TEXAS INSTRUMENTS

TMS7000 family

Data Manual

1991

8-bit Microcontrollers

TMS7000 Family Data Manual

International Manual

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to or to discontinue any semiconductor product or service identified in this publication without notice. TI advises its customers to obtain the latest version of the relevant information to verify, before placing orders, that the information being relied upon is current.

TI warrants performance of its semiconductor products to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Unless mandated by government requirements, specific testing of all parameters of each device is not necessarily performed.

TI assumes no liability for TI applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Texas Instruments products are not intended for use in life-support appliances, devices, or systems. Use of a TI product in such applications without the written consent of the appropriate TI officer is prohibited.

WARNING

This equipment is intended for use in a laboratory test environment only. It generates, uses, and can radiate radio frequency energy and has not been tested for compliance with the limits of computing devices pursuant to subpart J of part 15 of FCC rules, which are designed to provide reasonable protection against radio frequency interference. Operation of this equipment in other environments may cause interference with radio communications, in which case the user at his own expense will be required to take whatever measures may be required to correct this interference.

Copyright © 1990, Texas Instruments Incorporated

Read This First

This book replaces the following manuals:

- TMS7000 Family Data Manual, SPND001B
- TMS7000 Assembly Language Programmer's Guide, SPNU002B
- TMS7000 Software Development System Installation Guide, MPB52
- TMS7000 IBM CrossWare Support Reference Guide, MPB10
- TMS7000 VAX/VMS CrossWare Support Reference Guide, MPB53

The following table lists related publications.

TMS7000 Data Sheets and Data Manuals	Literature Number
TMS70C42/TMS70C02 Data Sheet	SPNS009
TMS7000 User's Guides	**Literature Number**
8051 – TMS7041 System Conversion User's Guide	SPNU003
TMS7500/TMS75C00 Data Encryption Device User's Guide	SPNU004
Link Editor User's Guide	SPDU037C
TMS7000 EVM User's Guide	SPNU007
TMS7000 Family Development System Support	**Literature Number**
XDS/22 with the TMS7042 Emulator Pocket Reference	SPDF010

How to Use This Manual

This manual is divided into four major parts:

- Hardware (Chapters 2–4)
- Software (Chapters 5–8)
- Development Support (Chapters 9–10)
- Customer Information (Chapter 11)

The chapters and their contents are summarized below.

Chapter 1 Introduction

- Introduces the TMS7000 family devices.
- Describes the different manual sections and their contents.

Chapter 2 TMS7000 Family Devices

- Details each TMS7000 family category and their key features.

- Summarizes the categories and compares their features.
- Provides key features, pinouts, and pin descriptions for each category of devices.

Chapter 3 TMS7000 Family Architecture

- Discusses operation of the microcomputers' hardware features
  - Registers
  - I/O
  - Memory and memory modes
  - Clock options
  - CMOS low-power modes
  - Interrupts
  - Timer/event counters (TMS70Cx0 devices)
  - Serial port (TMS70Cx2 devices only)

Chapter 4 Electrical Specifications

- Discusses for all device groups:
  - Absolute maximum ratings
  - Recommended operating characteristics
  - Recommended crystal/clockin operating characteristics
  - Memory interface timing
  - Read and write cycle timing
  - Ceramic resonator circuit application (where applicable)
  - Serial port timing (where applicable)

Chapter 5 TMS7000 Assembler

- Discusses basic assembler information, including:
  - Source statement format (placement of various fields in code)
  - Constants, symbols, terms, and expressions
- Discusses the various assembler directives, grouped in the following categories:
  - Directives that affect the location counter
  - Directives that affect assembler output
  - Directives that initialize constants

- Directives for linking programs
- Miscellaneous directives

- Assembler Output
  - Explains source listing format and resulting object code.
  - Presents normal completion and abnormal completion error messages.
  - Shows a sample cross reference listing.
  - Discusses object code and the various fields in object code format, and changing object code.
  - Shows assembling file examples.

Chapter 6 Assembly Language Instruction Set

- Provides general instruction set information, such as symbol definitions.
- Defines eight addressing modes used by the instructions.
- Summarizes the instruction set in table form.
- Presents the TMS7000 assembly language instruction set in alphabetical order.

Chapter 7 Linking Program Modules

- Discusses relocation capability, absolute and relocatable code.
- Discusses the Link Editor and includes a sample link control file.
- Reviews directives needed for linking programs.

Chapter 8 Macro Language

- Defines the TMS7000 Macro Assembler.
- Tells how to define macros and use macro libraries.
- Shows how strings, constants, and operators are used in macros.
- Discusses variables, parameters, substitution, and keywords.
- Presents the macro definition verbs.
- Provides macro examples.

Chapter 9 Design Aids

Includes several examples to help you use the TMS7000 family devices:

- Interfacing the TMS7000 to peripheral and memory devices such as extra EPROM and RAM
- Programming the TMS77C82
- Serial communication using the UART (serial port)
- Instruction set application notes
- Sample routines
- The status register
- Stack operations
- Multiplication and shifting

- The branch instruction
- Interrupts
- Write-Only registers

Chapter 10 Development Support
Discusses several products manufactured by Texas Instruments that enhance TMS7000 family design development, including:

- TMS77C82 Starter Kit
- XDS (Extended Development Support) Emulator
- EVM (evaluation module)
- Prototyping devices

Chapter 11 Customer Information

- Discusses quality and reliability.
- Discusses prototype manufacture and production flow, including device prefix designators – TMS, TMP, TMX, and SE.
- Illustrates mechanical package information for all TMS7000 family members
- Provides ordering information for the TMS7000 microcomputers and the Texas Instruments development support products.
- Adaptors and hardware
- Application Boards and Packages

Appendix A TMS7000 Bus Activity Tables

Appendix B TMS7000 NMOS to CMOS Conversion Guide

Appendix C Character Sets

Appendix D Hexadecimal Instruction Table/Opcode Map

Appendix E Instruction Opcode Set

Appendix F Glossary

Index

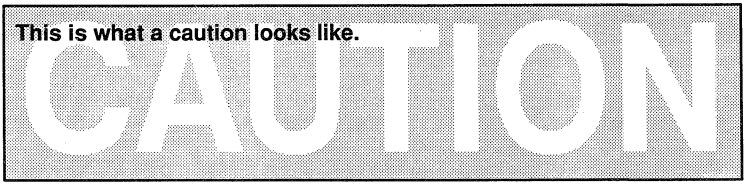
Related Documentation

Communication Equipment Employing Serial Binary Data Interchange, Engineering Department, Electronic Industries Association, 2001 Eye Street, N.W., Washington D.C. 20006, August 1969.

