AC Specifications With the Bus Isolation Board Installed

Symbol	Parameter	Minimum	Maximum	Notes
t1	CLK2 period	40 nS	t1 Max	
t2a	CLK2 high time	t2a Min + 2 nS		@ 2V
t3b	CLK2 low time	t3b Min + 2 nS		@ 0.8v
t6	A2-A31 valid delay	t6 Min + 3.5 nS	t6 Max + 24.6 nS	CL = 120 pF
t7	A2-A31 float delay	t14 Min + 5.5 nS	t14 Max + 32.6 nS	
t8	BE0# - BE3#, LOCK# valid delay	t8 Min + 3.5 nS	t8 Max + 24.6	CL = 75pF
t9	BE0# - BE3#, LOCK# float delay	t14 Min + 5.5 nS	t14 Max + 32.6	
t10	W/R#, M/IO#, D/C#, ADS# valid delay	t10 Min + 3.5 nS	t10 Min + 24.6	CL = 75 pF
t11	W/R#, M/IO#, D/C#, ADS# float delay	t14 Min + 5.5 nS	t14 Max + 32.6	
t12	D0-D31 write data valid delay	t12 Min + 4.5 nS	t12 Max + 20.6	CL = 120 pF
t13	D0-D31 write data float delay	7.5 nS	41.6 nS	
t14	HLDA valid delay	t14 Min = 3 nS	t14 Max + 21.2 nS	
t16	NA# hold time	t16 Min + 10.6 nS		
t18	BS16# hold time	t18 Min + 10.6 nS		
t20	READY# hold time	t20 Min + 10.6 nS		
t21	D0-D31 read setup time	t21 Min + 8.5 nS		
t22	D0-D31 read hold time	t22 Min + 7.6 nS		
t24	HOLD hold time	t24 Min + 10.6 nS		
t25	RESET setup time	t25 Min + 2.1 nS		
t26	RESET hold time	t26 Min + 2.1 nS		
t28	NMI, INTR hold time	t28 Min + 10.6 nS		
t30	PEREQ, ERROR#, BUSY# hold time	t30 Min + 10.6 nS		

SPECIFICATIONS

Emulator Capacitance Specifications

Symbol	Description	Typical	TAC Installed
C_{IN}	Input Capacitance		
	CLK2	35pF	45pF
	READY#, NMI, BS16#	25pF	35pF
	HOLD, BUSY#, PEREQ, NA#, INTR, ERROR#	10pF	20pF
	RESET	20pF	30pF
C_{OUT}	Output or I/O Capacitance		
	D0–D31	40pF	50pF
	A2–A31, BE0#–BE3#	30pF	40pF
	D/C#	35pF	45pF
	W/R#	40pF	50pF
	ADS#, M/IO#, LOCK#, HLDA	25pF	35pF

Note 1: Not tested. These specifications include the 80386 component and all additional emulator loading.
Note 2: The target-adapter cable adds a propagation delay of 0.5 nS.

Emulator DC Specifications Without the Bus Isolation Board Installed

Item	Description	Max.	Notes
PM-I_{CC}	Processor Module Supply Current	386-I_{CC} + 1.5 A	
I_{IH}	Input High Leakage Current		
	A2–A31, BE0#–BE3#, D0–D31	20μA	1
	HLDA, NMI, BS16#	10μA	1
	ADS#, M/IO#, LOCK#, READY#	10μA	1
	W/R#, D/C#	30μA	1
	CLK2	15μA	1
	RESET	5μA	2
I_{IL}	Input Low Leakage Current		
	A2–A31, BE0#–BE3#, D0–D31	600μA	1
	HLDA, NMI, BS16#	10μA	1
	ADS#, M/IO#, LOCK#, READY#	10μA	1
	W/R#	110μA	1
	D/C#	610μA	1
	CLK2	15μA	1
	RESET	5μA	2

Note 1: This specification is the DC input loading of the emulator circuitry only and does not include any 80386 leakage current.
Note 2: This specification replaces the 80386 specification for this signal.

SPECIFICATIONS

Emulator DC Specifications With the Bus Isolation Board Installed

Item	Description	Min.	Max.
BIB-I_{CC}	Bus Isolation Board Supply Current		PM-I_{CC} + 475 mA
V_{OL}	Output Low Voltage (I_{OL} = 48 mA)		
	A2–A31, BE0#–BE3#, D/C#, ADS#		0.5 v
	D0–D31, M/IO#, LOCK#, W/R#		0.5 v
	HLDA (I_{OL} = 24 mA)		0.44 v
V_{OH}	Output High Voltage (I_{OH} = 3 mA)		
	A2–A31, BE0#–BE3#, D/C#, ADS#	2.4 v	
	D0–D31, M/IO#, LOCK#, W/R#	2.4 v	
	HLDA (I_{OH} = 24 mA)	3.8 v	
I_{IH}	Input High Current		
	CLK2, RESET		1.0 μA
	READY#		25 μA
I_{IL}	Input Low Current		
	CLK2, RESET		1.0 μA
	READY#		250 μA
I_{IO}	Output Leakage Current		
	A2–A31, BE0#–BE3#, D/C#, ADS#		±20 μA
	D0–D31, M/IO#, LOCK#, W/R#		±20 μA

PROCESSOR MODULE INTERFACE CONSIDERATIONS

With the processor module directly attached to the target system without using the bus isolation board, the target system must meet the following requirements:

- The user bus controller must only drive the data bus during a valid read cycle of the emulator processor or while the emulator processor is in a hold state (the emulator processor uses the data bus to communicate with the emulator hardware).
- Before driving the address bus, the user system must gain control by asserting HOLD and receiving HLDA.
- The user reset signal is disabled during the interrogation mode. It is enabled in emulation, but is delayed by 2 or 4 CLK2 cycles.
- The user system must be able to drive one additional TTL load on all signals that go to the emulation processor.

When the target system does not satisfy the first two restrictions, the bus isolation board is used to isolate the emulation processor from the target system. With the isolation board installed, the processor CLK2 is restricted to running at 25 MHz.

The processor module derives its DC power from the target system through the 80386 socket. It requires 1500mA, including the 80386 current. The isolation board requires an additional 475mA.

The processor must be socketed. The printed circuit board design should locate the processor socket at the physical ends of the printed circuit board traces that connect the processor to the other logic of the target system. This reduces transmission line noise. Additionally, if the target system is enclosed in a box, pin one of the processor socket should be oriented to make connecting the processor module or target-adapter cable (TAC) easier.

The emulator uses the 386 microprocessor's pins C7, E13, and F13. The *80386 High Performance 32-Bit Microprocessor With Integrated Memory Management* data sheet specifies these pins as "N/C" (no connect). If the target system uses any of these pins, you must do one of the following:

- Use the bus isolation board.
- Use the target-adapter cable (TAC).
- Build an adapter to disconnect pins C7, E13, and F13 (i.e., a socket with these pins removed).

ICE™-386 DX 25 MHz SPECIFICATIONS AND REQUIREMENTS

HOST SYSTEM REQUIREMENTS

The user supplied host system can be either an IBM PC AT or Personal System/2 Model 60. Host system requirements to run the emulator include the following:

- DOS version 4.0, or Hewlett-Packard HP 9000 UNIX
- 640 Kbytes of RAM in conventional memory
- An Above board with 1 megabyte of RAM configured in expanded memory mode, EMM.SYS software version 3.2
- A 20 MB hard disk
- A serial port or the National Instruments GPIB-PCII, GPIB-PCIIA, or MC-GPIB board
- A math coprocessor if either the optional time tag board is used or if a math coprocessor resides on the target system

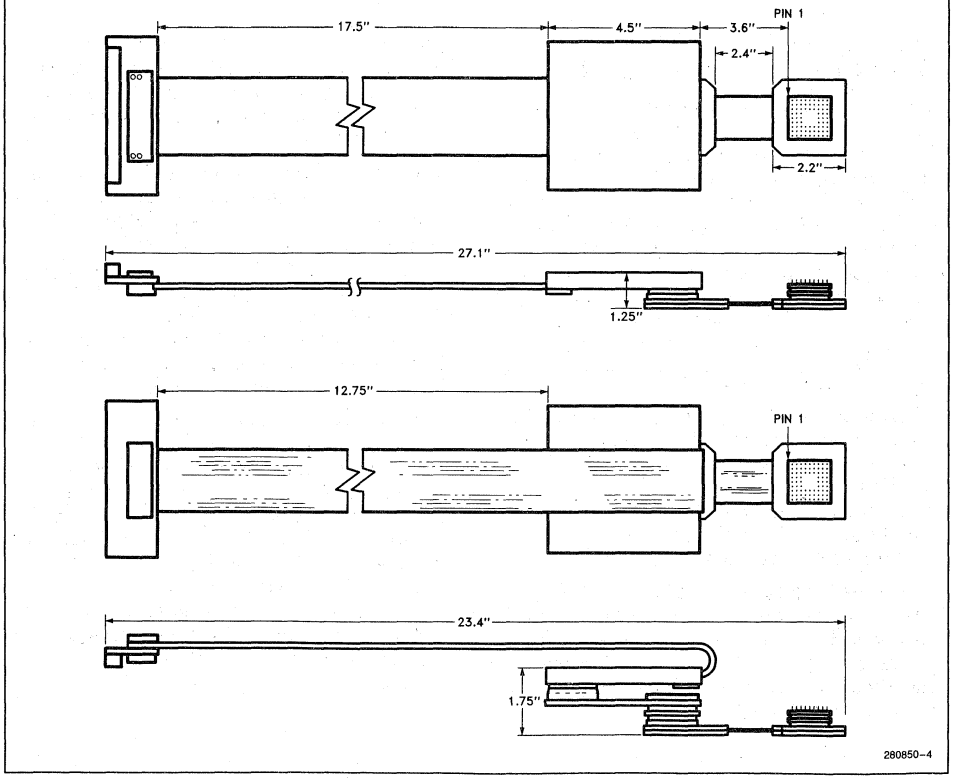
ELECTRICAL CHARACTERISTICS

100–120V or 220–240V selectable
50–60 Hz
2 amps (AC max) @ 120V
1 amp (AC max) @ 240V

ENVIRONMENTAL CHARACTERISTICS

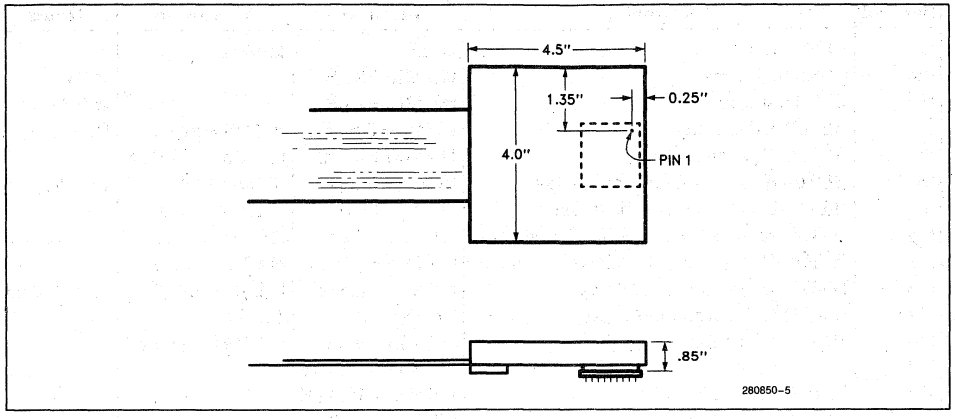
Operating temperature:	+10° to +40° C (50° to 104° F)
Operating Humidity:	Maximum of 85% relative humidity, non-condensing

The Processor Module and Bus Isolation Board Dimensions



ICE™-386 DX 25 MHz SPECIFICATIONS AND REQUIREMENTS

The Processor Module and Bus Isolation Board Dimensions



The Emulator's Physical Characteristics

Unit	Width		Height		Length	
	inches	cm	inches	cm	inches	cm
Base Unit	13.4	34.0	4.6	11.7	11.0	27.9
Processor Module	3.8	9.7	0.7	1.8	4.4	11.2
Optional Isolation Board	3.8	9.7	0.5	1.3	4.4	11.2
Power Supply	7.7	19.6	4.1	10.4	11.0	27.9
User Cable	1.9	4.8			17.3	43.9
Target-Adapter Cable	2.3	5.3	0.5	1.3	5.8	14.7
Serial Cable					144	366
Optional Clips Pod	3.3	8.4	0.8	2.0	6.0	15.2

ELECTRICAL SPECIFICATIONS

The synchronization input lines must be valid for at least four CLK2 cycles as they are only sampled on every other cycle. These input lines are standard TTL inputs. The synchronization output lines are driven by TTL open collector outputs that have 4.7K-ohm pull-up resistors. The synchronization input and output signals on the optional clips pod are standard TTL input and outputs.

8

ICE™-386 DX 25 MHz SPECIFICATIONS AND REQUIREMENTS

AC Specifications With the Bus Isolation Board Installed

Symbol	Parameter	Minimum	Maximum	Notes
t1	CLK2 period	50 nS	t1 Max	
t2a	CLK2 high time	t2a Min + 2 nS		@ 2V
t3b	CLK2 low time	t3b Min + 2 nS		@ 0.8v
t6	A2-A31 valid delay	t6 Min + 3.5 nS	t6 Max + 24.6 nS	CL = 120 pF
t7	A2-A31 float delay	t14 Min + 5.5 nS	t14 Max + 37.6 nS	
t8	BE0#-BE3#, LOCK# valid delay	t8 Min + 3.5 nS	t8 Max + 24.6	CL = 75pF
t9	BE0#-BE3#, LOCK# float delay	t14 Min + 5.5 nS	t14 Max + 32.6	
t10	W/R#, M/IO#, D/C#, ADS# valid delay	t10 Min + 3.5 nS	t10 Min + 24.6	CL = 75 pF
t11	W/R#, M/IO#, D/C#, ADS# float delay	t14 Min + 5.5 nS	t14 Max + 32.6	
t12	D0-D31 write data valid delay	t12 Min + 4.5 nS	t12 Max + 20.6	CL = 120 pF
t13	D0-D31 write data float delay	7.5 nS	41.6 nS	
t14	HLDA valid delay	t14 Min = 3 nS	t14 Max + 21.2 nS	
t16	NA# hold time	t16 Min + 10.6 nS		
t18	BS16# hold time	t18 Min + 10.6 nS		
t20	READY# hold time	t20 Min + 10.6 nS		
t21	D0-D31 read setup time	t21 Min + 8.5 nS		
t22	D0-D31 read hold time	t22 Min + 7.6 nS		
t24	HOLD hold time	t24 Min + 10.6 nS		
t25	RESET setup time	t25 Min + 2.1 nS		
t26	RESET hold time	t26 Min + 2.1 nS		
t28	NMI, INTR hold time	t28 Min + 10.6 nS		
t30	PEREQ, ERROR#, BUSY# hold time	t30 Min + 10.6 nS		

SPECIFICATIONS

Emulator Capacitance Specifications

Symbol	Description	Typical	TAC Installed
C_{IN}	Input Capacitance		
	CLK2	35pF	45pF
	READY#, NMI, BS16#	25pF	35pF
	HOLD, BUSY#, PEREQ, NA#, INTR, ERROR#	10pF	20pF
	RESET	20pF	30pF
C_{OUT}	Output or I/O Capacitance		
	D0-D31	40pF	50pF
	A2-A31, BE0#-BE3#	30pF	40pF
	D/C#	35pF	45pF
	W/R#	40pF	50pF
	ADS#, M/IO#, LOCK#, HLDA	25pF	35pF

Note 1: Not tested. These specifications include the 80386 component and all additional emulator loading.
Note 2: The target-adapter cable adds a propagation delay of 0.5 nS.

SPECIFICATIONS

Emulator DC Specifications Without the Bus Isolation Board Installed

Item	Description	Max.	Notes
PM-I_{CC}	Processor Module Supply Current	386SX-I_{CC} + 1.5 A	
I_{IH}	Input High Leakage Current		
	A2–A31, BE0#–BE3#, D0–D31	20μA	1
	HLDA, NMI, BS16#	10μA	1
	ADS#, M/IO#, LOCK#, READY#	10μA	1
	W/R#, D/C#	30μA	1
	CLK2	15μA	1
	RESET	5μA	2
I_{IL}	Input Low Leakage Current		
	A2–A31, BE0#–BE3#, D0–D31	600μA	1
	HLDA, NMI, BS16#	10μA	1
	ADS#, M/IO#, LOCK#, READY#	10μA	1
	W/R#110	μA	1
	D/C#	610μA	1
	CLK2	15μA	1
	RESET	5μA	2

Note 1: This specification is the DC input loading of the emulator circuitry only and does not include any 80386 leakage current.
Note 2: This specification replaces the 80386 specification for this signal.

Emulator DC Specifications With the Bus Isolation Board Installed

Item	Description	Min.	Max.
BIB-I_{CC}	Bus Isolation Board Supply Current		PM-I_{CC} + 475 mA
V_{OL}	Output Low Voltage (I_{OL} = 48 mA)		
	A2–A31, BE0#–BE3#, D/C#, ADS#		0.5 v
	D0–D31, M/IO#, LOCK#, W/R#		0.5 v
	HLDA (I_{OL} = 24 mA)		0.44 v
V_{OH}	Output High Voltage (I_{OH} = 3 mA)		
	A2–A31, BE0#–BE3#, D/C#, ADS#	2.4 v	
	D0–D31, M/IO#, LOCK#, W/R#	2.4 v	
	HLDA (I_{OH} = 24 mA)	3.8 v	
I_{IH}	Input High Current		
	CLK2, RESET		1.0 μA
	READY#		25 μA
I_{IL}	Input Low Current		
	CLK2, RESET		1.0 μA
	READY#		250 μA
I_{IO}	Output Leakage Current		
	A2–A31, BE0#–BE3#, D/C#, ADS#		±20 μA
	D0–D31, M/IO#, LOCK#, W/R# ±20 μA		

intel

ICE™-386 SX 20 MHz SPECIFICATIONS AND REQUIREMENTS

PROCESSOR MODULE INTERFACE CONSIDERATIONS

With the processor module directly attached to the target system without using the bus isolation board, the target system must meet the following requirements:

- The user bus controller must only drive the data bus during a valid read cycle of the emulator processor or while the emulator processor is in a hold state (the emulator processor uses the data bus to communicate with the emulator hardware).
- Before driving the address bus, the user system must gain control by asserting HOLD and receiving HLDA.
- The user reset signal is disabled during the interrogation mode. It is enabled in emulation, but is delayed by 2 or 4 CLK2 cycles.
- The user system must be able to drive one additional TTL load on all signals that go to the emulation processor.

When the target system does not satisfy the first two restrictions, the bus isolation board is used to isolate the emulation processor from the target system. With the isolation board installed, the processor CLK2 is restricted to running at 25 MHz.

The processor module derives its DC power from the target system through the 80386 socket. It requires 1500mA, including the 80386 current. The isolation board requires an additional 475mA.

The processor must be socketed. The printed circuit board design should locate the processor socket at the physical ends of the printed circuit board traces that connect the processor to the other logic of the target system. This reduces transmission line noise. Additionally, if the target system is enclosed in a box, pin one of the processor socket should be oriented to make connecting the processor module or target-adapter cable (TAC) easier.

The emulator uses the 386 microprocessor's pins C7, E13, and F13. The *80386 High Performance 32-Bit Microprocessor With Integrated Memory Management* data sheet specifies these pins as "N/C" (no connect). If the target system uses any of these pins, you must do one of the following:

- Use the bus isolation board.
- Use the target-adapter cable (TAC).
- Build an adapter to disconnect pins C7, E13, and F13 (i.e., a socket with these pins removed).

HOST SYSTEM REQUIREMENTS

The user supplied host system can be either an IBM PC AT or Personal System/2 Model 60. Host system requirements to run the emulator include the following:

- DOS version 4.0, or Hewlett-Packard HP9000 UNIX
- 640 Kbytes of RAM in conventional memory
- An Above board with 1 megabyte of RAM configured in expanded memory mode, EMM.SYS software version 3.2
- A 20 MB hard disk
- A serial port or the National Instruments GPIB-PCII, GPIB-PCIIA, or MC-GPIB board
- A math coprocessor if either the optional time tag board is used or if a math coprocessor resides on the target system

ELECTRICAL CHARACTERISTICS

100–120V or 220–240V selectable
50–60 Hz
2 amps (AC max) @ 120V
1 amp (AC max) @ 240V

ENVIRONMENTAL CHARACTERISTICS

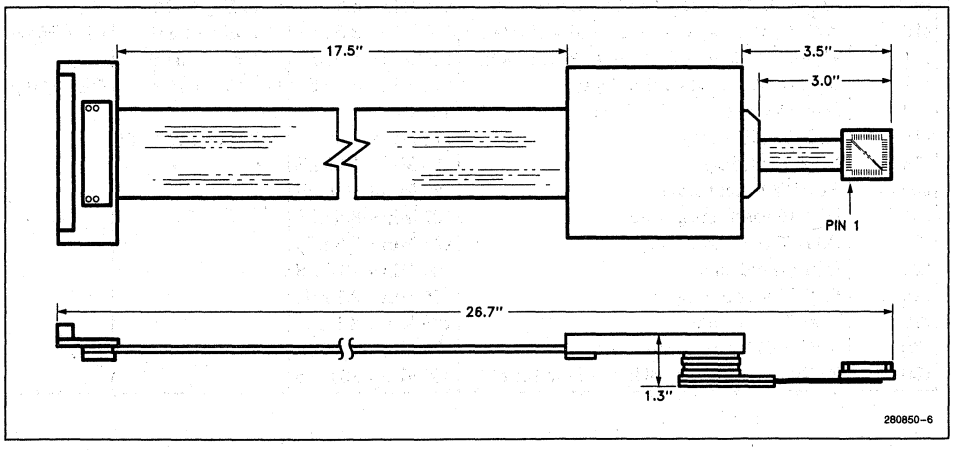
Operating temperature:	+10°C to +40°C (50°F to 104°F)
Operating Humidity:	Maximum of 85% relative humidity, non-condensing

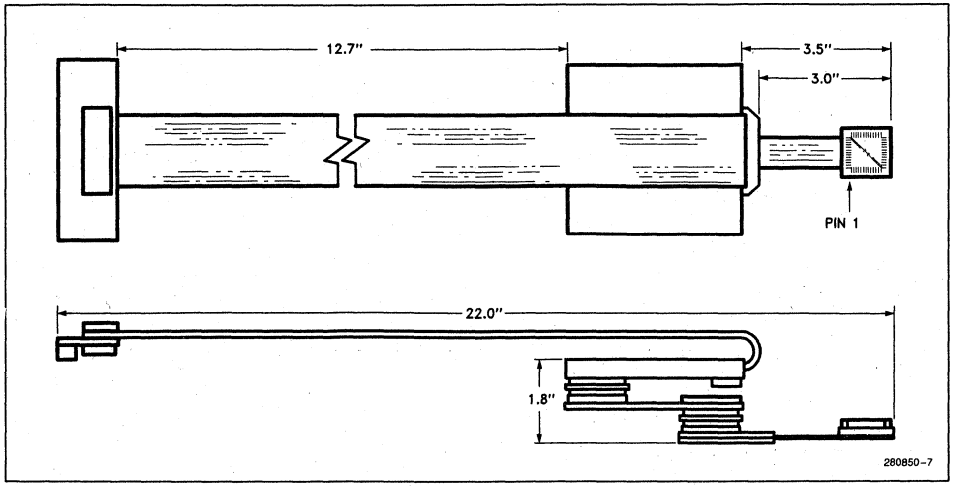
ICE™-386 SX 20 MHz SPECIFICATIONS AND REQUIREMENTS

The Emulator's Physical Characteristics

Unit	Width		Height		Length	
	inches	cm	inches	cm	inches	cm
Base Unit	13.4	34.0	4.6	11.7	11.0	27.9
Processor Module	3.8	9.7	0.7	1.8	4.4	11.2
Optional Isolation Board	3.8	9.7	0.5	1.3	4.4	11.2
Power Supply	7.7	19.6	4.1	10.4	11.0	27.9
User Cable	1.9	4.8			17.3	43.9
Target-Adapter Cable	2.3	5.3	0.5	1.3	5.1	13.0
Serial Cable					144	366
Optional Clips Pod	3.3	8.4	0.8	2.0	6.0	15.2

The Processor Module and Bus Isolation Board Dimensions





8

ICE™-386 SX 20 MHz SPECIFICATIONS AND REQUIREMENTS

ELECTRICAL SPECIFICATIONS

The synchronization input lines must be valid for at least four CLK2 cycles as they are only sampled on every other cycle. These input lines are standard TTL inputs. The synchronization output lines are driven by TTL open collector outputs that have 4.7K-ohm pull-up resistors. The synchronization input and output signals on the optional clips pod are standard TTL input and outputs.

AC Specifications With the Bus Isolation Board Installed

Symbol	Parameter	Minimum	Maximum	Notes
t1	CLK2 period	50 nS	t1 Max	
t2a	CLK2 high time	t2a Min + 2 nS		@ 2V
t3b	CLK2 low time	t3b Min + 2 nS		@ 0.8v
t6	A1-A23 valid delay	t6 Min + 3.5 nS	t6 Max + 24.6 nS	CL = 120 pF
t7	A1-A23 float delay	t14 Min + 5.5 nS	t14 Max + 37.6 nS	
t8	BLE#, BHE# LOCK# valid delay	t8 Min + 3.5 nS	t8 Max + 24.6	CL = 75pF
t9	BLE#, BHE# LOCK# float delay	t14 Min + 5.5 nS	t14 Max + 37.6	
t10	W/R#, M/IO#, D/C#, ADS# valid delay	t10 Min + 3.5 nS	t10 Min + 24.6	CL = 75 pF
t11	W/R#, M/IO#, D/C#, ADS# float delay	t14 Min + 5.5 nS	t14 Max + 37.6	
t12	D0-D15 write data valid delay	t12 Min + 4.5 nS	t12 Max + 20.6	CL = 120 pF
t13	D0-D15 write data float delay	7.5 nS	45.6 nS	
t14	HLDA valid delay	t14 Min = 3 nS	t14 Max + 21.2 nS	
t16	NA# hold time	t16 Min + 10.6 nS		
t20	READY# hold time	t20 Min + 10.6 nS		
t21	D0-D15 read setup time	t21 Min + 8.5 nS		
t22	D0-D15 read hold time	t22 Min + 7.6 nS		
t24	HOLD hold time	t24 Min + 10.6 nS		
t25	RESET setup time	t25 Min + 2.1 nS		
t26	RESET hold time	t26 Min + 2.1 nS		
t28	NMI, INTR hold time	t28 Min + 10.6 nS		
t30	PEREQ, ERROR#, BUSY# hold time	t30 Min + 10.6 nS		

SPECIFICATIONS

Emulator Capacitance Specifications With the Target-Adapter Cable Installed

Symbol	Description	Typical (Note 1)
C_{IN}	Input Capacitance	
	CLK2	55pF
	READY#, ERROR#	35pF
	HOLD, BUSY#, PEREQ, NA#, INTR, NMI	20pF
	RESET	30pF
C_{OUT}	Output or I/O Capacitance	
	D15-D0	50pF
	A15-A1, BLE#	40pF
	A23-A16, BHE#, D/C#	30pF
	HLDA, W/R#	55pF
	ADS#, M/IO#, LOCK#	35pF

Note 1: Not tested. These specifications include the 80386SX component and all additional emulator loading.

Emulator DC Specifications Without the Bus Isolation Board Installed

Item	Description	Max.	Notes
PM-I_{CC}	Processor Module Supply Current	386SX-I_{CC} + 940 mA	
I_{IH}	Input High Leakage Current		
	A23-A1, BLE#, BHE#, D/C#, HLDA	0.02 mA	1
	D15-D0	0.06 mA	1
	ADS#, M/IO#, LOCK#, READY#, ERROR#	0.01 mA	1
	W/R#	0.03 mA	1
	CLK2	0.04 mA	1
	RESET	0.06 mA	2
I_{IL}	Input Low Leakage Current		
	A23-A1, BLE#, BHE#, D/C#	0.6 mA	1
	D15-D0	0.06 mA	1
	ADS#, M/IO#, LOCK#, READY#, ERROR#	0.01 mA	1
	W/R#	0.51 mA	1
	CLK2	0.62 mA	1
	RESET	0.6 mA	2
	HLDA	0.02 mA	1

Note 1: This specification is the DC input loading of the emulator circuitry only and does not include any 80386SX leakage current.
Note 2: This specification replaces the 80386SX specification for this signal.

SPECIFICATIONS

Emulator DC Specifications With the Bus Isolation Board Installed

Item	Description	Min.	Max.
BIB-I_{CC}	Bus Isolation Board Supply Current		PM-I_{CC} + 350 mA
V_{OL}	Output Low Voltage (I_{OL} = 48 mA)		
	A23-A1, BLE#, BHE#, D/C#, ADS#		0.5v
	D15-D0, M/IO#, LOCK#, W/R#		0.5v
	HLDA (I_{OL} = 24 mA)		0.44 v
V_{OH}	Output High Voltage (I_{OH} = 3 mA)		
	A23-A1, BLE#, BHE#, D/C#, ADS#	2.4 v	
	D15-D0, M/IO#, LOCK#, W/R#	2.4v	
	HLDA (I_{OH} = 24 mA)	3.8 v	
I_{IH}	Input High Current		
	CLK2, RESET		1.0 μA
	READY#		2.5 μA
I_{IL}	Input Low Current		
	CLK2, RESET		1.0 μA
	READY#		250 μA
I_{IO}	Output Leakage Current		
	A23-A1, BLE#, BHE#, D/C#, ADS#		±20 μA
	D15-D0, M/IO#, LOCK#, W/R#		± ±20 mA

PROCESSOR MODULE INTERFACE CONSIDERATIONS

With the processor module directly attached to the target system without using the bus isolation board, the target system must meet the following requirements:

- The user bus controller must only drive the data bus during a valid read cycle of the emulator processor or while the emulator processor is in a hold state (the emulator processor uses the data bus to communicate with the emulator hardware).
- Before driving the address bus, the user system must gain control by asserting HOLD and receiving HLDA.
- The user reset signal is disabled during the interrogation mode. It is enabled in emulation, but is delayed by 2 or 4 CLK2 cycles.
- The user system must be able to drive one additional TTL load on all signals that go to the emulation processor.

When the target system does not satisfy the first two restrictions, the bus isolation board is used to isolate the emulation processor from the target system. With the isolation board installed, the processor CLK2 is restricted to running at 20 MHz.

The processor module derives its DC power from the target system through the 80386SX socket. It requires 1400mA, including the 80386SX current. The isolation board requires an additional 350mA.

The processor must be socketed, for example using Textool 2-0100-07243-000 or AMP 821949-4 sockets.