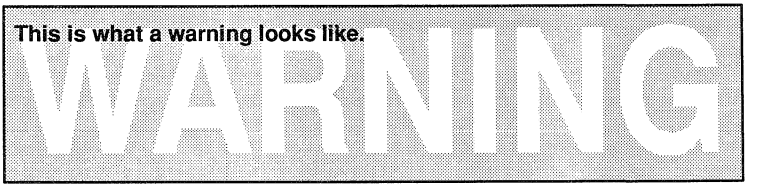
Information about Cautions and Warnings

This book may contain cautions and warnings.

- A **caution** describes a situation that could potentially damage your software or equipment.



- A **warning** describes a situation that could potentially cause harm to **you**.



The information in a caution or a warning is provided for your protection. Please read each caution and warning carefully.

Trademarks

CodeView, MS-Windows, MS, and *MS-DOS* are trademarks of Microsoft Corp.
DEC, Digital DX, VAX, VMS, and *Ultrix* are trademarks of Digital Equipment Corp.
EPIC, XDS, TIGA, and *TIGA-340* are trademarks of Texas Instruments, Inc.
PC-DOS, PGA, and *Micro Channel* are trademarks of IBM Corp.

Contents

1 Introduction **1-1**
1.1 Updates Added to This Manual 1-4

2 TMS7000 Family Devices **2-1**
2.1 Summary and Device Comparison 2-2
2.2 TMS70Cx0 and TMS70CTx0 Key Features 2-5
2.2.1 TMS70CTx0 Key Features 2-5
2.2.2 TMS70Cx0 Key Features 2-6
2.3 TMS70Cx2 Devices 2-10
2.4 TMS70Cx8 Devices 2-13
2.4.1 TMS70Cx8 Key Features 2-13
2.5 SE70CP160, TMS77C82, and SE70CP168 Prototyping Devices 2-18
2.5.1 SE70CP160 (CMOS) Piggyback Prototyping Device Key Features 2-18
2.5.2 SE70CP168 Piggyback Prototyping Device Key Features 2-20
2.5.3 TMP77C82 JDL CMOS EPROM Prototyping Device Key Features 2-22

3 TMS7000 Family Architecture **3-1**
3.1 On-Chip RAM and Registers 3-7
3.1.1 Register File (RF) 3-7
3.1.2 Peripheral File (PF) 3-7
3.1.3 Stack Pointer (SP) 3-7
3.1.4 Status Register (ST) 3-8
3.1.5 Program Counter (PC) 3-9
3.2 On-Chip General Purpose I/O Ports 3-10
3.2.1 Port A 3-14
3.2.2 Port B 3-14
3.2.3 Port C 3-15
3.2.4 Port D 3-15
3.2.5 Ports E and F 3-15
3.2.6 Port G 3-15
3.3 Memory Modes 3-16
3.3.1 Single-Chip Mode 3-22
3.3.2 Peripheral-Expansion Mode 3-25
3.3.3 Full-Expansion Mode 3-27
3.3.4 Microprocessor Mode 3-29
3.4 System Clock Options 3-31

3.4.1 System Clock Connections 3-31
3.4.2 Low-Power Mask Option 3-33
3.5 Low-Power Modes 3-36
3.5.1 TMS70Cx0 and TMS70CTx0 Low-Power Modes 3-36
3.5.2 TMS70Cx2 Devices 3-37
3.5.3 TMS70Cx8 Devices 3-37
3.6 Interrupts and System Reset 3-39
3.6.1 Device Initialization 3-39
3.6.2 Interrupt Operation 3-43
3.6.3 Interrupt Control 3-45
3.6.4 Multiple Interrupt Servicing 3-49
3.6.5 External Interrupt Servicing 3-50
3.6.6 External Interrupt Signals 3-51
3.7 Programmable Timer/Event Counters 3-53
3.7.1 Control Registers for Timer/Event Counters 1 and 2 (TMS70Cx0 and TMS70CTx0 Devices) 3-60
3.7.2 Control Registers for Timer/Event Counters 1 and 2 (TMS70Cx2, TMS77C82, and TMS70Cx8 Devices) 3-60
3.7.3 Timer Start/Stop (Bit 7) and Capture Latch 3-61
3.7.4 Clock Source Control (Bit 6) (See note below.) 3-63
3.7.5 Idle/Timer Halt Bit (Bit 5) 3-64
3.7.6 Cascading Timers 3-64
3.7.7 Timer and Prescaler Operation 3-64
3.7.8 Timer Interrupts 3-66
3.7.9 Pulse Width Modulation, Timer Output Function (TMS70Cx2, TMS77C82, and TMS70Cx8) 3-67
3.8 Serial Port (TMS70Cx2, TMS77C82, and TMS70Cx8 Devices Only) 3-68
3.8.1 Serial Port Registers 3-69
3.8.2 Serial Port Clock Sources 3-83
3.8.3 Multiprocessor Communication 3-86
3.8.4 Serial Port Initialization 3-89
3.8.5 Timer 3 3-90
3.8.6 Initialization Examples 3-91
3.8.7 Serial Port Interrupts 3-95

4 Electrical Specifications 4-1
4.1 TMS70C00, TMS70C20, and TMS70C40 Specifications (Wide Voltage) 4-2
4.2 TMS70C00, TMS70C20, and TMS70C40 Specifications (5V +10%) 4-10
4.3 TMS70CT20 and TMS70CT40 Specifications (5 V + 10%) 4-17
4.4 TMS70C02, TMS70C42, and TMS70C82 Specifications (Wide Voltage) 4-21
4.4.1 Serial Port Timing 4-29
4.5 TMS70C02, TMS70C42, and TMS70C82 Specifications (5V +10%) 4-31
4.5.1 Serial Port Timing 4-38
4.6 TMS70C08 and TMS70C48 Specifications 4-40
4.7 SE70CP160A Specifications 4-47

4.8 TMS77C82 Specifications 4-51
4.8.1 Serial Port Timing 4-57
4.9 SE70CP168 Specifications 4-59

5 The TMS7000 Assembler 5-1
5.1 Source Statement Format 5-2
5.1.1 Label Field 5-5
5.1.2 Command Field 5-5
5.1.3 Operand Field 5-5
5.1.4 Comment Field 5-5
5.2 Constants 5-6
5.2.1 Decimal Integer Constants 5-6
5.2.2 Binary Integer Constants 5-6
5.2.3 Hexadecimal Integer Constants 5-6
5.2.4 Character Constants 5-7
5.2.5 Assembly-Time Constants 5-7
5.3 Symbols 5-8
5.3.1 Predefined Symbols 5-8
5.3.2 Terms 5-8
5.3.3 Character Strings 5-9
5.4 Expressions 5-10
5.4.1 Arithmetic Operators in Expressions 5-10
5.4.2 Logical Operands in Expressions 5-11
5.4.3 Parentheses in Expressions 5-11
5.4.4 Well-Defined Expressions 5-11
5.4.5 Relocatable Symbols in Expressions 5-11
5.4.6 Externally Defined Symbols in Expressions 5-12
5.5 Assembler Directives 5-13
5.6 Symbolic Addressing Techniques 5-49
5.7 Assembler Output 5-50
5.7.1 Source Listing 5-50
5.7.2 Normal Completion Error Messages 5-51
5.7.3 Abnormal Completion Error Messages 5-53
5.7.4 Cross-Reference Listing 5-53
5.8 Object Code 5-55
5.8.1 Object Code Format 5-56
5.9 Assembling Files – Examples 5-63

6 Assembly Language Instruction Set 6-1
6.1 Definitions 6-2
6.2 Addressing Modes 6-3
6.2.1 Single Register Addressing Mode 6-3
6.2.2 Dual Register Addressing Mode 6-4
6.2.3 Peripheral-File Addressing Mode 6-4

6.2.4 Immediate Addressing Mode 6-5
6.2.5 Program Counter Relative Addressing Mode 6-5
6.2.6 Direct Memory Addressing Mode 6-6
6.2.7 Register File Indirect Addressing Mode 6-6
6.2.8 Indexed Addressing Mode 6-7
6.3 Instruction Set Overview 6-8
6.4 Software Compatibility 6-71
6.4.1 TMS70C42 and TMS70C82 Directly Compatible 6-71
6.4.2 TMS70C20, TMS70C40, TMS70CT20, TMS70CT40 6-71

7 Linking Program Modules 7-1
7.1 Relocation Capability 7-2
7.2 Link Editor Operation 7-4
7.3 Directives Used for Linking 7-6
7.4 Creating Linkable Files 7-7
7.5 Linking Files – Examples 7-12

8 Macro Language 8-1
8.1 Defining Macros 8-2
8.1.1 Using Macro Libraries 8-2
8.1.1.1 Using Macro Libraries on MS/PC-DOS Systems 8-4
8.1.2 Sample Macros 8-5
8.2 Strings, Constants, and Operators 8-7
8.3 Variables 8-9
8.3.1 Parameters 8-9
8.3.2 Macro Variable Components 8-9
8.3.3 Variable Qualifiers 8-11
8.3.4 Symbol Components 8-11
8.4 Keywords 8-13
8.4.1 Symbol Attribute Component Keywords 8-13
8.4.2 Parameter Attribute Keywords 8-13
8.5 Assigning Values to Parameters 8-15
8.6 Verbs 8-18
8.7 Model Statements 8-28
8.8 Macro Examples 8-29
8.8.1 Macro ID 8-29
8.8.2 Macro GENCMT 8-30
8.8.3 Macro FACT 8-30
8.8.4 Macro PULSE 8-31
8.9 Macro Error Messages 8-32

9 Design Aids 9-1
9.1 Microprocessor Interface Example 9-2
9.1.1 Read Cycle Timing 9-4

9.1.2 Write Cycle Timing for Microprocessor Mode 9-5
9.2 Programming the TMS77C82 9-6
9.2.1 Programming the TMS77C82 Using an EPROM Programmer 9-6
9.2.2 EPROM Integrity Protection Using the R Bit 9-8
9.2.3 Programming the TMS77C82 Using the TMS7000 Evaluation Module 9-9
9.2.4 Modify the RTC/EVM7000C Debug Monitor to Enable 12.5 Volt VPP Programming 9-10
9.2.5 TMS77C82JDL Erasure 9-11
9.3 Serial Communication with the TMS7000 Family 9-12
9.3.1 Communication Formats 9-12
9.3.2 Software UART (All TMS7000 Devices) 9-13
9.3.3 Hardware UART (TMS7xCx2) 9-19
9.4 The Status Register 9-24
9.4.1 Compare and Jump Instructions 9-24
9.4.2 Addition and Subtraction Instructions 9-25
9.4.3 Swap and Rotation Instructions 9-26
9.5 Stack Operations 9-27
9.6 Subroutine Instructions 9-28
9.7 Multiplication and Shifting 9-30
9.8 The Branch Instruction 9-31
9.9 Interrupts 9-32
9.10 Write-Only Registers 9-34
9.11 Sample Routines 9-35
9.11.1 Clear RAM 9-35
9.11.2 RAM Self Test 9-35
9.11.3 ROM Checksum 9-36
9.11.4 Binary-to-BCD Conversion 9-37
9.11.5 BCD-to-Binary Conversion 9-37
9.11.6 BCD String Addition 9-38
9.11.7 Fast Parity 9-38
9.11.8 Overflow and Underflow 9-39
9.11.9 Bubble Sort 9-40
9.11.10 Table Search 9-40
9.11.11 16-Bit Address Stack Operations 9-41
9.11.12 16-by-16 (32-Bit) Multiplication 9-42
9.11.13 Binary Division, Example 1 9-43
9.11.14 Binary Division, Example 2 9-43
9.11.15 Binary Division, Example 3 9-44
9.11.16 Keyboard Scan 9-45
9.11.17 8-Bit Analog-to-Digital Converter 9-47
9.11.18 Motor Speed Controller 9-48