The printed circuit board design should locate the processor socket at the physical ends of the printed circuit board traces that connect the processor to the other logic of the target system. This reduces transmission line noise. Additionally, if the target system is enclosed in a box, pin one of the processor socket should be oriented away from the target system's box opening to make connecting the target-adapter cable easier.

intel

ICE™-376 SPECIFICATIONS AND REQUIREMENTS

HOST SYSTEM REQUIREMENTS

The user supplied host system can be either an IBM PC AT or Personal System/2 Model 60. Host system requirements to run the emulator include the following:

- DOS version 4.0, or Hewlett-Packard HP9000 UNIX
- 640 Kbytes of RAM in conventional memory
- An Above board with 1 megabyte of RAM configured in expanded memory mode, EMM.SYS software version 3.2
- A 20 MB hard disk
- A serial port or the National Instruments GPIB-PCII, GPIB-PCIIA, or MC-GPIB board
- A math coprocessor if either the optional time tag board is used or if a math coprocessor resides on the target system

ELECTRICAL CHARACTERISTICS

100–120V or 220–240V selectable
50–60 Hz
2 amps (AC max) @ 120V
1 amp (AC max) @ 240V

ENVIRONMENTAL CHARACTERISTICS

Operating temperature: +10°C to +40°C (50°F to 104°F)

Operating Humidity: Maximum of 85% relative humidity, non-condensing

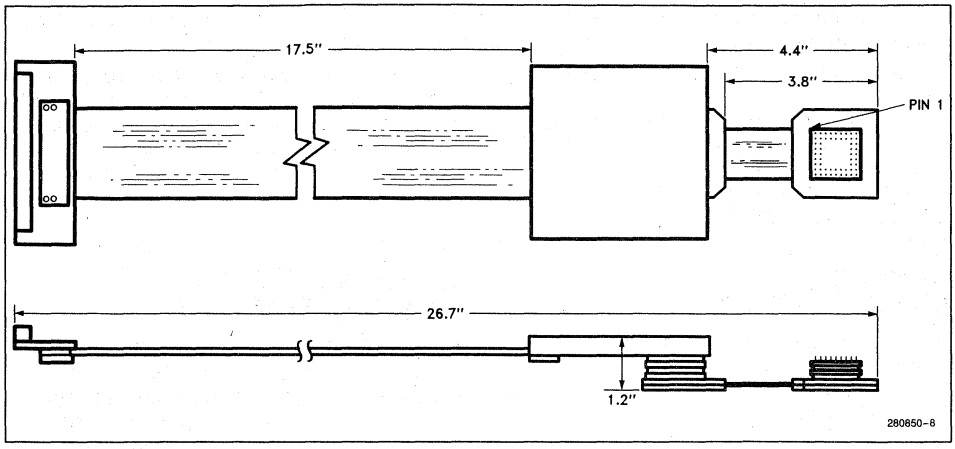
The Emulator's Physical Characteristics

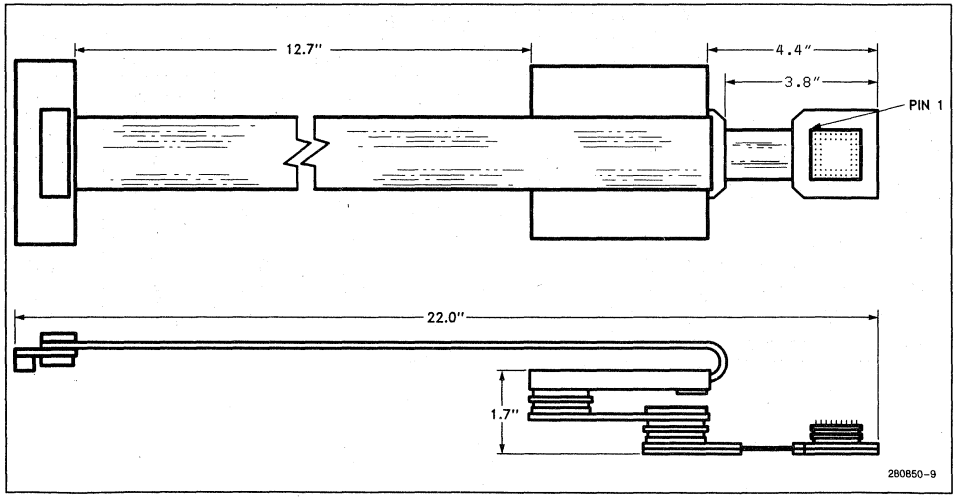
Unit	Width		Height		Length	
	inches	cm	inches	cm	inches	cm
Base Unit	13.4	34.0	4.6	11.7	11.0	27.9
Processor Module	3.8	9.7	0.7	1.8	4.4	11.2
Optional Isolation Board	3.8	9.7	0.5	1.3	4.4	11.2
Power Supply	7.7	19.6	4.1	10.4	11.0	27.9
User Cable	1.9	4.8			17.3	43.9
100-Pin Target-Adapter Cable	2.3	5.3	0.5	1.3	5.1	13.0
88-Pin Target-Adapter Cable	2.3	5.3	0.5	1.3	5.8	14.7
Serial Cable					144	366
Optional Clips Pod	3.3	8.4	0.8	2.0	6.0	15.2

8

ICE™-376 SPECIFICATIONS AND REQUIREMENTS

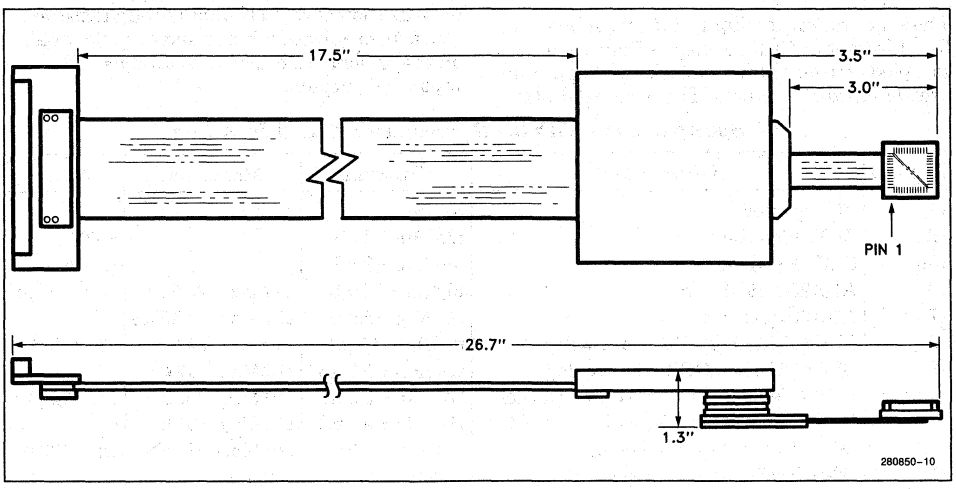
The Processor Module and Bus Isolation Board Dimensions (88 Pin PGA)

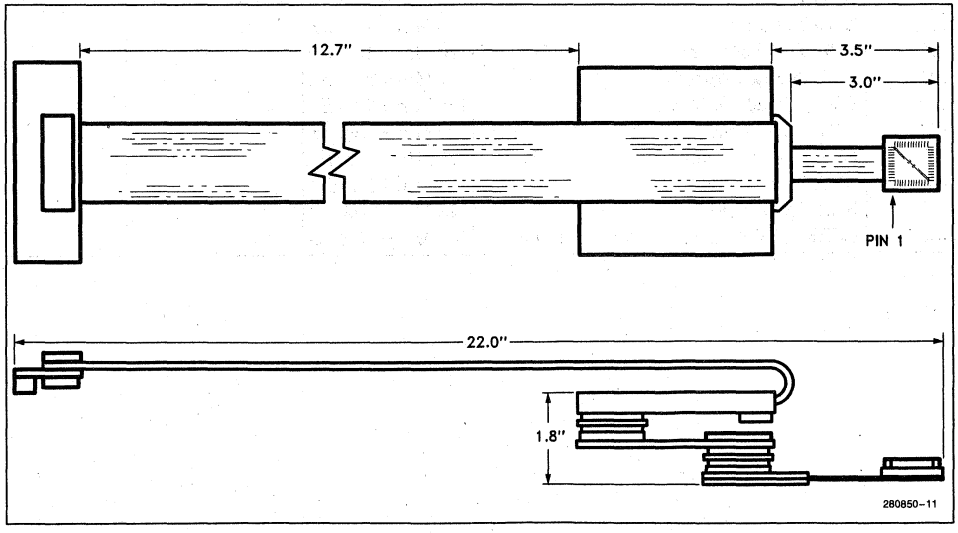




ICE™-376 SPECIFICATIONS AND REQUIREMENTS

The Processor Module and Bus Isolation Board Dimensions (100 Pin PQFP)





ICE™-376 SPECIFICATIONS AND REQUIREMENTS

ELECTRICAL SPECIFICATIONS

The synchronization input lines must be valid for at least four CLK2 cycles as they are only sampled on every other cycle. These input lines are standard TTL inputs. The synchronization output lines are driven by TTL open collector outputs that have 4.7K-ohm pull-up resistors. The synchronization input and output signals on the optional clips pod are standard TTL input and outputs.

AC Specifications With the Bus Isolation Board Installed

Symbol	Parameter	Minimum	Maximum	Notes
t1	CLK2 period	50 nS	t1 Max	
t2a	CLK2 high time	t2a Min + 2 nS		@ 2V
t3b	CLK2 low time	t3b Min + 2 nS		@ 0.8v
t6	A1-A23 valid delay	t6 Min + 3.5 nS	t6 Max + 24.6 nS	CL = 120 pF
t7	A1-A23 float delay	t14 Min + 5.5 nS	t14 Max + 37.6 nS	
t8	BLE#, BHE# LOCK# valid delay	t8 Min + 3.5 nS	t8 Max + 24.6	CL = 75pF
t9	BLE#, BHE# LOCK# float delay	t14 Min + 5.5 nS	t14 Max + 37.6	
t10	W/R#, M/IO#, D/C#, ADS# valid delay	t10 Min + 3.5 nS	t10 Min + 24.6	CL = 75 pF
t11	W/R#, M/IO#, D/C#, ADS# float delay	t14 Min + 5.5 nS	t14 Max + 37.6	
t12	D0-D15 write data valid delay	t12 Min + 4.5 nS	t12 Max + 20.6	CL = 120 pF
t13	D0-D15 write data float delay	7.5 nS	45.6 nS	
t14	HLDA valid delay	t14 Min = 3 nS	t14 Max + 21.2 nS	
t16	NA# hold time	t16 Min + 10.6 nS		
t20	READY# hold time	t20 Min + 10.6 nS		
t21	D0-D15 read setup time	t21 Min + 8.5 nS		
t22	D0-D15 read hold time	t22 Min + 7.6 nS		
t24	HOLD hold time	t24 Min + 10.6 nS		
t25	RESET setup time	t25 Min + 2.1 nS		
t26	RESET hold time	t26 Min + 2.1 nS		
t28	NMI, INTR hold time	t28 Min + 10.6 nS		
t30	PEREQ, ERROR#, BUSY# hold time	t30 Min + 10.6 nS		

SPECIFICATIONS

Emulator Capacitance Specifications With Target-Adapter Cable Installed

Symbol	Description	Typical (Note 1)
C_{IN}	Input Capacitance	
	CLK2	55pF
	READY#, ERROR#	35pF
	HOLD, BUSY#, PEREQ, NA#, INTR, NMI	20pF
	RESET	30pF
C_{OUT}	Output or I/O Capacitance	
	D15-D0	50pF
	A15-A1, BLE#	40pF
	A23-A16, BHE#, D/C#	30pF
	HLDA, W/R#	55pF
	ADS#, M/IO#, LOCK#	35pF

Note 1: Not tested. These specifications include the 80376 component and all additional emulator loading.

Emulator DC Specifications Without the Bus Isolation Board Installed

Item	Description	Max.	Notes
PM-I_{CC}	Processor Module Supply Current	376-I_{CC} + 940 mA	
I_{IH}	Input High Leakage Current		
	A23-A1, BLE#, BHE#, D/C#, HLDA	0.02 mA	1
	D15-D0	0.06 mA	1
	ADS#, M/IO#, LOCK#, READY#, ERROR#	0.01 mA	1
	W/R#	0.03 mA	1
	CLK2	0.04 mA	1
	RESET	0.06 mA	2
I_{IL}	Input Low Leakage Current		
	A23-A1, BLE#, BHE#, D/C#	0.6 mA	1
	D15-D0	0.06 mA	1
	ADS#, M/IO#, LOCK#, READY#, ERROR#	0.01 mA	1
	W/R#	0.51 mA	1
	CLK2	0.62 mA	1
	RESET	0.6 mA	2
	HLDA	0.02 mA	1

Note 1: This specification is the DC input loading of the emulator circuitry only and does not include any 80376 leakage current.
Note 2: This specification replaces the 80376 specification for this signal.

SPECIFICATIONS

Emulator DC Specifications With the Bus Isolation Board Installed

Item	Description	Min.	Max.
BIB-I_{CC}	Bus Isolation Board Supply Current		PM-I_{CC} + 350 mA
V_{OL}	Output Low Voltage (I_{OL} = 48 mA)		
	A23-A1, BLE#, BHE#, D/C#, ADS#		0.5v
	D15-D0, M/IO#, LOCK#, W/R#		0.5v
	HLDA (I_{OL} = 24 mA)		0.44 v
V_{OH}	Output High Voltage (I_{OH} = 3 mA)		
	A23-A1, BLE#, BHE#, D/C#, ADS#	2.4 v	
	D15-D0, M/IO#, LOCK#, W/R#	2.4 v	
	HLDA (I_{OH} = 24 mA)	3.8 v	
I_{IH}	Input High Current		
	CLK2, RESET		1.0 μA
	READY#		25 μA
I_{IL}	Input Low Current		
	CLK2, RESET		1.0 μA
	READY#		250 μA
I_{IO}	Output Leakage Current		
	A23-A1, BLE#, BHE#, D/C#, ADS#		±20 μA
	D15-D0, M/IO#, LOCK#, W/R#		±20 μA

PROCESSOR MODULE INTERFACE CONSIDERATIONS

With the processor module directly attached to the target system without using the bus isolation board, the target system must meet the following requirements:

- The user bus controller must only drive the data bus during a valid read cycle of the emulator processor or while the emulator processor is in a hold state (the emulator processor uses the data bus to communicate with the emulator hardware).
- Before driving the address bus, the user system must gain control by asserting HOLD and receiving HLDA.
- The user reset signal is disabled during the interrogation mode. It is enabled in emulation, but is delayed by 2 or 4 CLK2 cycles.
- The user system must be able to drive one additional TTL load on all signals that go to the emulation processor.

When the target system does not satisfy the first two restrictions, the bus isolation board is used to isolate the emulation processor from the target system. With the isolation board installed, the processor CLK2 is restricted to running at 20 MHz.

The processor module derives its DC power from the target system through the 80376 socket. It requires 1400mA, including the 80376 current. The isolation board requires an additional 350mA.

The processor must be socketed, for example using Textool 2-0100-07243-000 or AMP 821949-4 sockets.

The printed circuit board design should locate the processor socket at the physical ends of the printed circuit board traces that connect the processor to the other logic of the target system. This reduces transmission line noise. Additionally, if the target system is enclosed in a box, pin one of the processor socket should be oriented away from the target system's box opening to make connecting the target-adapter cable easier.