10 The TMS7000 8-Bit MCU Development Support 10-1
10.1 The RTC/EVM7000 Evaluation Module 10-2

10.1.1 Functional Overview 10-2
10.1.2 Operating System 10-3
10.2 The Interactive Software for the EVM7000 10-4
10.2.1 General Information 10-4
10.3 The Extended Development Support 10-6
10.3.1 XDS/22 Components 10-8
10.4 The Assembly Language 10-9
10.4.1 General 10-9
10.4.2 Assembly Language Application 10-9
10.5 The Link Editor 10-11

11 Customer Information 11-1
11.1 Mask ROM Prototype and Production Flow 11-2
11.1.1 Reserved ROM Locations 11-5
11.1.2 Manufacturing Mask Options 11-5
11.2 Mechanical Package Information 11-7
11.3 TMS7000 Family Numbering and Symbol Conventions 11-16
11.3.1 Device Prefix Designators 11-16
11.3.2 Device Numbering Convention 11-17
11.3.3 Device Symbols 11-17
11.3.3.1 TMS7000 Family Members with On-Chip ROM 11-17
11.3.3.2 TMS7000 Family Members Without On-Chip ROM 11-18
11.4 Development Support Tools Ordering Information 11-19
11.4.1 TMS7000 Macro Assembler/Linker 11-19
11.4.2 TMS7000 XDS Emulators 11-19
11.4.3 TMS7000 Evaluation Modules 11-19
11.4.4 Adaptors and Hardware 11-19

A TMS7000 Bus Activity Tables A-1
A.1 TMS7000 Operating Modes A-2
A.2 TMS7000 Addressing Modes A-3
A.3 Instruction Execution A-5
A.3.1 An Example Using the Bus Activity Tables A-7

B TMS7000 NMOS to CMOS Conversion Guide B-1
B.1 Converting from a TMS70x0 Device to a TMS70xC0 Device B-2
B.1.1 Software B-2
B.1.2 Hardware B-2
B.1.3 Electrical Specifications B-3
B.2 Converting from a TMS70x2 Device to a TMS70Cx2 Device B-4
B.2.1 Software B-4
B.2.2 Hardware B-4
B.2.3 Electrical Specifications B-5

C Character Sets C-1

D Hexadecimal Instruction Table/Opcode Map D-1

E Instruction Opcode Set E-1

F Glossary F-1

Figures

1–1 TI CMOS 8-bit Microcontroller Spectrum 1-1
2–1 Pinout for the TMS70CT20 and TMS70CT40 2-7
2–2 Pinouts for TMS70C00, TMS70C20, TMS70C40 2-7
2–3 Pinouts for TMS70C02, TMS70C42 , TMS70C82, and TMS77C82 Devices 2-11
2–4 Pinouts for TMS70C08 and TMS70C48 2-14
2–5 SE70CP168 Pinout 2-24
2–6 SE70CP160 Pinout 2-25
2–7 TMP77C82JDL Pinout 2-26
3–1 TMS77C82 Block Diagram 3-2
3–2 TMS70Cx2 Block Diagram 3-3
3–3 TMS70CTx0 Block Diagram 3-4
3–4 TMS70Cx0 Block Diagram 3-5
3–5 TMS70Cx8 Block Diagram 3-6
3–6 Example of Stack Initialization in the Register File 3-8
3–7 Status Register (ST) 3-8
3–8 Bidirectional I/O Logic 3-12
3–9 I/O Ports — Single-Chip Mode 3-22
3–10 Single-Chip Mode Memory Map 3-23
3–11 I/O Ports — Peripheral-Expansion Mode 3-26
3–12 Peripheral-Expansion Mode Memory Map 3-26
3–13 I/O Ports — Full-Expansion Mode 3-28
3–14 Full-Expansion Mode Memory Map 3-29
3–15 Microprocessor Mode Memory Map 3-30
3–16 System Clock Connections 3-32
3–17 Frequency Versus Resistance 3-33
3–18 Internal Clock Circuit Block Diagrams 3-34
3–19 HALT/DELAY Block Diagram 3-38
3–20 Sample Initialization Routine for TMS70Cx0 Devices 3-41
3–21 Sample Initialization Routine for TMS70Cx2 and TMS77C82 Devices 3-41
3–22 CPU Interface to Interrupt Logic 3-44
3–23 IOCNT0 — I/O Control Register 0 (P0 for All Devices) 3-46

3–24 IOCNT1 — I/O Control Register 1 3-47
3–25 IOCNT2 — I/O Control Register 2 (P1 for TMS70Cx2 and TMS70Cx8 Only) 3-47
3–26 8-Bit Programmable Timer/Event Counters — Timer 1 (TMS70Cx0 and TMS70CTx0) . 3-54
3–27 16-Bit Programmable Timer/Event Counters — Timer 1 (TMS70Cx2, TMS77C82, and TMS70Cx8) 3-55
3–28 Timer 1 Data and Control Registers (TMS70Cx0 and TMS70CTx0) 3-56
3–29 Timer 1 Data and Control Registers (TMS70Cx2, TMS77C82, and TMS70Cx8) 3-57
3–30 16-Bit Programmable Timer/Event Counters — Timer 2 (TMS70Cx2, TMS77C82, and TMS70Cx8) 3-58
3–31 Timer 2 Data and Control Registers (TMS70Cx2, TMS77C82, and TMS70Cx8) 3-59
3–32 Serial Port Functional Blocks 3-69
3–33 Serial Mode Register — SMODE 3-71
3–34 Serial Control 0 Register — SCTL0 3-73
3–35 Serial Port Status Register — SSTAT 3-75
3–36 Serial Port Control 1 Register — SCTL1 3-77
3–37 Timer 3 Data Register — T3DATA 3-78
3–38 Receive Buffer — RXBUF 3-79
3–39 Transmitter Buffer — TXBUF 3-79
3–40 Asynchronous Communication Format 3-84
3–41 Isosynchronous Communication Format 3-84
3–42 Serial I/O Communication Format 3-85
3–43 Double-Buffered WUT and TXSHF 3-87
3–44 Motorola Multiprocessor Communication Format 3-88
3–45 Intel Multiprocessor Communication Format 3-89
3–46 8-Bit Timer 3 (TMS7xCx2 and TMS70Cx8) 3-90
4–1 Clock Timing 4-5
4–2 Operating Frequency Range 4-7
4–3 Typical Operating Current vs. Supply Voltage 4-7
4–4 Typical Power-Down Current vs. Oscillator Frequency 4-8
4–5 Typical Operating ICC vs. Oscillator Frequency 4-8
4–6 Typical Output Source Characteristics 4-9
4–7 Typical Output Sink Characteristics 4-9
4–8 Output Loading Circuit for Test 4-12
4–9 Measurement Points for Switching Characteristics 4-12
4–10 Clock Timing 4-14
4–11 Output Loading Circuit for Test 4-16
4–12 Read and Write Cycle Timing 4-16
4–13 Output Loading Circuit for Test 4-18

4–14 Measurement Points for Switching Characteristics 4-18
4–15 Clock Timing 4-20
4–16 Output Loading Circuit for Test 4-23
4–17 Clock Timing 4-25
4–18 Operating Frequency Range 4-26
4–19 Typical Operating Current vs. Supply Voltage 4-26
4–20 Typical Operating ICC vs. Oscillator Frequency 4-27
4–21 Typical Operating Current vs. Supply Voltage 4-27
4–22 Typical Output Source Characteristics 4-28
4–23 Typical Output Sink Characteristics 4-28
4–24 Output Loading Circuit for Test 4-33
4–25 Measurement Points for Switching Characteristics 4-34
4–26 Clock Timing 4-35
4–27 Output Loading Circuit for Test 4-37
4–28 Read and Write Cycle Timing 4-37
4–29 Output Loading Circuit for Test 4-42
4–30 Measurement Points for Switching Characteristics 4-43
4–31 Clock Timing 4-44
4–32 Read and Write Cycle Timing 4-46
4–33 Clock Timing 4-50
4–34 Output Loading Circuit for Test 4-53
4–35 Clock Timing 4-55
4–36 Output Loading Circuit for Test 4-61
5–1 TMS7000 Source Code Example 5-3
5–2 TMS7000 Listing File Example 5-4
5–3 Cross-Reference Listing Format 5-54
5–4 Sample Object Code 5-55
6–1 Single Register Addressing Mode Object Code 6-4
6–2 Dual Register Addressing Mode Byte Requirements 6-4
6–3 Peripheral-File Addressing Mode Byte Requirements 6-5
6–4 Immediate Addressing Mode Object Code 6-5
6–5 Program Counter Relative Addressing Mode Object Code 6-6
6–6 Direct Memory Addressing Mode Object Code 6-6
6–7 Register File Indirect Addressing Mode Object Code 6-7
6–8 Indexed Addressing Mode Object Code 6-7
7–1 Absolute Source File Example 7-7
7–2 PROG1.ASM 7-8
7–3 PROG2.ASM 7-8

7–4 PROG3.ASM 7-9
7–5 REGDEF.ASM 7-9
7–6 LINK.CTL 7-10
7–7 LINK.MAP 7-10
9–1 TMS7xCx2 Microprocessor Interface Sample Circuit 9-3
9–2 EPROM Programmer 40-to-28-Pin Conversion Socket 9-6
9–3 44-Pin PLCC to 44-Pin Socket 9-8
9–4 Asynchronous Communication Format 9-12
9–5 I/O Interface 9-13
9–6 Start Bit Detection 9-13
9–7 Status Register 9-24
9–8 Swap and Rotation Operations 9-26
9–9 A Dispatch Table with an Interpretive Program Counter (IPC) 9-27
9–10 Example of a Subroutine Call by Means of a TRAP Instruction 9-29
10–1 Extended Development Support (XDS) System, Model XDS/22 10-6
10–2 Emulation Block Diagram 10-7
11–1 Prototype and Production Flow 11-2
11–2 28-Pin Plastic Package, 70-MIL Pin Spacing (Type N2 Package Suffix) 11-8
11–3 40-Pin Plastic Package, 100-MIL Pin Spacing (Type N Package Suffix) 11-9
11–4 40-Pin Ceramic Package, 100-MIL Pin Spacing (Type JD Package Suffix) 11-10
11–5 40-Pin Ceramic Piggyback Package, 100-MIL Pin Spacing (Type JD Package Suffix) 11-11
11–6 40-Pin N2 Plastic Package, 0.070" Pin Center Spacing 0.600" Pin Row Spacing 11-12
11–7 44-Pin Plastic-Leaded Chip Carrier FN Package 11-13
11–8 68-Pin Plastic-Leaded Chip Carrier FN Package 11-14
11–9 64-Pin Flat Package PG Package 11-15
11–10 Development Flowchart 11-16
11–11 TMS7000 Family Nomenclature 11-17
11–12 TI Standard Symbolization 11-18
11–13 TI Standard Symbolization with Customer Part Number 11-18
11–14 TI Standard Symbolization for Devices without On-Chip ROM 11-18
A–1 Read and Write Timing Diagram A-7