ORDERING INFORMATION

IN-CIRCUIT EMULATORS ORDER CODES

pICE376D	ICE376 In-circuit Emulator for 80376 component. Operates to 16 MHz. Includes control unit, power supply, 376 Processor Module with PQFP adaptor, Stand-Alone Self-Test board, bus Isolation Board, and DOS 3.x host software and interface cable.
ICE37616H	HP9000 hosted In-circuit Emulator for 80376 component. Operates to 16 MHz.
ICE386SX20D	ICE386SX 20 MHz In-circuit Emulator for 80386 SX component. Includes control unit, power supply, 386 SX Processor Module with PQFP adaptor, Stand-Alone Self-Test board, bus Isolation Board, and DOS 3.x host software and interface cable.
ICE386SX20H	HP9000 hosted In-circuit Emulator for 80386 SX component. Operates to 20 MHz.
pICE38625D	ICE386 25 MHz In-circuit Emulator for 80386 DX component. Includes control unit, power supply, 386 DX Processor Module with 132 pin PGA adaptor, Stand-Alone Self-Test board, bus Isolation Board, and DOS 3.x host software and interface cable.
ICE386DX33D	ICE386DX 33 MHz In-circuit Emulator for 80386 DX component. Includes control unit, power supply, 386 DX Processor Module with 132 pin PGA adaptor, Stand-Alone Self-Test board, bus Isolation Board, and DOS 3.x host software and interface cable.
ICE386DX25H	HP9000 hosted In-circuit Emulator for 80386 DX component. Operates to 25 MHz.
ICE386DX33H	HP9000 hosted In-circuit Emulator for the 386 DX component. This custom unit operates to 33 MHz.

intel

ORDERING INFORMATION

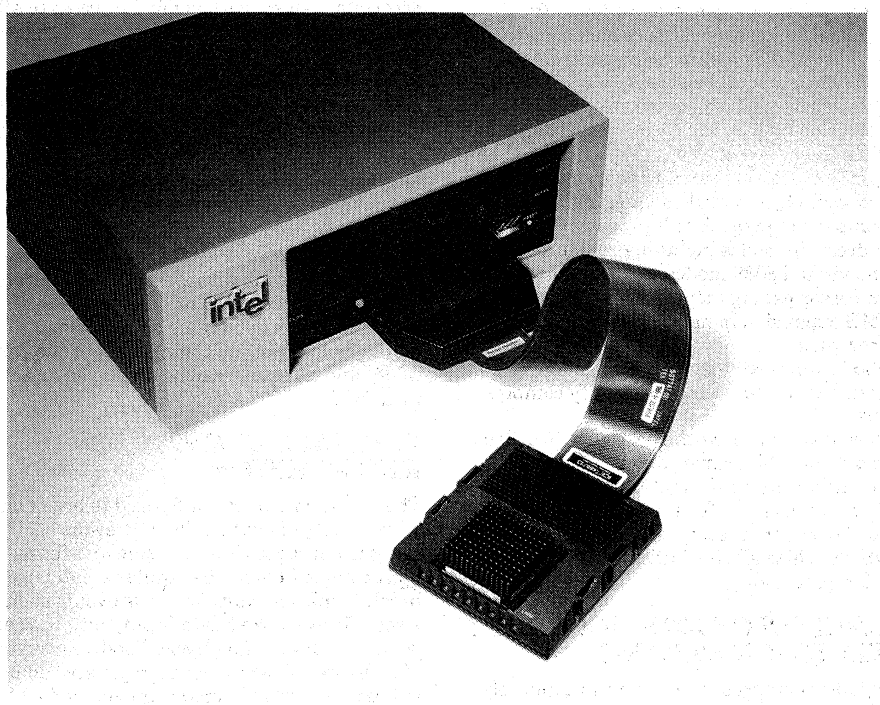
IN-CIRCUIT EMULATOR CONVERSION KITS ORDER CODES

pTOICE386SX20D	Conversion kit to adapt emulator base to support the 80386 SX component. Operates to 20 MHz. Includes ICE386SX 20 MHz Processor Module and DOS 3.x host software.
pICE376TO386D	Conversion kit to adapt ICE376 emulator to support the 80386 DX component at 25 MHz. Includes ICE386 25 MHz Processor Module and DOS 3.x host software.
pICE386SXTO376D	Conversion kit to adapt ICE386SX 16 or 20 MHz emulator to support the 80376 component. Operates to 16 MHz. Includes ICE376 emulator Processor Module and DOS 3.x host software.
pICE386SXTO386D	Conversion kit to adapt ICE386SX 16 or 20 MHz emulator to support the 80386 DX component at 25 MHz. Includes ICE386 25 MHz Processor Module and DOS 3.x host software.
pICE386TO376D	Conversion kit to adapt ICE386 25 MHz emulator to support the 80376 component. Operates to 16 MHz. Includes ICE376 emulator Processor Module and DOS 3.x host software.
TOICE386DX33D	Conversion kit to adapt emulator base to support the 80386 DX 33 MHz component. Operates to 33 MHz. Includes ICE386DX 33 MHz emulator Processor Module and DOS 3.x host software.

IN-CIRCUIT EMULATOR OPTION ORDER CODES

p88PGAADAPT	Adaptor for ICE376 emulator to support 88 pin PGA component packaging.
pICE3XXCPO	Clips Pod Option for ICE376, ICE386SX 16 or 20 MHz, ICE386 25 MHz, and ICE386DX 33 MHz emulators.
pICE3XXTTB	Time Tag Board Option for ICE376, ICE386SX 16 or 20 MHz, ICE386 25 MHz, and ICE386DX 33 MHz emulators.
DT0AB	2 MB Above Board.

INTEL i486™ IN-CIRCUIT EMULATOR



280894-1

ACCURATE AND SOPHISTICATED EMULATION FOR THE INTEL i486™ FAMILY OF MICROPROCESSORS

The Intel ICE™-486 In-Circuit Emulator is the world's leading tool for debugging software and hardware designs based on the Intel i486™ family of microprocessors. From the inventor of the microprocessor comes a development tool that allows you complete access and control over the sophisticated capabilities of the i486 microprocessor.

The ICE-486 features real-time emulation of the i486 microprocessor at speeds up to 33 MHz. Its standard high-level, symbolic debug capability saves valuable development time. The flexible breakpoint capability and 8K deep trace buffer provide the power to identify and solve the toughest hardware and software bugs.

Intel product quality and world-class technical support and service give you the time-to-market advantage in designing your i486 microprocessor based product. And your investment in development tools is protected via interchangeable probes for the 386™ family and i486 microprocessors.

8

November 1990
Order Number: 280894-001

intel

FEATURES

OVERVIEW

- Intel technology to access and modify all internal processor registers including the i486 processor on-chip cache control registers and floating point registers
- Non-intrusive, 100% accurate emulation and execution history to processor speeds of 33 MHz
- Symbolic support saves time in referencing program objects while debugging
- Complete support for all processor addressing modes including real, protected, and virtual 8086 modes with support for i486 processor paging modes
- 2MB expansion memory to debug large programs
- Maximum flexibility in break-point specification cuts time to identify complex bugs
- Deep trace buffer with the ability to collect and display 8K frames of bus and/or execution trace information
- Comprehensive software development tools optimized for creating 32-bit applications which utilize all the features of the i486 microprocessor

BENEFITS OF 100% ACCURATE EXECUTION HISTORY

The i486 microprocessor can simultaneously fetch and execute instructions. However, due to code branching, fetched instructions are not necessarily executed. Additionally, the i486 can execute instructions from the on-chip cache with no associated external bus activity. The ICE-486 emulator uses Intel technology to access the internal processor conditions that are not available to emulators which simply monitor the external buses for detection of internal events. Emulators which do not have access to the internal processor conditions cannot guarantee accurate display of what actually was executed by the microprocessor. With an Intel ICE-486, you can be certain that the emulator is displaying execution history with 100% accuracy, even when executing code from the on-chip cache memory.

OPENING THE DOOR TO PROTECTED MODE

Intel i486 emulators support protected mode operation of the i486 microprocessor. The emulator can display and modify task state segments and global, local, and interrupt descriptor tables (with symbolic access to all descriptor components such as privilege level and segment type). Emulator functions are sensitive to the operating mode of the processor, saving user setup time in debugging complex protected mode applications.

Intel i486 emulators support all aspects of protected mode addressing, including paged virtual memory. You can automatically translate virtual addresses to linear and physical addresses. Physical addresses can be translated to symbolic references to indicate the module, procedure, or data segment accessed. When debugging a memory management system, components of the page tables and directory can be displayed and modified.

FLEXIBLE EVENT RECOGNITION

The emulator can be configured to break on a wide variety of events. Flexible event recognition saves time isolating and fixing the most complex bugs. The emulator can trigger breakpoints on a variety of bus events such as a specific or masked data input, output, read, write, or a fetch at a physical address or range of addresses. On-chip debug registers can be set to break on virtual, linear, or symbolic address execution, access, or writes using the cache memory or RAM.

There are several other triggering options: You can break on a task switch, an external signal from another emulator or logic analyzer, multiple occurrences of an event, full trace buffer, halt or shutdown cycles, or interrupt acknowledge cycle. And up to four sequential event triggers can be combined with a high-level construct.

8K TRACE BUFFER FOR COLLECTING EXECUTION AND BUS ACTIVITY

Intel i486 emulators can continuously capture all or selective bus activity, and/or optionally capture execution information. The trace buffer can store up to 8,192 frames with PRE, POST, and CENTERED collection modes.

FEATURES

With the emulator halted, the trace buffer contents can be displayed in bus cycle or execution instruction formats. Dynamic trace capability allows the contents of the trace buffer to be displayed as bus cycles during full-speed emulation, a benefit when the processor cannot be halted while debugging time-critical systems. Symbolic information can be included in the trace display. When debugging high-level language programs, the callstack can also be displayed to show the current chain of procedure calls.

SYMBOLIC DEBUGGING SAVES DEVELOPMENT TIME

With the symbolic debugging capability of Intel languages and the Intel ICE-486, all data structures such as register, descriptor table, and page table contents can be examined and modified using symbolic names. Memory locations can be examined and modified using symbolic references to the source program (procedure and variable names, line numbers, or program labels). This eliminates the time-consuming use of virtual, linear, or physical address referencing used in other emulator systems. Variable type information (such as byte, word, record, or array) is also provided to make the debugging process faster and easier.

ACCESSING THE POWER

The Intel ICE-486 in-circuit emulator features a sophisticated command structure that allows you to easily access all the capabilities of the emulator.

On-line help, a syntax guide, command line editing, command history, and detailed error messages promote ease of learning and use. I/O redirection and the ability to escape to the host operating system increase versatility for the user with complex debugging needs. Creation of customized debug procedures with variables and literal definitions simplifies and automates debugging tasks used in design, test and evaluation, manufacturing test, or field service.

SYSTEM CONNECTIVITY AND CONFIGURATION

The Intel i486 emulator can be combined with a variety of lab instruments to extend the capability of the tool. I/O sync lines allow emulator event control and synchronization with external tools such as logic analyzers, scopes, or another emulator.

INCLUDED OPTIONAL USE EQUIPMENT

The Relocatable Expansion Memory (REM) board included with the ICE-486 system allows you to map 2 MB of memory for developing large applications before prototype target system memory is completely functional. Also, an optional isolation board is provided with the emulator. It buffers signals to the emulator, protecting the emulator from potential damage caused by an untested prototype target system. The REM board can be used in conjunction with the isolation board to overlay EPROMs. This technique avoids the slow process of programming new EPROMs each time a new version of EPROM software is compiled.

A stand-alone/self-test board is also provided with the emulator. The stand-alone/self-test board, in conjunction with the REM board, allows execution and debugging of code to begin before target system availability. It also allows execution of the emulator confidence tests so you always know with certainty that the emulator is functioning properly.

EMULATOR OPTIONS

An optional time tag board synchronizes multiple Intel i486 emulators and adds 20-nanosecond resolution time stamp information to the trace buffer.

An optional clips pod allows eight data lines to be captured in the trace buffer and displayed in the CYCLES format.

SOFTWARE COMPLETES THE SYSTEM

Intel provides comprehensive software development tools which work together with the ICE-486 emulator for the most complete development environment available from a single vendor. C, PL/M, and Fortran compilers are available in addition to a Macroassembler. A builder and binder, available for configuring and linking software modules, greatly simplify configuration of code modules for protected mode systems. The DB-386/486 source-level software debugger with its powerful windowed interface completes the picture.

To further reduce the effort necessary to integrate software into the final target configuration, Intel tools produce ROM code directly, saving you the time and headaches frequently encountered using converter utilities from other vendors.

intel

FEATURES

WORLD-CLASS, WORLDWIDE SERVICES

Intel i486 development tools are supplemented by a full array of support services. Seminars, classes and workshops, on-site consulting, field application engineering expertise, telephone hotline support, and software and hardware maintenance are just a few of the services which are available from Intel to insure your design requirements are met on time and with minimal problems. Only Intel gives you one source to call for complete development tool support for your i486 design.

ICE™-486 33 MHz SPECIFICATIONS AND REQUIREMENTS

HOST SYSTEM REQUIREMENTS

The user supplied host system can be either an IBM® PC/AT®, Personal System/2® Model 60, or Model 80. Host system requirements to run the emulator include the following:

- DOS version 3.3
- 640K bytes of RAM in conventional memory
- An Above™ board with 1 megabyte of RAM configured in expanded memory mode, EMM.SYS software version 3.2, or
- One megabyte of RAM configured as expanded memory using 386MAX
- A 20MB hard disk
- A serial port or the National Instruments GPIB-PCII™, GPIB-PCIIA™, or MC-GPIB™ board
- A math coprocessor is required in the host system if either the optional time tag board is used or if the on-chip floating point unit is utilized by the target system

ELECTRICAL CHARACTERISTICS

100-120V or 220-240V selectable
50-60 Hz
2 amps (AC max) @ 120V
1 amp (AC max) @ 240V

ENVIRONMENTAL CHARACTERISTICS

Operating temperature: +10°C to +40°C (50°F to 104°F)

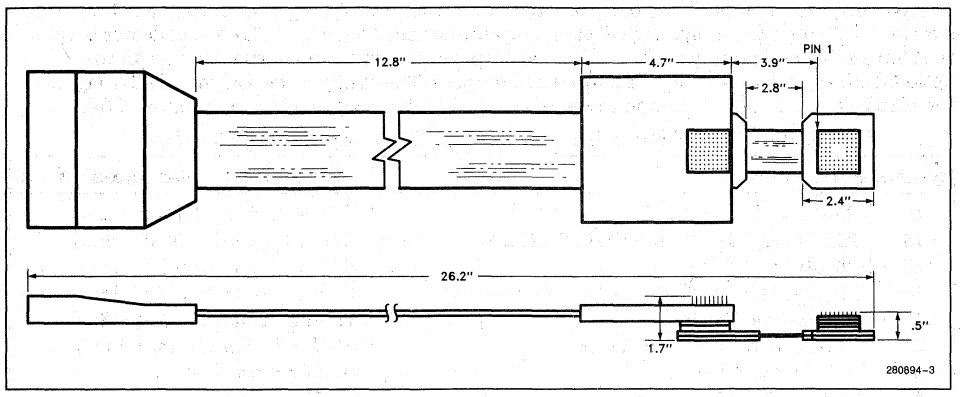
Operating Humidity: Maximum of 85% relative humidity, non-condensing

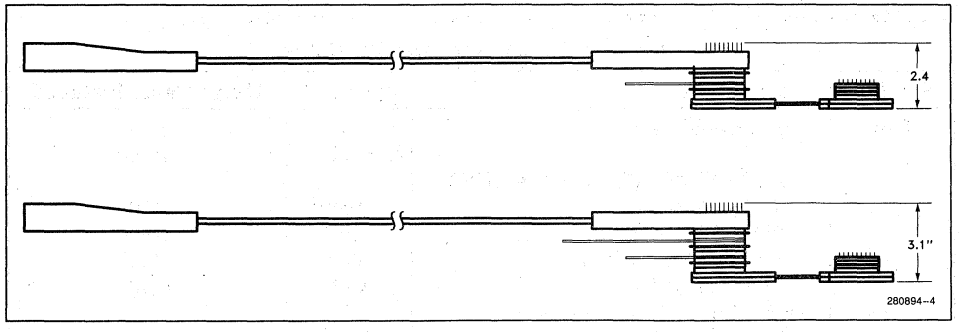
The Emulator's Physical Characteristics

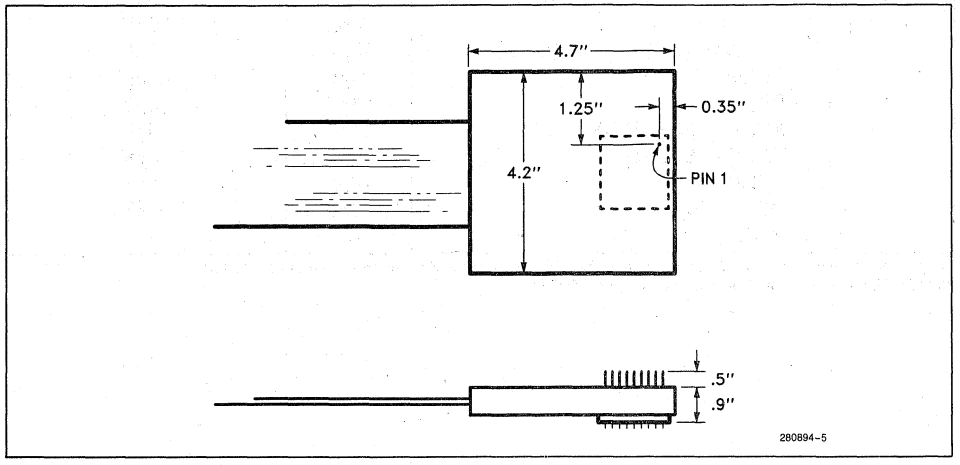
Unit	Width		Height		Length	
	inches	cm	inches	cm	inches	cm
Base Unit	13.4	34.0	4.6	11.7	11.0	27.9
Processor Module	4.2	10.7	1.4	3.6	4.7	11.9
Optional Isolation Board	3.5	8.9	0.7	1.8	3.5	8.9
Power Supply	7.7	19.6	4.1	10.4	11.0	27.9
User Cable	1.9	4.8			15.8	40.1
Hinge Cable	2.6	6.6	0.5	1.3	7.7	19.6
Relocatable Expansion Memory Board	3.5	8.9	0.7	1.8	6.0	15.2
Serial Cable					144	366
Optional Clips Pod	3.3	8.4	0.8	2.0	6.0	15.2

ICE™-486 33 MHz SPECIFICATIONS AND REQUIREMENTS

THE PROCESSOR MODULE AND OPTIONAL BOARD DIMENSIONS







8

ICE™-486 33 MHz SPECIFICATIONS AND REQUIREMENTS

ELECTRICAL SPECIFICATIONS

The synchronization input lines must be valid for at least two CLK cycles. These input lines are standard TTL inputs. The synchronization output lines are driven by TTL open collector outputs that have 4.7K-ohm pull-up resistors. The synchronization input and output signals on the optional clips pod are standard TTL input and outputs. The emulator delays the RESET signal to the i486 by a maximum of 8ns and the A20M# and FLUSH# signals by a maximum of 5ns.