Tables

1–1 Typical Applications for 8-bit Microcontrollers 1-2
1–2 TMS7000 CMOS Family Members 1-3
2–1 TMS7000 CMOS Family Feature Summary 2-3
2–2 TMS70x0 and TMS70Cx0 Pin Descriptions 2-8
2–3 TMS70CTx0 Pin Descriptions 2-9
2–4 TMS7000 CMOS Family Feature Summary 2-10
2–5 TMS70Cx2 and TMS77Cx2 Pin Descriptions 2-12
2–6 TMS70Cx8 Pin Descriptions 2-16
2–7 TMS77C82 Pin Descriptions 2-27
3–1 TMS70Cx0 Port Configuration 3-12
3–2 TMS70CTx0 Port Configuration 3-13
3–3 TMS70Cx8 Port Configuration 3-13
3–4 TMS70Cx2 and TMS77C82 Port Configuration 3-14
3–5 Mode Selection Conditions (MC Pin) 3-16
3–6 TMS70Cx0 and TMS70CTx0 Memory Map 3-17
3–7 TMS70Cx2 and TMS77C82 Memory Map 3-17
3–8 TMS70C48 Memory Map 3-18
3–9 TMS70Cx0 and TMS70CTx0 Peripheral Memory Map 3-18
3–10 TMS70Cx2 and TMS77C82 Peripheral Memory Map 3-19
3–11 TMS70C48 Peripheral Memory Map 3-20
3–12 Low-Power Options for TMS70Cx0 and TMS70CTx0 Devices 3-36
3–13 Low-Power Options for TMS7xCx2 and TMS70Cx8 Devices 3-37
3–14 Interrupt Summary 3-39
3–15 External Interrupt Operation 3-43
3–16 I/O Control Registers 3-45
3–17 Serial Port Control Registers 3-69
3–18 Timer Values for Commonly Used Baud Rates Using Asynchronous Modes — TMS7xCx2 and TMS70Cx8 3-91
4–1 Absolute Maximum Rating over Operating Free-Air Temperature Range (Unless Otherwise Noted) 4-2
4–2 Recommended Operating Conditions 4-2

4–3 Electrical Characteristics over Full Range of Operating Conditions 4-4
4–4 Supply Current Requirements 4-5
4–5 Recommended Crystal/Clockin Operating Conditions over Full Operating Range 4-6
4–6 Absolute Maximum Rating over Operating Free-Air Temperature Range (Unless Otherwise Noted) 4-10
4–7 Recommended Operating Conditions 4-11
4–8 Electrical Characteristics over Full Range of Operating Conditions 4-12
4–9 AC Characteristics for I/O Ports 4-12
4–10 Supply Current Requirements 4-13
4–11 Recommended Crystal/Clockin Operating Conditions over Full Operating Range 4-13
4–12 Memory Interface Timings 4-15
4–13 Absolute Maximum Ratings over Operating Free-Air Temperature Range (Unless Otherwise Noted) 4-17
4–14 Recommended Operating Conditions 4-17
4–15 Electrical Characteristics over Full Range of Operating Conditions 4-18
4–16 AC Characteristics for I/O Port 4-18
4–17 Supply Current Requirements 4-19
4–18 Recommended Crystal/Clockin Operating Conditions over Full Operating Range 4-19
4–19 Absolute Maximum Ratings over Operating Free-Air Temperature Range (Unless Otherwise Noted) 4-21
4–20 Recommended Operating Conditions 4-22
4–21 Electrical Characteristics over Full Range of Operating Conditions 4-23
4–22 Supply Current Requirements 4-24
4–23 Recommended Crystal/Clockin Operating Conditions over Full Operating Range 4-25
4–24 Absolute Maximum Ratings over Operating Free-Air Temperature Range (Unless Otherwise Noted) 4-31
4–25 Recommended Operating Conditions 4-32
4–26 Electrical Characteristics over Full Range of Operating Conditions 4-33
4–27 AC Characteristics for Input/Output Ports 4-33
4–28 Supply Current Requirements 4-34
4–29 Recommended Crystal/Clockin Operating Conditions over Full Operating Range 4-35
4–30 Memory Interface Timings 4-36
4–31 Absolute Maximum Ratings over Operating Free-Air Temperature Range (Unless Otherwise Noted) 4-40
4–32 Recommended Operating Conditions 4-41
4–33 Electrical Characteristics over Full Range of Operating Conditions 4-42
4–34 AC Characteristics for Input/Output Ports 4-42
4–35 Supply Current Requirements 4-43

4–36 Recommended Crystal/Clockin Operating Conditions over Full Operating Range 4-44
4–37 Memory Interface Timings 4-45
4–38 Absolute Maximum Rating over Operating Free-Air Temperature Range (Unless Otherwise Noted) 4-47
4–39 Recommended Operating Conditions 4-47
4–40 Electrical Characteristics over Full Range of Operating Conditions 4-48
4–41 Supply Current Requirements 4-49
4–42 Recommended Crystal/Clockin Operating Conditions over Full Operating Range 4-50
4–43 Absolute Maximum Ratings over Operating Free-Air Temperature Range (Unless Otherwise Noted) 4-51
4–44 Recommended Operating Conditions 4-52
4–45 Electrical Characteristics over Full Range of Operating Conditions 4-53
4–46 Supply Current Requirements 4-54
4–47 Recommended Crystal/Clockin Operating Conditions over Full Operating Range 4-55
4–48 Memory Interface Timings 4-56
4–49 Absolute Maximum Ratings over Operating Free-Air Temperature Range (unless otherwise noted) 4-59
4–50 Recommended Operating Conditions 4-59
4–51 Electrical Characteristics over the Full Range of Operating Conditions 4-60
4–52 Supply Current Requirements 4-61
5–1 Results of Operations on Absolute and Relocatable Items in Expressions 5-12
5–2 Summary of Assembler Directives 5-14
5–3 Assembly Listing Errors 5-51
5–4 Abnormal Completion Error Messages 5-53
5–5 Symbol Attributes 5-54
5–6 Tag Characters 5-56
5–7 Object Record Format and Tags 5-59
6–1 TMS7000 Symbol Definitions 6-2
6–2 TMS7000 Addressing Modes 6-3
6–3 TMS7000 Family Instruction Overview 6-8
6–4 Compare Instruction Examples — Status Bit Values 6-28
7–1 Linker Commands Used to Link TMS7000 Program Modules 7-5
8–1 Variable Qualifiers 8-11
8–2 Variable Qualifiers for Symbol Components 8-12
8–3 Symbol Attribute Keywords 8-13
8–4 Parameter Attribute Keywords 8-14
8–5 Macro Language Verb Summary 8-18

8–6 Macro Error Messages 8-32
9–1 Memory Address Decode 9-2
9–2 TMS6716-25N Timing Characteristics 9-5
9–3 TMS27C64-15 Timing Characteristics 9-5
9–4 SN74AS363, SN74AF138, and SN74AS32 Propagation Delay Times 9-5
9–5 Truth Table for EPROM and R Bit 9-9
9–6 Serial Port Control Registers 9-19
9–7 Compare Instruction Examples: Status Bit Values 9-25
9–8 Status Bit Values for Conditional Jump Instructions 9-25
9–9 Multi-Bit Right or Left Shifts by Immediate Multiply 9-30
9–10 Write-Only Registers 9-34
11–1 Valid ROM Start Addresses 11-5
11–2 Package Types 11-7
A–1 Alphabetical Index of Instruction Groups A-9
A–2 Instruction Acquisition Mode — Opcode Fetch A-11
A–3 Instruction Acquisition Mode — Interrupt Handling A-11
A–4 Instruction Acquisition Mode — Reset A-12
A–5 Double Operand Functions — Addressing Modes (ADD, ADC, AND, BTJO, BTJZ, CMP, DAC, DSB, MOV, MPY, OR, SBB, SUB, XOR) . A-13
A–6 Double Operand Functions — Functional Modes (ADD, ADC, AND, BTJO, BTJZ, CMP, DAC, DSB, MOV, MPY, OR, SBB, SUB, XOR) . A-14
A–7 Miscellaneous Functions — Addressing Modes (DINT, EINT, IDLE, LDSP, NOP, POP ST, PUSH ST, RETI, RETS, SETC, STSP) A-15
A–8 Miscellaneous Functions — Functional Modes (DINT,EINT,IDLE,LDSP,NOP,POP ST,PUSH ST,RETI,RETS,SETC,STSP) A-15
A–9 Long Addressing Functions — Addressing Modes (BR, CALL, CMPA, LDA, STA) A-16
A–10 Long Addressing Functions — Functional Modes (BR, CALL, CMPA, LDA, STA) A-16
A–11 Single Operand Functions, Special — Addressing Modes (CLR, DEC, INC, INV, MOV A B, MOV A RN, MOV B RN, SWAP, TSTA/CLRC, TSTB, XCHB) A-17
A–12 Single Operand Functions, Special — Functional Modes (CLR, DEC, INC, INV, MOV A B, MOV A RN, MOV B RN, SWAP, TSTA/CLRC, TSTB, XCHB) A-17
A–13 Single Operand Functions, Normal — Addressing Modes (DECD, DJNZ, POP, PUSH, RL, RLC, RR, RRC) A-18
A–14 Single Operand Functions, Normal — Functional Modes (DECD, DJNZ, POP, PUSH, RL, RLC, RR, RRC) A-18
A–15 Double Operand Functions, Peripheral — Addressing Modes (ANDP, BTJOP, BTJZP, MOVP, ORP, XORP) A-19
A–16 Double Operand Functions, Peripheral — Functional Modes (ANDP, BTJOP, BTJZP, MOVP, ORP, XORP) A-20

A–17 Move Double — Addressing Mode (MOVD) A-21
A–18 Move Double — Functional Mode (MOVD) A-21
A–19 Relative Jumps — Addressing and Functional Modes (JMP, JN/JLT, JZ/JEQ, JC/JHS, JP/JGT, JPZ/JGE, JNZ/JNE, JNC, JL) A-22
A–20 Traps — Addressing and Functional Modes (Trap 0 through Trap 23) A-22
C–1 ASCII Character Set C-2
C–2 Control Characters C-3

Examples

A–1 Execution Steps for ADD (Instruction Acquisition) A-8
A–2 Execution Steps for ADD (Addressing Modes) A-8
A–3 Execution Steps for ADD (Functional Modes) A-8

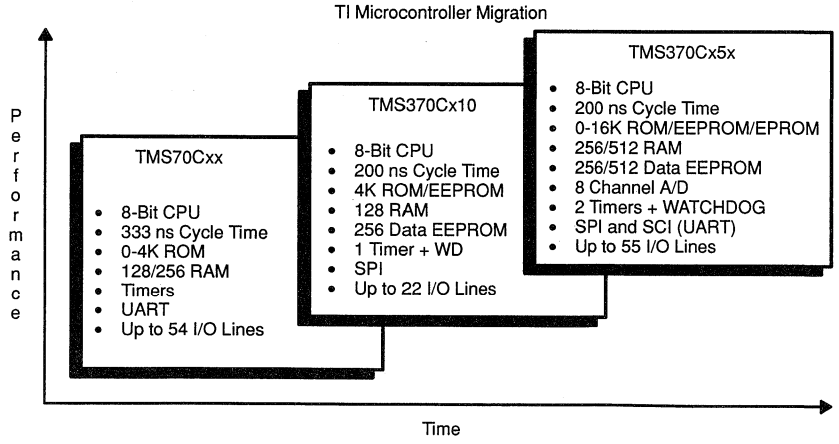
Chapter 1

Introduction

The TMS7000 is a family of 8-bit single-chip microcontrollers. These microcontrollers incorporate a CPU, memory (ROM, RAM, EPROM), bit I/O, serial communications port, timers, interrupts, and external bus interface logic, all on a single chip. The CMOS Microcontrollers provide an outstanding speed/power ratio as a result of the reliable silicon-gate CMOS technology. The terms *TMS7000* and *TMS7000 family* refer to all TMS7000 devices: TMS70C00, TMS70C20, TMS70C40, TMS70C02, TMS70C42, TMS70CT20, TMS70CT40, TMS70C82, TMS70C48, TMS77C82 and all future members, unless otherwise stated.

Eight-bit microcontrollers have the versatility to cover a wide spectrum of applications. Texas Instruments has two microcontroller families — TMS7000 and TMS370 — which provide reliable alternatives to satisfy the design requirements.

Figure 1–1. TI CMOS 8-bit Microcontroller Spectrum



As shown in Figure 1–1, the TMS7000 satisfies those applications in the low to mid range. On the other hand, the TMS370 family with its on-chip EEPROM, superior performance and other peripheral support functions including the

on-chip analog-to-digital converter addresses the high end applications. Table 1–1 describes typical applications for eight-bit microcontrollers.