Emulator AC Specifications with the Isolation Board Installed

Symbol	Parameter	Minimum	Maximum	Notes
t6	A2-A31 valid delay	t6 Min + 1.5ns	t6 Max + 5ns	
t6	BE0-3#, M/IO#, W/R#, ADS#, HLDA valid delay	t6 Min + 2.5ns	t6 Max + 8ns	
t6	A2-A31 valid delay	t6 Min + 1.5ns	t6 Max + 11ns	1
t6	BE0-3#, M/IO#, W/R#, ADS# valid delay	t6 Min + 2.5ns	t6 Max + 14ns	1
t8a	BLAST# valid delay	t8a Min + 2.5ns	t8a Max + 8ns	
t10	D0-D31 write data valid delay	t10 Min + 1.5ns	t10 Max + 5ns	
t16	RDY# setup time	t16 Min + 5ns		
t22	A4-A31, D0-D31 input set up time	t22 Min + 5ns		

Note 1: Use these specifications for any bus cycle that begins on the same clock that HLDA is de-asserted.

Emulator Capacitance Specifications

Symbol	Description	Typical	Hinge Cable Installed
C_{IN}	**Input Capacitance:**		
	CLK	55pF	70pF
	A20M#, AHOLD, FLUSH#, HOLD,		
	IGNNE#, INTR	20pF	30pF
	BS8#, BS16#, EADS#, KEN#	25pF	40pF
	BRDY#, NMI, RDY#	30pF	40pF
	RESET, BOFF#	35pF	50pF
C_{OUT}	**Output or I/O Capacitance:**		
	D0 - D31, BLAST#, D/C#, LOCK#,	30pF	50pF
	PLOCK#	25pF	50pF
	A2 - A31, ADS#	25pF	45pF
	HLDA, M/IO#	25pF	40pF
	PWT	35pF	55pF
	BE0# - BE3#, PCD	15pF	35pF
	BREQ#, PCHK#	15pF	30pF
	FERR#	20pF	35pF
	DP0 - DP3	40pF	60pF
	W/R#		

Note 1: Not tested. These specifications include the i486 component and all additional emulator loading.
Note 2: The hinge cable adds a propagation delay of 0.5 ns.

ICE™-486 33 MHz SPECIFICATIONS AND REQUIREMENTS

Emulator DC Specifications without the Isolation Board Installed

Item	Description	Maximum	Notes
PM-I_{CC}	Processor Module Supply Current	486I_{CC} + 1.5A	
I_{IH}	Input High Leakage Current		
	D0-31	15uA	1
	A2-31, BE#0-3, PWT	5uA	1
	W/R#, D/C#, M/IO#, LOCK#, PLOCK#	5uA	1
	BLAST#, HLDA	5uA	1
	BS16#, BS8#, EADS#, KEN#, NMI	5uA	1
	BOFF#, RDY#, BRDY#	25uA	1
	PCD	30uA	1
	CLK	15uA	1
	RESET	30uA	2
	A20M#, FLUSH#	5uA	2
I_{IL}	Input Low Leakage Current		
	D0-31	15uA	1
	A2-31, BE#0-3, PWT	5uA	1
	W/R#, D/C#, M/IO#, LOCK#, PLOCK#	5uA	1
	BLAST#, HLDA	5uA	1
	BS16#, BS8#, EADS#, KEN#, NMI	5uA	1
	BOFF#, RDY#, BRDY#	250uA	1
	PCD	255uA	1
	CLK	15uA	1
	RESET	255uA	2
	A20M#, FLUSH#	5uA	2

Note 1: This specification is the DC loading of the emulator circuitry only and does not include any i486 leakage current.
Note 2: This specification replaces the i486 specification for this signal.

Emulator DC Specifications with the Isolation Board Installed

Item	Description		Minimum	Maximum	Notes
V_{OH}	Output High Voltage				
	A2-A31, D0-D31	(I_{OH} = 15 mA)	2.4V		2
	BE0-3#, M/IO#,	(I_{OH} = 3 mA)	2.4V		
	W/R#, ADS#, BLAST#	(I_{OH} 3 mA)	2.4V		
	HLDA	(I_{OH} 3.2 mA)	2.4V		
V_{OL}	Output Low Voltage				
	A2-A31, D0-D31	(I_{OL} = 64 mA)		0.55V	2
	BE0-3#, M/IO#,	(I_{OL} = 64 mA)		0.55V	
	W/R#, ADS#, BLAST#	(I_{OL} = 64 mA)		0.55V	
	HLDA	(I_{OL} = 24 mA)		0.5V	
I_{LI}	Input Leakage Current				
	A2-A31, D0-D31			±15uA	2
I_{IH}	Input High Current				
	CLK, RESET, BRDY#, BOFF#, AHOLD			25uA	1
	RDY#			30uA	2
I_{IL}	Input Low Current				
	CLK, RESET, BRDY#, BOFF#, AHOLD			250uA	1
	RDY#			255uA	2

Note 1: This specification is for the Isolation Board only and does not include any processor module loading.
Note 2: These specifications replace the i486 specifications for this signal.

ICE™-486 33 MHz SPECIFICATIONS AND REQUIREMENTS

PROCESSOR MODULE INTERFACE CONSIDERATIONS

With the processor module directly attached to the target system without using the optional isolation board, the target system must meet the following requirements:

- The bus controller must only enable data transceivers onto the bus during valid read cycles of the 486 CPU or while another bus master has gained access to the bus through the use of HOLD/HLDA or BOFF#.
- Before another bus master drives the local processor address bus, the other bus master must gain access to the address bus through the use of HOLD/HLDA, AHOLD or BOFF#.
- The user system must be able to drive one additional CMOS load (approximately 25pF) on all signals that go to the emulation processor.

If the target system does not satisfy the restrictions, the optional isolation board should be used to isolate the emulation processor from the target system. To guarantee proper operation with the optional isolation board, the clock period should be increased by the round trip buffer delay (10 ns) unless the target system design already has enough timing margin.

The processor module derives its DC power from the target system through the 486 CPU socket. It requires 2200mA, including the i486 current. The optional isolation board requires an additional 500mA. The REM board requires an additional 2100mA.

The processor must be socketed. The printed circuit board design should locate the processor socket at the physical ends of the printed circuit board traces that connect the processor to the other logic of the target system. This reduces transmission line noise. If the target system is enclosed in a box, orient pin one of the processor socket to simplify connecting the ICB. This makes connecting the hinge cable easier. The ICE-486 emulator hinge cable adds an additional 15pF of capacitive loading and approximately 0.5ns of propagation delay to each 486 CPU signal.

Pins specified as N.C. in the 486 CPU pin description must be left unconnected. Connection of any of these pins to power, ground or any other signal may cause the processor or the ICE-486 emulator to malfunction.

ORDERING INFORMATION

ICE™-486 IN-CIRCUIT EMULATOR ORDER CODES

ICE48633D	ICE-486 In-circuit emulator for 80486 component. Operates to 33 MHz. Includes control unit, power supply, 80486 Processor Module, Stand-Alone/Self-Test Board, Optional Bus Isolation Board, Relocatable Expansion Memory Board, host software and cables.

ICE™-486 IN-CIRCUIT EMULATOR CONVERSION KIT ORDER CODES

BASECONV386	Conversion kit to upgrade the 386 family emulator base to support the 80486 processor module.
TOICE48633D	Conversion kit to adapt the above upgraded base to support the 80486 component. Includes ICE486 33 MHz Processor Module, Stand-Alone/Self-Test Board, Optional Bus Isolation Board, Relocatable Expansion Memory Board, and host software.

ICE™-486 IN-CIRCUIT EMULATOR OPTION ORDER CODES

DT0AB	2Mb Intel Above™ Board.
ICE3XXCP0	Clips Pod Option for ICE-486.
ICE3XXTTB	Time Tag Board Option for ICE-486.

DOMESTIC SALES OFFICES

ALABAMA

Intel Corp.
5015 Bradford Dr., #2
Huntsville 35805
Tel: (205) 830-4010
FAX: (205) 837-2640

ARIZONA

†Intel Corp.
410 North 44th Street
Suite 500
Phoenix 85008
Tel: (602) 231-0386
FAX: (602) 244-0446

Intel Corp.
7225 N. Mona Lisa Rd.
Suite 215
Tucson 85741
Tel: (602) 544-0227
FAX: (602) 544-0232

CALIFORNIA

†Intel Corp.
21515 Vanowen Street
Suite 116
Canoga Park 91303
Tel: (818) 704-8500
FAX: (818) 340-1144

†Intel Corp.
300 N. Continental Blvd.
Suite 100
El Segundo 90245
Tel: (213) 640-6040
FAX: (213) 640-7133

Intel Corp.
1 Sierra Gate Plaza
Suite 280C
Roseville 95678
Tel: (916) 782-8086
FAX: (916) 782-8153

†Intel Corp.
9665 Chesapeake Dr.
Suite 325
San Diego 92123
Tel: (619) 292-8086
FAX: (619) 292-0628

†Intel Corp.*
400 N. Tustin Avenue
Suite 450
Santa Ana 92705
Tel: (714) 835-9642
TWX: 910-595-1114
FAX: (714) 541-9157

†Intel Corp.*
San Tomas 4
2700 San Tomas Expressway
2nd Floor
Santa Clara 95051
Tel: (408) 986-8086
TWX: 910-338-0255
FAX: (408) 727-2620

COLORADO

Intel Corp.
4445 Northpark Drive
Suite 100
Colorado Springs 80907
Tel: (719) 594-6622
FAX: (303) 594-0720

†Intel Corp.*
600 S. Cherry St.
Suite 700
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289
FAX: (303) 322-8670

CONNECTICUT

†Intel Corp.
301 Lee Farm Corporate Park
83 Wooster Heights Rd.
Danbury 06810
Tel: (203) 748-3130
FAX: (203) 794-0339

FLORIDA

†Intel Corp.
800 Fairway Drive
Suite 160
Deerfield Beach 33441
Tel: (305) 421-0506
FAX: (305) 421-2444

†Intel Corp.
5850 T.G. Lee Blvd.
Suite 340
Orlando 32822
Tel: (407) 240-8000
FAX: (407) 240-8097

Intel Corp.
11300 4th Street North
Suite 170
St. Petersburg 33716
Tel: (813) 577-2413
FAX: (813) 578-1607

GEORGIA

†Intel Corp.
20 Technology Parkway
Suite 150
Norcross 30092
Tel: (404) 449-0541
FAX: (404) 605-9762

ILLINOIS

†Intel Corp.*
Woodfield Corp. Center III
300 N. Martingale Road
Suite 400
Schaumburg 60173
Tel: (708) 605-8031
FAX: (708) 706-9762

INDIANA

†Intel Corp.
8910 Purdue Road
Suite 350
Indianapolis 46268
Tel: (317) 875-0623
FAX: (317) 875-8938

IOWA

Intel Corp.
1930 St. Andrews Drive N.E.
2nd Floor
Cedar Rapids 52402
Tel: (319) 393-5510

KANSAS

†Intel Corp.
10985 Cody St.
Suite 140
Overland Park 66210
Tel: (913) 345-2727
FAX: (913) 345-2076

MARYLAND

†Intel Corp.*
10010 Junction Dr.
Suite 200
Annapolis Junction 20701
Tel: (301) 206-2860
FAX: (301) 206-3677
(301) 206-3678

MASSACHUSETTS

†Intel Corp.*
Westford Corp. Center
3 Carlisle Road
2nd Floor
Westford 01886
Tel: (508) 692-0960
TWX: 710-343-6333
FAX: (508) 692-7867

MICHIGAN

†Intel Corp.
7071 Orchard Lake Road
Suite 100
West Bloomfield 48322
Tel: (313) 851-8096
FAX: (313) 851-8770

MINNESOTA

†Intel Corp.
3500 W. 80th St.
Suite 360
Bloomington 55431
Tel: (612) 835-6722
TWX: 910-576-2867
FAX: (612) 831-6497

MISSOURI

†Intel Corp.
4203 Earth City Expressway
Suite 131
Earth City 63045
Tel: (314) 291-1990
FAX: (314) 291-4341

NEW JERSEY

†Intel Corp.*
Lincroft Office Center
125 Half Mile Road
Red Bank 07701
Tel: (908) 747-2233
FAX: (908) 747-0983

Intel Corp.
280 Corporate Center
75 Livingston Avenue
First Floor
Roseland 07068
Tel: (201) 740-0111
FAX: (201) 740-0626

NEW YORK

Intel Corp.*
850 Crosskeys Office Park
Fairport 14450
Tel: (716) 425-2750
TWX: 510-253-7391
FAX: (716) 223-2561

†Intel Corp.*
2950 Express Dr., South
Suite 130
Islandia 11722
Tel: (516) 231-3300
TWX: 510-227-6236
FAX: (516) 348-7939

†Intel Corp.
300 Westage Business Center
Suite 230
Fishkill 12524
Tel: (914) 897-3860
FAX: (914) 897-3125

Intel Corp.
Seventeen State Street
14th Floor
New York 10004
Tel: (212) 248-8086
FAX: (212) 248-0888

NORTH CAROLINA

†Intel Corp.
5800 Executive Center Dr.
Suite 105
Charlotte 28212
Tel: (704) 568-8966
FAX: (704) 535-2236

†Intel Corp.
5540 Centerview Dr.
Suite 215
Raleigh 27606
Tel: (919) 851-9537
FAX: (919) 851-8974

OHIO

†Intel Corp.*
3401 Park Center Drive
Suite 220
Dayton 45414
Tel: (513) 890-5350
TWX: 810-450-2528
FAX: (513) 890-8658

†Intel Corp.*
25700 Science Park Dr.
Suite 100
Beachwood 44122
Tel: (216) 464-2736
TWX: 810-427-9298
FAX: (804) 282-0673

OKLAHOMA

Intel Corp.
6801 N. Broadway
Suite 115
Oklahoma City 73162
Tel: (405) 848-8086
FAX: (405) 840-9819

OREGON

†Intel Corp.
15254 N.W. Greenbrier Pkwy.
Building B
Beaverton 97006
Tel: (503) 645-8051
TWX: 910-467-8741
FAX: (503) 645-8181

PENNSYLVANIA

†Intel Corp.*
925 Harvest Drive
Suite 200
Blue Bell 19422
Tel: (215) 641-1000
FAX: (215) 641-0785

†Intel Corp.*
400 Penn Center Blvd.
Suite 610
Pittsburgh 15235
Tel: (412) 823-4970
FAX: (412) 829-7578

PUERTO RICO

†Intel Corp.
South Industrial Park
P.O. Box 910
Las Piedras 00671
Tel: (809) 733-8616

TEXAS

Intel Corp.
8911 N. Capital of Texas Hwy.
Suite 4230
Austin 78759
Tel: (512) 794-8086
FAX: (512) 338-9335

†Intel Corp.*
12000 Ford Road
Suite 400
Dallas 75234
Tel: (214) 241-8087
FAX: (214) 484-1180

†Intel Corp.*
7322 S.W. Freeway
Suite 1490
Houston 77074
Tel: (713) 988-8086
TWX: 910-881-2490
FAX: (713) 988-3660

UTAH

†Intel Corp.
428 East 6400 South
Suite 104
Murray 84107
Tel: (801) 263-8051
FAX: (801) 268-1457

VIRGINIA

†Intel Corp.
9030 Stony Point Pkwy.
Suite 360
Richmond 23235
Tel: (804) 330-9393
FAX: (804) 330-3019

WASHINGTON

†Intel Corp.
155 108th Avenue N.E.
Suite 386
Bellevue 98004
Tel: (206) 453-8086
TWX: 910-443-3002
FAX: (206) 451-9556

Intel Corp.
408 N. Mullan Road
Suite 102
Spokane 99206
Tel: (509) 928-8086
FAX: (509) 928-9467

WISCONSIN

Intel Corp.
330 S. Executive Dr.
Suite 102
Brookfield 53005
Tel: (414) 784-8087
FAX: (414) 796-2115

CANADA

BRITISH COLUMBIA

Intel Semiconductor of
Canada, Ltd.
4585 Canada Way
Suite 202
Burnaby V5G 4L6
Tel: (604) 298-0387
FAX: (604) 298-8234

ONTARIO

†Intel Semiconductor of
Canada, Ltd.
2650 Queensview Drive
Suite 250
Ottawa K2B 8H6
Tel: (613) 829-9714
FAX: (613) 820-5936

†Intel Semiconductor of
Canada, Ltd.
190 Attwell Drive
Suite 500
Rexdale M9W 6H8
Tel: (416) 675-2105
FAX: (416) 675-2438

QUEBEC

†Intel Semiconductor of
Canada, Ltd.
1 Rue Holiday
Suite 115
Tour East
Pt. Claire H9R 5N3
Tel: (514) 694-9130
FAX: 514-694-0064

†Sales and Service Office
*Field Application Location

DOMESTIC DISTRIBUTORS

ALABAMA

Arrow Electronics, Inc.
1015 Henderson Road
Huntsville 35805
Tel: (205) 837-6955
FAX: 205-751-1581

Hamilton/Avnet Computer
4930 I Corporate Drive
Huntsville 35805

Hamilton/Avnet Electronics
4940 Research Drive
Huntsville 35805
Tel: (205) 837-7210
FAX: 205-721-0356

MTI Systems Sales
4950 Corporate Drive
Suite 120
Huntsville 35806
Tel: (205) 830-9526
FAX: (205) 830-9557

Pioneer/Technologies Group, Inc.
4825 University Square
Huntsville 35805
Tel: (205) 837-9300
FAX: 205-837-9358

ALASKA

Hamilton/Avnet Computer
1400 W. Benson Blvd., Suite 400
Anchorage 99503

ARIZONA

†Arrow Electronics, Inc.
4134 E. Wood Street
Phoenix 85040
Tel: (602) 437-0750
TWX: 910-951-1550

Hamilton/Avnet Computer
30 South McKemy Avenue
Chandler 85226

Hamilton/Avnet Computer
90 South McKemy Road
Chandler 85226

†Hamilton/Avnet Electronics
505 S. Madison Drive
Tempe 85281
Tel: (602) 231-5140
TWX: 910-950-0077

Hamilton/Avnet Electronics
30 South McKemy
Chandler 85226
Tel: (602) 961-6669
FAX: 602-961-4073

Wyle Distribution Group
4141 E. Raymond
Phoenix 85040
Tel: (602) 249-2232
TWX: 910-371-2871

CALIFORNIA

Arrow Commercial System Group
1502 Crocker Avenue
Hayward 94544
Tel: (415) 489-5371
FAX: (415) 489-9393

Arrow Commercial System Group
14242 Chambers Road
Tustin 92680
Tel: (714) 544-0200
FAX: (714) 731-8438

†Arrow Electronics, Inc.
19748 Dearborn Street
Chatsworth 91311
Tel: (213) 701-7500
TWX: 910-493-2086

†Arrow Electronics, Inc.
9511 Ridgehaven Court
San Diego 92123
Tel: (619) 565-4800
FAX: 619-279-8062

†Arrow Electronics, Inc.
521 Weddell Drive
Sunnyvale 94086
Tel: (408) 745-6600
TWX: 910-339-9371

†Arrow Electronics, Inc.
2961 Dow Avenue
Tustin 92680
Tel: (714) 838-5422
TWX: 910-595-2860

Hamilton/Avnet Computer
3170 Pullman Street
Costa Mesa 92626

Hamilton/Avnet Computer
1361B West 190th Street
Gardena 90248

Hamilton/Avnet Computer
4103 Northgate Blvd.
Sacramento 95834

Hamilton/Avnet Computer
4545 Viewridge Avenue
San Diego 92123

Hamilton/Avnet Computer
1175 Bordeaux Drive
Sunnyvale 94089

Hamilton/Avnet Electronics
21150 Califa Street
Woodland Hills 91367

†Hamilton/Avnet Electronics
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4150
TWX: 910-595-2638

†Hamilton/Avnet Electronics
1175 Bordeaux Drive
Sunnyvale 94086
Tel: (408) 743-3300
TWX: 910-339-9332

†Hamilton/Avnet Electronics
4545 Ridgeview Avenue
San Diego 92123
Tel: (619) 571-7500
TWX: 910-595-2638

†Hamilton/Avnet Electronics
21150 Califa St.
Woodland Hills 91376
Tel: (818) 594-0404
FAX: 818-594-8233

†Hamilton/Avnet Electronics
10950 W. Washington Blvd.
Culver City 20230
Tel: (213) 558-2458
TWX: 910-340-6364

†Hamilton/Avnet Electronics
1361B West 190th Street
Gardena 90248
Tel: (213) 217-6700
TWX: 910-340-6364

†Hamilton/Avnet Electronics
4103 Northgate Blvd.
Sacramento 95834
Tel: (916) 920-3150

Pioneer/Technologies Group, Inc.
134 Rio Robles
San Jose 95134
Tel: (408) 954-9100
FAX: 408-954-9113

Wyle Distribution Group
124 Maryland Street
El Segundo 90254
Tel: (213) 322-8100

Wyle Distribution Group
7431 Chapman Ave.
Garden Grove 92641
Tel: (714) 891-1717
FAX: 714-891-1621

†Wyle Distribution Group
2951 Sunrise Blvd., Suite 175
Rancho Cordova 95742
Tel: (916) 638-5282

†Wyle Distribution Group
9525 Chesapeake Drive
San Diego 92123
Tel: (619) 565-9171
TWX: 910-335-1590

†Wyle Distribution Group
3000 Bowers Avenue
Santa Clara 95051
Tel: (408) 727-2500
TWX: 408-988-2747

†Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel: (714) 863-9953
TWX: 910-371-7127

†Wyle Distribution Group
26677 W. Agoura Rd.
Calabasas 91302
Tel: (818) 880-9000
TWX: 372-0232

COLORADO

Arrow Electronics, Inc.
7060 South Tucson Way
Englewood 80112
Tel: (303) 790-4444

Hamilton/Avnet Computer
9605 Maroon Circle, Ste. 200
Engelwood 80112

†Hamilton/Avnet Electronics
9605 Maroon Circle
Suite 200
Englewood 80112
Tel: (303) 799-0663
TWX: 910-935-0787

†Wyle Distribution Group
451 E. 124th Avenue
Thornton 80241
Tel: (303) 457-9953
TWX: 910-936-0770

CONNECTICUT

†Arrow Electronics, Inc.
12 Beaumont Road
Wallingford 06492
Tel: (203) 265-7741
TWX: 710-476-0162

Hamilton/Avnet Computer
Commerce Industrial Park
Commerce Drive
Danbury 06810

†Hamilton/Avnet Electronics
Commerce Industrial Park
Commerce Drive
Danbury 06810
Tel: (203) 797-2800
TWX: 710-456-9974

†Pioneer/Standard Electronics
112 Main Street
Norwalk 06851
Tel: (203) 853-1515
FAX: 203-838-9901

FLORIDA

†Arrow Electronics, Inc.
400 Fairway Drive
Suite 102
Deerfield Beach 33441
Tel: (305) 429-8200
FAX: 305-428-3991

†Arrow Electronics, Inc.
37 Skyline Drive
Suite 3101
Lake Marv 32746
Tel: (407) 323-0252
FAX: 407-323-3189

Hamilton/Avnet Computer
6801 N.W. 15th Way
Ft. Lauderdale 33309

Hamilton/Avnet Computer
3247 Spring Forest Road
St. Petersburg 33702

†Hamilton/Avnet Electronics
6801 N.W. 15th Way
Ft. Lauderdale 33309
Tel: (305) 971-2900
FAX: 305-971-5420

†Hamilton/Avnet Electronics
3197 Tech Drive North
St. Petersburg 33702
Tel: (813) 573-3930
FAX: 813-572-4329

†Hamilton/Avnet Electronics
6947 University Boulevard
Winter Park 32792
Tel: (407) 628-3888
FAX: 407-678-1878

†Pioneer/Technologies Group, Inc.
337 Northlake Blvd., Suite 1000
Alta Monte Springs 32701
Tel: (407) 834-9090
FAX: 407-834-0865

Pioneer/Technologies Group, Inc.
674 S. Military Trail
Deerfield Beach 33442
Tel: (305) 428-8877
FAX: 305-481-2950

GEORGIA

Arrow Commercial System Group
3400 C. Corporate Way
Deluth 30139
Tel: (404) 623-8825
FAX: (404) 623-8802

†Arrow Electronics, Inc.
4250 E. Rivergreen Parkway
Deluth 30136
Tel: (404) 497-1300
TWX: 810-766-0439

Hamilton/Avnet Computer
5825 D. Peachtree Corners E.
Norcross 30092

†Hamilton/Avnet Electronics
5825 D Peachtree Corners
Norcross 30092
Tel: (404) 447-7500
TWX: 810-766-0432

Pioneer/Technologies Group, Inc.
3100 F Northwoods Place
Norcross 30071
Tel: (404) 448-1711
FAX: 404-446-8270

ILLINOIS

†Arrow Electronics, Inc.
1140 W. Thorndale
Itasca 60143
Tel: (708) 250-0500
TWX: 708-250-0916

Hamilton/Avnet Computer
1130 Thorndale Avenue
Bensenville 60106

†Hamilton/Avnet Electronics
1130 Thorndale Avenue
Bensenville 60106
Tel: (708) 860-7780
TWX: 708-860-8530

MTI Systems Sales
1100 W. Thorndale
Itasca 60143
Tel: (708) 773-2300

†Pioneer/Standard Electronics
2171 Executive Dr., Suite 200
Addison 60101
Tel: (708) 495-9680
FAX: 708-495-9831

INDIANA

†Arrow Electronics, Inc.
7108 Lakeview Parkway West Drive
Indianapolis 46268
Tel: (317) 299-2071
FAX: 317-299-0255

Hamilton/Avnet Computer
485 Gradle Drive
Carmel 46032

Hamilton/Avnet Electronics
485 Gradle Drive
Carmel 46032
Tel: (317) 844-9333
FAX: 317-844-5921

†Pioneer/Standard Electronics
9350 Priority Way
West Drive
Indianapolis 46250
Tel: (317) 573-0880
FAX: 317-573-0979

†Certified Technical Distributor

DOMESTIC DISTRIBUTORS (Contd.)

IOWA

Hamilton/Avnet Computer
915 33rd Avenue SW
Cedar Rapids 52404

Hamilton/Avnet Electronics
915 33rd Avenue, S.W.
Cedar Rapids 52404
Tel: (319) 362-4757

KANSAS

Arrow Electronics, Inc.
8208 Melrose Dr., Suite 210
Lenexa 66214
Tel: (913) 541-9542
FAX: 913-541-0328

Hamilton/Avnet Computer
15313 W. 95th Street
Lenexa 61219

†Hamilton/Avnet Electronics
15313 W. 95th
Overland Park 66215
Tel: (913) 888-8900
FAX: 913-541-7951

KENTUCKY

Hamilton/Avnet Electronics
805 A. Newtown Circle
Lexington 40511
Tel: (606) 259-1475

MARYLAND

†Arrow Electronics, Inc.
8300 Guilford Drive
Suite H, River Center
Columbia 21046
Tel: (301) 995-6002
FAX: 301-381-3854

Hamilton/Avnet Computer
6822 Oak Hall Lane
Columbia 21045

†Hamilton/Avnet Electronics
6822 Oak Hall Lane
Columbia 21045
Tel: (301) 995-3500
FAX: 301-995-3593

†Mesa Technology Corp.
9720 Patuxent Woods Dr.
Columbia 21046
Tel: (301) 290-8150
FAX: 301-290-6474

†Pioneer/Technologies Group, Inc.
9100 Gaither Road
Gaithersburg 20877
Tel: (301) 921-0660
FAX: 301-921-4255

MASSACHUSETTS

Arrow Electronics, Inc.
25 Upton Dr.
Wilmington 01887
Tel: (508) 658-0900
TWX: 710-393-6770

Hamilton/Avnet Computer
10 D Centennial Drive
Peabody 01960

†Hamilton/Avnet Electronics
10D Centennial Drive
Peabody 01960
Tel: (508) 532-9838
FAX: 508-596-7802

†Pioneer/Standard Electronics
44 Hartwell Avenue
Lexington 02173
Tel: (617) 861-9200
FAX: 617-863-1547

Wyle Distribution Group
15 Third Avenue
Burlington 01803
Tel: (617) 272-7300
FAX: 617-272-6809

MICHIGAN

†Arrow Electronics, Inc.
19880 Haggerty Road
Livonia 48152
Tel: (313) 665-4100
TWX: 810-223-6020

Hamilton/Avnet Computer
2215 S.E. A-5
Grand Rapids 49508

Hamilton/Avnet Computer
41650 Garden Rd., Ste. 100
Novi 48050

Hamilton/Avnet Electronics
2215 29th Street S.E.
Space A5
Grand Rapids 49508
Tel: (616) 243-8805
FAX: 616-698-1831

Hamilton/Avnet Electronics
41650 Garden Brook
Novi 48050
Tel: (313) 347-4271
FAX: 313-347-4021

†Pioneer/Standard Electronics
4505 Broadmoor S.E.
Grand Rapids 49508
Tel: (616) 698-1800
FAX: 616-698-1831

†Pioneer/Standard Electronics
13485 Stamford
Livonia 48150
Tel: (313) 525-1800
FAX: 313-427-3720

MINNESOTA

†Arrow Electronics, Inc.
5230 W. 73rd Street
Edina 55435
Tel: (612) 830-1800
TWX: 910-576-3125

Hamilton/Avnet Computer
12400 Whitewater Drive
Minnetonka 55343

†Hamilton/Avnet Electronics
12400 Whitewater Drive
Minnetonka 55434
Tel: (612) 932-0600
TWX: 910-576-2720

†Pioneer/Standard Electronics
7625 Golden Triange Dr.
Suite G
Eden Prairie 55343
Tel: (612) 944-3355
FAX: 612-944-3794

MISSOURI

†Arrow Electronics, Inc.
2380 Schuetz
St. Louis 63141
Tel: (314) 567-6888
FAX: 314-567-1164

Hamilton/Avnet Computer
739 Goddard Avenue
Chesterfield 63005

†Hamilton/Avnet Electronics
741 Goddard
Chesterfield 63005
Tel: (314) 537-1600
FAX: 314-537-4248

NEW HAMPSHIRE

Hamilton/Avnet Computer
2 Executive Park Drive
Bedford 03102

Hamilton/Avnet Computer
444 East Industrial Park Dr.
Manchester 03103

NEW JERSEY

†Arrow Electronics, Inc.
4 East Stow Road
Unit 11
Marlton 08053
Tel: (609) 596-8000
FAX: 609-596-9632

†Arrow Electronics
6 Century Drive
Parsipanny 07054
Tel: (201) 538-0900
FAX: 201-538-0900

Hamilton/Avnet Computer
1 Keystone Ave., Bldg. 36
Cherry Hill 08003

Hamilton/Avnet Computer
10 Industrial Road
Fairfield 07006

†Hamilton/Avnet Electronics
1 Keystone Ave., Bldg. 36
Cherry Hill 08003
Tel: (609) 424-0110
FAX: 609-751-2552

†Hamilton/Avnet Electronics
10 Industrial
Fairfield 07006
Tel: (201) 575-3390
FAX: 201-575-5839

†MTI Systems Sales
9 Law Drive
Fairfield 07006
Tel: (201) 227-5552
FAX: 201-575-6336

†Pioneer/Standard Electronics
14-A Madison Rd.
Fairfield 07006
Tel: (201) 575-3510
FAX: 201-575-3454

NEW MEXICO

Alliance Electronics Inc.
10510 Research Avenue
Albuquerque 87123
Tel: (505) 292-3360
FAX: 505-292-6537

Hamilton/Avnet Computer
5659 Jefferson, N.E. Suites A & B
Albuquerque 87109

†Hamilton/Avnet Electronics
5659A Jefferson N.E.
Albuquerque 87109
Tel: (505) 765-1500
FAX: 505-243-1395

NEW YORK

†Arrow Electronics, Inc.
3375 Brighton Henrietta Townline Rd.
Rochester 14623
Tel: (716) 427-0300
TWX: 510-253-4766

Arrow Electronics, Inc.
20 Oser Avenue
Hauppauge 11788
Tel: (516) 231-1000
TWX: 510-227-6623

Hamilton/Avnet Computer
933 Motor Parkway
Haupauge 11788

Hamilton/Avnet Computer
2060 Townline
Rochester 14623

†Hamilton/Avnet Electronics
933 Motor Parkway
Hauppauge 11788
Tel: (516) 231-9800
TWX: 510-224-6166

†Hamilton/Avnet Electronics
2060 Townline Rd.
Rochester 14623
Tel: (716) 272-2744
TWX: 510-253-5470

Hamilton/Avnet Electronics
103 Twin Oaks Drive
Syracuse 13206
Tel: (315) 437-0288
TWX: 710-541-1560

†MTI Systems Sales
38 Harbor Park Drive
Port Washington 11050
Tel: (516) 621-6200
FAX: 510-223-0846

Pioneer/Standard Electronics
68 Corporate Drive
Binghamton 13904
Tel: (607) 722-9300
FAX: 607-722-9562

Pioneer/Standard Electronics
40 Oser Avenue
Hauppauge 11787
Tel: (516) 231-9200
FAX: 510-227-9869

†Pioneer/Standard Electronics
60 Crossway Park West
Woodbury, Long Island 11797
Tel: (516) 921-8700
FAX: 516-921-2143

†Pioneer/Standard Electronics
840 Fairport Park
Fairport 14450
Tel: (716) 381-7070
FAX: 716-381-5955

NORTH CAROLINA

†Arrow Electronics, Inc.
5240 Greensdairy Road
Raleigh 27604
Tel: (919) 876-3132
TWX: 510-928-1856

Hamilton/Avnet Computer
3510 Spring Forest Road
Raleigh 27604

†Hamilton/Avnet Electronics
3510 Spring Forest Drive
Raleigh 27604
Tel: (919) 878-0819
TWX: 510-928-1836

Pioneer/Technologies Group, Inc.
9401 L-Southern Pine Blvd.
Charlotte 28210
Tel: (919) 527-8188
FAX: 704-522-8564

Pioneer Technologies Group, Inc.
2810 Meridian Parkway
Suite 148
Durham 27713
Tel: (919) 544-5400
FAX: 919-544-5885

OHIO

Arrow Commercial System Group
284 Cramer Creek Court
Dublin 43017
Tel: (614) 889-9347
FAX: (614) 889-9680

†Arrow Electronics, Inc.
6238 Cochran Road
Solon 44139
Tel: (216) 248-3990
TWX: 810-427-9409

Hamilton/Avnet Computer
7764 Washington Village Dr.
Dayton 45459

Hamilton/Avnet Computer
30325 Bainbridge Rd., Bldg. A
Solon 44139

†Hamilton/Avnet Electronics
7760 Washington Village Dr.
Dayton 45459
Tel: (513) 439-6733
FAX: 513-439-6711

†Hamilton/Avnet Electronics
30325 Bainbridge
Solon 44139
Tel: (216) 349-5100
TWX: 810-427-9452

Hamilton/Avnet Computer
777 Brooksedge Blvd.
Westerville 43081
Tel: (614) 882-7004
FAX: 614-882-8650

Hamilton/Avnet Electronics
777 Brooksedge Blvd.
Westerville 43081
Tel: (614) 882-7004

MTI Systems Sales
23400 Commerce Park Road
Beachwood 44122
Tel: (216) 464-6688

†Pioneer/Standard Electronics
4433 Interpoint Boulevard
Dayton 45424
Tel: (513) 236-9900
FAX: 513-236-8133

†Pioneer/Standard Electronics
4800 E. 131st Street
Cleveland 44105
Tel: (216) 587-3600
FAX: 216-663-1004

†Certified Technical Distributor

DOMESTIC DISTRIBUTORS (Contd.)

OKLAHOMA

Arrow Electronics, Inc.
4719 South Memorial Dr.
Tulsa 74145

†Hamilton/Avnet Electronics
12121 E. 51st St., Suite 102A
Tulsa 74146
Tel: (918) 252-7297

OREGON

†Almac Electronics Corp.
1885 N.W. 169th Place
Beaverton 97005
Tel: (503) 629-8090
FAX: 503-645-0611

Hamilton/Avnet Computer
9409 Southwest Nimbus Ave.
Beaverton 97005

†Hamilton/Avnet Electronics
9409 S.W. Nimbus Ave.
Beaverton 97005
Tel: (503) 627-0201
FAX: 503-641-4012

Wyle
9640 Sunshine Court
Bldg. G, Suite 200
Beaverton 97005
Tel: (503) 643-7900
FAX: 503-646-5466

PENNSYLVANIA

Arrow Electronics, Inc.
650 Seco Road
Monroeville 15146
Tel: (412) 856-7000

Hamilton/Avnet Computer
2800 Liberty Ave., Bldg. E
Pittsburgh 15222

Hamilton/Avnet Electronics
2800 Liberty Ave.
Pittsburgh 15238
Tel: (412) 281-4150

Pioneer/Standard Electronics
259 Kappa Drive
Pittsburgh 15238
Tel: (412) 782-2300
FAX: 412-963-8255

†Pioneer/Technologies Group, Inc.
Delaware Valley
261 Gibralter Road
Horsham 19044
Tel: (215) 674-4000
FAX: 215-674-3107

TENNESSEE

Arrow Commercial System Group
3635 Knight Road
Suite 7
Memphis 38118
Tel: (901) 367-0540
FAX: (901) 367-2081

TEXAS

Arrow Electronics, Inc.
3220 Commander Drive
Carrollton 75006
Tel: (214) 380-6464
FAX: (214) 248-7208

Hamilton/Avnet Computer
1807A West Braker Lane
Austin 78758

Hamilton/Avnet Computer
Forum 2
4004 Beltline, Suite 200
Dallas 75244

Hamilton/Avnet Computer
4850 Wright Rd., Suite 190
Stafford 77477

†Hamilton/Avnet Electronics
1807 W. Braker Lane
Austin 78758
Tel: (512) 837-8911
TWX: 910-874-1319

†Hamilton/Avnet Electronics
4004 Beltline, Suite 200
Dallas 75234
Tel: (214) 308-8111
TWX: 910-860-5929

†Hamilton/Avnet Electronics
4850 Wright Rd., Suite 190
Stafford 77477
Tel: (713) 240-7733
TWX: 910-881-5523

†Pioneer/Standard Electronics
1826-D Kramer
Austin 78758
Tel: (512) 835-4000
FAX: 512-835-9829

†Pioneer/Standard Electronics
13710 Omega Road
Dallas 75244
Tel: (214) 386-7300
FAX: 214-490-6419

†Pioneer/Standard Electronics
10530 Rockley Road
Houston 77099
Tel: (713) 495-4700
FAX: 713-495-5642

†Wyle Distribution Group
1810 Greenville Avenue
Richardson 75081
Tel: (214) 235-9953
FAX: 214-644-5064

UTAH

Hamilton/Avnet Computer
1585 West 2100 South
Salt Lake City 84119

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City 84119
Tel: (801) 972-2800
TWX: 910-925-4018

†Wyle Distribution Group
1325 West 2200 South
Suite E
West Valley 84119
Tel: (801) 974-9953

WASHINGTON

†Almac Electronics Corp.
14360 S.E. Eastgate Way
Bellevue 98007
Tel: (206) 643-9992
FAX: 206-643-9709

Hamilton/Avnet Computer
17761 Northeast 78th Place
Redmond 98052

†Hamilton/Avnet Electronics
17761 N.E. 78th Place
Redmond 98052
Tel: (206) 881-6697
FAX: 206-867-0159

Wyle Distribution Group
15385 N.E. 90th Street
Redmond 98052
Tel: (206) 881-1150
FAX: 206-881-1567

WISCONSIN

Arrow Electronics, Inc.
200 N. Patrick Blvd., Ste. 100
Brookfield 53005
Tel: (414) 792-0150
FAX: 414-792-0156

Hamilton/Avnet Computer
20875 Crossroads Circle
Suite 400
Waukesha 53186

†Hamilton/Avnet Electronics
28875 Crossroads Circle
Suite 400
Waukesha 53186
Tel: (414) 784-4510
FAX: 414-784-9509

CANADA

ALBERTA

Hamilton/Avnet Computer
2816 21st Street Northeast
Calgary T2E 6Z2

Hamilton/Avnet Electronics
2816 21st Street N.E. #3
Calgary T2E 6Z3
Tel: (403) 230-3586
FAX: 403-250-1591

Zentronics
6815 #8 Street N.E.
Suite 100
Calgary T2E 7H
Tel: (403) 295-8818
FAX: 403-295-8714

BRITISH COLUMBIA

†Hamilton/Avnet Electronics
8610 Commerce Ct.
Burnaby V5A 4N6
Tel: (604) 420-4101
FAX: 604-437-4712

Zentronics
108-11400 Bridgeport Road
Richmond V6X 1T2
Tel: (604) 273-5575
FAX: 604-273-2413

ONTARIO

Arrow Electronics, Inc.
36 Antares Dr., Unit 100
Nepean K2E 7W5
Tel: (613) 226-6903
FAX: 613-723-2018

†Arrow Electronics, Inc.
1093 Meyerside, Unit 2
Mississauga L5T 1M4
Tel: (416) 673-7769
FAX: 416-672-0849

Hamilton/Avnet Computer
Canada System Engineering Group
3688 Nashua Drive
Units 7 & 8
Mississuaga L4V 1M5

Hamilton/Avnet Computer
3688 Nashua Drive
Units 9 & 10
Mississuaga L4V 1M5

Hamilton/Avnet Computer
6845 Rexwood Road
Units 7, 8, & 9
Mississuaga L4V 1R2

Hamilton/Avnet Computer
190 Colonade Road
Nepean K2E 7J5

†Hamilton/Avnet Electronics
6845 Rexwood Road
Units 3-4-5
Mississauga L4T 1R2
Tel: (416) 677-7432
FAX: 416-677-0940

†Hamilton/Avnet Electronics
190 Colonnade Road South
Nepean K2E 7L5
Tel: (613) 226-1700
FAX: 613-226-1184

†Zentronics
1355 Meyerside Drive
Mississauga L5T 1C9
Tel: (416) 564-9600
FAX: 416-564-8320

†Zentronics
155 Colonnade Road
Unit 17
Nepean K2E 7K1
Tel: (613) 226-8840
FAX: 613-226-6352

QUEBEC

Arrow Electronics Inc.
1100 St. Regis
Dorval H9P 2T5
Tel: (514) 421-7411
FAX: 514-421-7430

Arrow Electronics, Inc.
500 Boul. St-Jean-Baptiste
Suite 280
Quebec G2E 5R9
Tel: (418) 871-7500
FAX: 418-871-6816

Hamilton/Avnet Computer
2795 Rue Halpern
St. Laurent H4S 1P8

†Hamilton/Avnet Electronics
2795 Halpern
St. Laurent H2E 7K1
Tel: (514) 335-1000
FAX: 514-335-2481

†Zentronics
520 McCaffrey
St. Laurent H4T 1N3
Tel: (514) 737-9700
FAX: 514-737-5212

†Certified Technical Distributor

EUROPEAN SALES OFFICES

FINLAND

Intel Finland OY
Ruosilantie 2
00390 Helsinki
Tel: (358) 0 544 644
TLX: 123332

FRANCE

Intel Corporation S.A.R.L.
1, Rue Edison-BP 303
78054 St. Quentin-en-Yvelines Cedex
Tel: (33) (1) 30 57 70 00
TLX: 699016

ISRAEL

Intel Semiconductor Ltd.
Atidim Industrial Park-Neve Sharet
P.O. Box 43202
Tel-Aviv 61430
Tel: (972) 03-498080
TLX: 371215

ITALY

Intel Corporation Italia S.p.A.
Milanofiori Palazzo E
20094 Assago
Milano
Tel: (39) (02) 89200950
TLX: 341286

NETHERLANDS

Intel Semiconductor B.V.
Postbus 84130
3099 CC Rotterdam
Tel: (31) 10.407.11.11
TLX: 22283

SPAIN

Intel Iberia S.A.
Zurbaran, 28
28010 Madrid
Tel: (34) (1) 308.25.52
TLX: 46880

SWEDEN

Intel Sweden A.B.
Dalvagen 24
171 36 Solna
Tel: (46) 8 734 01 00
TLX: 12261

SWITZERLAND

Intel Semiconductor A.G.
Zuerichstrasse
8185 Winkel-Rueti bei Zuerich
Tel: (41) 01/860 62 62
TLX: 825977

UNITED KINGDOM

Intel Corporation (U.K.) Ltd.
Pipers Way
Swindon, Wiltshire SN3 1RJ
Tel: (44) (0793) 696000
TLX: 444447/8

WEST GERMANY

Intel GmbH
Dornacher Strasse 1
8016 Feldkirchen bei Muenchen
Tel: (49) 089/90992-0
FAX: (49) 089/904/3948

Intel GmbH
Abraham Lincoln Strasse 16-18
6200 Wiesbaden
Tel: (49) 06121/7605-0
TLX: 4-186183

Intel GmbH
Zettachring 10A
7000 Stuttgart 80
Tel: (49) 0711/7287-280
TLX: 7-254826

EUROPEAN DISTRIBUTORS/REPRESENTATIVES

AUSTRIA

Bacher Electronics G.m.b.H.
Rotenmuehlgasse 26
1120 Wien
Tel: (43) (0222) 83 56 46
TLX: 31532

BELGIUM

Ineico Belgium S.A.
Av. des Croix de Guerre 94
1120 Bruxelles
Oorlogskruisenlaan, 94
1120 Brussel
Tel: (32) (02) 216 01 60
TLX: 64475 or 22090

DENMARK

ITT-Multikomponent
Naverland 29
2600 Glostrup
Tel: (45) (0) 2 45 66 45
TLX: 33 355

FINLAND

OY Fintronic AB
Melkonkatu 24A
00210 Helsinki
Tel: (358) (0) 6926022
TLX: 124224

FRANCE

Almex
Zone industrielle d'Antony
48, rue de l'Aubepine
BP 102
92164 Antony cedex
Tel: (33) (1) 46 66 21 12
TLX: 250067

Jermyn
60, rue des Gemeaux
Silic 580
94653 Rungis Cedex
Tel: (33) (1) 49 78 49 78
TLX: 261585

Metrologie
Tour d'Asnieres
4, av. Laurent-Cely
92606 Asnieres Cedex
Tel: (33) (1) 47 90 62 40
TLX: 611448

Tekelec-Airtronic
Cite des Bruyeres
Rue Carle Vernet - BP 2
92310 Sevres
Tel: (33) (1) 45 34 75 35
TLX: 204552

IRELAND

Micro Marketing Ltd.
Glenageary Office Park
Glenageary
Co. Dublin
Tel: (21) (353) (01) 856288
FAX: (21) (353) (01) 857364
TLX: 31584

ISRAEL

Eastronics Ltd.
11 Rozanis Street
P.O.B. 39300
Tel-Aviv 61392
Tel: (972) 03-475151
TLX: 33638

ITALY

Intesi
Divisione ITT Industries GmbH
Viale Milanofiori
Palazzo E/5
20090 Assago (MI)
Tel: (39) 02/824701
TLX: 311351

Lasi Elettronica S.p.A.
V. le Fulvio Testi, 126
20092 Cinisello Balsamo (MI)
Tel: (39) 02/2440012
TLX: 352040

Telcom S.r.l.
Via M. Civitali 75
20148 Milano
Tel: (39) 02/4049046
TLX: 335654

ITT Multicomponents
Viale Milanofiori E/5
20090 Assago (MI)
Tel: (39) 02/824701
TLX: 311351

Silverstar
Via Dei Gracchi 20
20146 Milano
Tel: (39) 02/49961
TLX: 332189

NETHERLANDS

Koning en Hartman
Elektrotechniek B.V.
Energieweg 1
2627 AP Delft
Tel: (31) (1) 15/609906
TLX: 38250

NORWAY

Nordisk Elektronikk (Norge) A/S
Postboks 123
Smedsvingen 4
1364 Hvalstad
Tel: (47) (02) 84 62 10
TLX: 77546

PORTUGAL

ATD Portugal LDA
Rua Dos Lusiados, 5 Sala B
1300 Lisboa
Tel: (35) (1) 64 80 91
TLX: 61562

Ditram
Avenida Miguel Bombarda, 133
1000 Lisboa
Tel: (35) (1) 54 53 13
TLX: 14182

SPAIN

ATD Electronica, S.A.
Plaza Ciudad de Viena, 6
28040 Madrid
Tel: (34) (1) 234 40 00
TLX: 42477

Metrologia Iberica, S.A.
Ctra. de Fuencarral, n.80
28100 Alcobendas (Madrid)
Tel: (34) (1) 653 86 11

SWEDEN

Nordisk Elektronik AB
Torshamnsgatan 39
Box 36
164 93 Kista
Tel: (46) 08-03 46 30
TLX: 105 47

SWITZERLAND

Industrade A.G.
Hertistrasse 31
8304 Wallisellen
Tel: (41) (01) 8328111
TLX: 56788

TURKEY

EMPA Electronic
Lindwurmstrasse 95A
8000 Muenchen 2
Tel: (49) 089/53 80 570
TLX: 528573

UNITED KINGDOM

Accent Electronic Components Ltd.
Jubilee House, Jubilee Road
Letchworth, Herts SG6 1QH
Tel: (44) (0462) 670011
FAX: (44) (0462) 682467
TWX: 826505

Bytech Components Ltd.
12A Cedarwood
Chineham Business Park
Crockford Lane
Basingstoke
Hants RG24 0WD
Tel: (0256) 707107
FAX: 0256-707162

Conformix
Unit 5
A1M Business Centre
Dixons Hill Road
Welham Green
South Hatfield
Herts AL9 7JE
Tel: (07072) 73282
FAX: (07072) 61678

Bytech Systems
3 The Western Centre
Western Road
Bracknell RG12 1RW
Tel: (44) (0344) 55333
FAX: (44) (0344) 867270
TWX: 849624

Jermyn
Vestry Estate
Otford Road
Sevenoaks
Kent TN14 5EU
Tel: (44) (0732) 450144
FAX: (44) (0732) 451251
TWX: 95142

MMD Ltd.
3 Bennet Court
Bennet Road
Reading
Berkshire RG2 0QX
Tel: (44) (0734) 313232
FAX: (44) (0734) 313255
TWX: 846669

Rapid Recall, Ltd.
Rapid House
Oxford Road
High Wycombe
Buckinghamshire HP11 2EE
Tel: (44) (0494) 26271
FAX: (44) (0494) 21860
TWX: 837931

Rapid Recall, Ltd.
28 High Street
Nantwich
Cheshire CW5 5AS
Tel: (0270) 627505
FAX: (0270) 629883
TWX: 36329

WEST GERMANY

Electronic 2000 AG
Stahlgruberring 12
8000 Muenchen 82
Tel: (49) 089/42001-0
TLX: 522561

ITT Multikomponent GmbH
Postfach 1265
Bahnhofstrasse 44
7141 Moeglingen
Tel: (49) 07141/4879
TLX: 7264472

Jermyn GmbH
Im Dachsstueck 9
6250 Limburg
Tel: (49) 06431/508-0
TLX: 415257-0

Metrologie GmbH
Meglingerstrasse 49
8000 Muenchen 71
Tel: (49) 089/78042-0
TLX: 5213189

Proelectron Vertriebs GmbH
Max Planck Strasse 1-3
6072 Dreieich
Tel: (49) 06103/30434-3
TLX: 417903

YUGOSLAVIA

H.R. Microelectronics Corp.
2005 de la Cruz Blvd., Ste. 223
Santa Clara, CA 95050
U.S.A.
Tel: (1) (408) 988-0286
TLX: 387452

Rapido Electronic Components S.p.a.
Via C. Beccaria, 8
34133 Trieste
Italia
Tel: (39) 040/360555
TLX: 460461

INTERNATIONAL SALES OFFICES

AUSTRALIA

Intel Australia Pty. Ltd.
Unit 13
Allambie Grove Business Park
25 Frenchs Forest Road East
Frenchs Forest, NSW, 2086
Tel: 61-2975-3300
FAX: 61-2975-3375

BRAZIL

Intel Semicondutores do Brazil LTDA
Av. Paulista, 1159-CJS 404/405
01311 - Sao Paulo - S.P.
Tel: 55-11-287-5899
TLX: 3911153146 ISDB
FAX: 55-11-287-5119

CHINA/HONG KONG

Intel PRC Corporation
15/F, Office 1, Citic Bldg.
Jian Guo Men Wai Street
Beijing, PRC
Tel: (1) 500-4850
TLX: 22947 INTEL CN
FAX: (1) 500-2953

Intel Semiconductor Ltd.*
10/F East Tower
Bond Center
Queensway, Central
Hong Kong
Tel: (852) 844-4555
FAX: (852) 868-1989

INDIA

Intel Asia Electronics, Inc.
4/2, Samrah Plaza
St. Mark's Road
Bangalore 560001
Tel: 011-91-812-215065
TLX: 953-845-2646 INTL IN
FAX: 091-812-215067

JAPAN

Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi
Ibaraki, 300-26
Tel: 0298-47-8511
TLX: 3656-160
FAX: 0298-47-8450

Intel Japan K.K.*
Daiichi Mitsugi Bldg.
1-8889 Fuchu-cho
Fuchu-shi, Tokyo 183
Tel: 0423-60-7871
FAX: 0423-60-0315

Intel Japan K.K.*
Bldg. Kumagaya
2-69 Hon-cho
Kumagaya-shi, Saitama 360
Tel: 0485-24-6871
FAX: 0485-24-7518

Intel Japan K.K.*
Kawa-asa Bldg.
2-11-5 Shin-Yokohama
Kohoku-ku, Yokohama-shi
Kanagawa, 222
Tel: 045-474-7661
FAX: 045-471-4394

Intel Japan K.K.*
Ryokuchi-Eki Bldg.
2-4-1 Terauchi
Toyonaka-shi, Osaka 560
Tel: 06-863-1091
FAX: 06-863-1084

Intel Japan K.K.
Shinmaru Bldg.
1-5-1 Marunouchi
Chiyoda-ku, Tokyo 100
Tel: 03-201-3621
FAX: 03-201-6850

Intel Japan K.K.
Green Bldg.
1-16-20 Nishiki
Naka-ku, Nagoya-shi
Aichi 450
Tel: 052-204-1261
FAX: 052-204-1285

KOREA

Intel Korea, Ltd.
16th Floor, Life Bldg.
61 Yoido-dong, Youngdeungpo-Ku
Seoul 150-010
Tel: (2) 784-8186, 8286, 8386
TLX: K29312 INTELKO
FAX: (2) 784-8096

SINGAPORE

Intel Singapore Technology, Ltd.
101 Thomson Road #21-05/06
United Square
Singapore 1130
Tel: 250-7811
TLX: 39921 INTEL
FAX: 250-9256

TAIWAN

Intel Technology Far East Ltd.
8th Floor, No. 205
Bank Tower Bldg.
Tung Hua N. Road
Taipei
Tel: 886-2-716-9660
FAX: 886-2-717-2455

INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

ARGENTINA

Dafsys S.R.L.
Chacabuco, 90-6 Piso
1069-Buenos Aires
Tel: 54-1-334-7726
FAX: 54-1-334-1871

AUSTRALIA

Email Electronics
15-17 Hume Street
Huntingdale, 3166
Tel: 011-61-3-544-8244
TLX: AA 30895
FAX: 011-61-3-543-8179

NSD-Australia
205 Middleborough Rd.
Box Hill, Victoria 3128
Tel: 03 8900970
FAX: 03 8990819

BRAZIL

Elebra Componentes
Rua Geraldo Flausina Gomes, 78
7 Andar
04575 - Sao Paulo - S.P.
Tel: 55-11-534-9641
TLX: 55-11-54593/54591
FAX: 55-11-534-9424

CHINA/HONG KONG

Novel Precision Machinery Co., Ltd.
Room 728 Trade Square
681 Cheung Sha Wan Road
Kowloon, Hong Kong
Tel: (852) 360-8999
TWX: 32032 NVTNL HX
FAX: (852) 725-3695

INDIA

Micronic Devices
Arun Complex
No. 65 D.V.G. Road
Basavanagudi
Bangalore 560 004
Tel: 011-91-812-600-631
011-91-812-611-365
TLX: 9538458332 MDBG

Micronic Devices
No. 516 5th Floor
Swastik Chambers
Sion, Trombay Road
Chembur
Bombay 400 071
TLX: 9531 171447 MDEV

Micronic Devices
25/8, 1st Floor
Bada Bazaar Marg
Old Rajinder Nagar
New Delhi 110 060
Tel: 011-91-11-5723509
011-91-11-589771
TLX: 031-63253 MDND IN

Micronic Devices
6-3-348/12A Dwarakapuri Colony
Hyderabad 500 482
Tel: 011-91-842-226748

S&S Corporation
1587 Kooser Road
San Jose, CA 95118
Tel: (408) 978-6216
TLX: 820281
FAX: (408) 978-8635

JAPAN

Asahi Electronics Co. Ltd.
KMM Bldg. 2-14-1 Asano
Kokurakita-ku
Kitakyushu-shi 802
Tel: 093-511-6471
FAX: 093-551-7861

CTC Components Systems Co., Ltd.
4-8-1 Dobashi, Miyamae-ku
Kawasaki-shi, Kanagawa 213
Tel: 044-852-5121
FAX: 044-877-4268

Dia Semicon Systems, Inc.
Flower Hill Shinmachi Higashi-kan
1-23-9 Shinmachi, Setagaya-ku
Tokyo 154
Tel: 03-439-1600
FAX: 03-439-1601

Okaya Koki
2-4-18 Sakae
Naka-ku, Nagoya-shi 460
Tel: 052-204-2916
FAX: 052-204-2901

Ryoyo Electro Corp.
Konwa Bldg.
1-12-22 Tsukiji
Chuo-ku, Tokyo 104
Tel: 03-546-5011
FAX: 03-546-5044

KOREA

J-Tek Corporation
Dong Sung Bldg. 9/F
158-24, Samsung-Dong, Kangnam-Ku
Seoul 135-090
Tel: (822) 557-8039
FAX: (822) 557-8304

Samsung Electronics
Samsung Main Bldg.
150 Taepyung-Ro-2KA, Chung-Ku
Seoul 100-102
C.P.O. Box 8780
Tel: (822) 751-3680
TWX: KORSST K 27970
FAX: (822) 753-9065

MEXICO

SSB Electronics, Inc.
675 Palomar Street, Bldg. 4, Suite A
Chula Vista, CA 92011
Tel: (619) 585-3253
TLX: 287751 CBALL UR
FAX: (619) 585-8322

Dicopel S.A.
Tochtli 368 Fracc. Ind. San Antonio
Azcapotzalco
C.P. 02760-Mexico, D.F.
Tel: 52-5-561-3211
TLX: 177 3790 Dicome
FAX: 52-5-561-1279

PHI S.A. de C.V.
Fco. Villa esq. Ajusco s/n
Cuernavaca – Morelos
Tel: 52-73-13-9412
FAX: 52-73-17-5333

NEW ZEALAND

Email Electronics
36 Olive Road
Penrose, Auckland
Tel: 011-64-9-591-155
FAX: 011-64-9-592-681

SINGAPORE

Electronic Resources Pte, Ltd.
17 Harvey Road
#03-01 Singapore 1336
Tel: (65) 283-0888
TWX: RS 56541 ERS
FAX: (65) 289-5327

SOUTH AFRICA

Electronic Building Elements
178 Erasmus St. (off Watermeyet St.)
Meyerspark, Pretoria, 0184
Tel: 011-2712-803-7680
FAX: 011-2712-803-8294

TAIWAN

Micro Electronics Corporation
12th Floor, Section 3
285 Nanking East Road
Taipei, R.O.C.
Tel: (886) 2-7198419
FAX: (886) 2-7197916

Acer Sertek Inc.
15th Floor, Section 2
Chien Kuo North Rd.
Taipei 18479 R.O.C.
Tel: 886-2-501-0055
TWX: 23756 SERTEK
FAX: (886) 2-5012521

*Field Application Location



UNITED STATES
Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

JAPAN
Intel Japan K.K.
5-6 Tokodai, Tsukuba-shi
Ibaraki, 300-26

FRANCE
Intel Corporation S.A.R.L.
1, Rue Edison, BP 303
78054 Saint-Quentin-en-Yvelines Cedex

UNITED KINGDOM
Intel Corporation (U.K.) Ltd.
Pipers Way
Swindon
Wiltshire, England SN3 1RJ

GERMANY
Intel GmbH
Dornacher Strasse 1
8016 Feldkirchen bei Muenchen

HONG KONG
Intel Semiconductor Ltd.
10/F East Tower
Bond Center
Queensway, Central

CANADA
Intel Semiconductor of Canada, Ltd.
190 Attwell Drive, Suite 500
Rexdale, Ontario M9W 6H8

intel®

Microprocessors

This year marks the 20-year anniversary of the invention of the microprocessor. This invention resulted in the mass proliferation of computing technology creating the microcomputer revolution of the 1980's. Intel has continued it's technological lead with faster and more capable products for microcomputing.

Intel offers an architecture that provides both the performance and the compatibility needed to take progress from one generation of products to the next.

This handbook contains extensive information on Intel's microprocessor families, numeric coprocessors, cache and memory controllers, and floppy and hard disk controllers. A development tools section is also included for the 8051, 8096, 8086/186/188, 286, 386™ and 486™ processors.

The data sheets and application notes contained in this handbook offer comprehensive charts, diagrams, instructions and hardware information for leading-edge 32-bit system development.

Printed in USA/0191/55K/RRD/GC
Microprocessors

ISBN# 1-55512-115-2