Table 1–1. Typical Applications for 8-bit Microcontrollers

Automotive	Telecom
Instrumentation	Feature phones
Audio entertainment control	Autodialers
Cruise control	Answering machines
Anti-skid braking system	Modem control
Climate control	Digital switches
Engine control	Digital subsets
Trip computer	
Computer	**Industrial**
Printers and plotters	Motor control
Disk controllers	Stepper motors
Tape drive control	Metering and measurement
Keyboards	Robotics
Touch screen and mouse	
Consumer	**Business**
Home security	Cash registers
Cable TV systems	Automatic bank tellers
Appliance control	Barcode readers

TI supports the TMS7000 family with a variety of development tools. The Extended Development Systems provide realtime in-circuit emulation, onboard software breakpoints, and reverse assembler. With its extensive debugging features, critical development time is reduced. The evaluation module provides low cost in-circuit emulation for the TMS7000 family members. TI offers a wide range of prototyping devices for the TMS7000 CMOS family members with the SE70CP160 for use with the TMS70C20/40/CT20/CT40, the TMP77C82JDL for use with the TMS70C42/82, and the SE70CP168 for use with the TMS70C48.

Table 1–2 details the broad TMS7000 CMOS family members.

Table 1–2. TMS7000 CMOS Family Members

	TMS70C42 TMS70C02/ TMS70C82	TMS70C40/ TMS70C20/ TMS70C00	TMS70CT40/ TMS70CT20	TMS70C48/ TMS70C08
On-Chip ROM (K bytes)	8 / 4 / 0	4 / 2 / 0	4 / 2	4 / 0
Internal RAM (bytes)	256	128	128	256
Interrupt levels	6	4	4	6
Timers: 21-bit 13-bit 10-bit	2 – 1	– 1 –	– 1 –	2 – 1
I/O Lines: Bidirectional Input Only Output Only	24 – 8	16 8 8	12 4 4	46 – 8
Additional I/O:	UART	–		UART
Package:	40 pin DIP 44 pin PLCC	40 pin DIP 44 pin PLCC	28 pin DIP	64 pin Flat 68 pin PLCC
Prototyping: Piggyback EPROM	– TMS77C82	SE70CP160 –	SE70CP160 –	SE70CP168 –

1.1 Updates Added to This Manual

This manual replaces the previous TMS7000 Family Data Manual (literature number SND001B). Additional information has been added to the following sections:

- TMS370 Family Devices — Briefly described in Chapter 1
- TMS70CTx0 Devices — New devices added to family:
  - Key Features ..... Section 2.2.1
  - Pinouts ..... Figure 2–1
  - Pin Description ..... Table 2–3
  - Memory Map ..... Section 3.3
  - Functionality ..... Chapter 3
  - Electrical Specifications ..... Sections 4.2 and 4.3
  - Mechanical Drawings ..... Section 11.2
- TMS70Cx8 Devices — New devices added to family:
  - Key Features ..... Section 2.4.1
  - Pinouts ..... Figure 2–4
  - Pin Description ..... Table 2–6
  - Memory Map ..... Section 3.3
  - Functionality ..... Chapter 3
  - Electrical Specifications ..... Section 4.6
  - Mechanical Drawings ..... Section 11.2
- NMOS to CMOS Conversion Guide: Appendix B describes the alterations required when upgrading from an existing TMS7000 NMOS design to a CMOS design.
- TMS77C82 EPROM Device — New device added to the family:
  - Key Features ..... Section 2.5.3
  - Pinouts ..... Figure 2–6
  - Pin Description ..... Table 2–7
  - Memory Map ..... Section 3.3
  - Functionality ..... Chapter 3
  - Electrical Specifications ..... Section 4.8
  - Programming Algorithm Design Aid ..... Section 9.3
  - Mechanical Drawings ..... Section 11.2

Chapter 2

TMS7000 Family Devices

This chapter discusses the features of the TMS7000 family of microcomputers. All family members are instruction-set compatible, allowing easy migration within the TMS7000 family by maintaining a software base, development tools, and design expertise.

The TMS7000 family devices are divided into several categories:

- **TMS70Cx0 devices** include the TMS70C00, TMS70C20, and TMS70C40
- **TMS70CTx0 devices** include the TMS70CT20 and TMS70CT40
- **TMS70Cx2 devices** include the TMS70C02, TMS70C42, TMS70C82
- **TMS70Cx8 devices** include the TMS70C08 and TMS70C48
- **EPROM devices** include the TMS77C82
- **Prototyping devices** include the SE70CP160, SE70CP168, and TMP77C82JDL

This chapter begins with a summary and comparison of the TMS7000 family devices, and then provides key features, pinouts, and pin descriptions for the individual categories.

Section		Page
2.1	Summary and Device Comparison	2-2
2.2	TMS70Cx0 and TMS70CTx0 Devices	2-5
2.3	TMS70Cx2 Devices	2-10
2.4	TMS70Cx8 Devices	2-13
2.5	SE70CP160, SE70CP162, TMS77C82, and SE70CP168 Prototyping Devices	2-18

Note:

Throughout this manual, the term *TMS7000* or *TMS7000family* refers to all members of the group.

2.1 Summary and Device Comparison

- The **TMS70Cx0** CMOS devices replace the original TMS70x0 range of NMOS devices. They are architecture and instruction-set compatible and feature a powerful 8-bit CPU, 8 input pins, 8 output pins and 16 bit-programmable I/O pins. On-chip ROM sizes of 0, 2 and 4K bytes are available, with on-chip RAM of 128 bytes. They have a single 8-bit timer/event counter with automatic timer reload, 5-bit prescaler and capture function. Three prioritized interrupts are available from two external pins and the timer. The devices also feature two low-current modes; a wake-up mode which halts the CPU but leaves the timer active, and a halt mode that provides RAM retention only. Software selected interrupts resume CPU activity. Off-chip memory expansion is provided by two software controlled modes which allow either 256 bytes or 64K bytes of external address range.

- The **TMS70CTX0** CMOS devices have the same basic features as the TMS70Cx0, but have been I/O- and specification-reduced to provide the most cost-reduced TMS7000 versions. They provide four input pins, four output pins, and 12 bit-programmable I/O pins. The timer/capture functions remain identical, with the exception that no event counter function is available. Off-chip memory expansion is only available up to 256 bytes. ROMless devices are not available in this range.

- The **TMS70Cx2** CMOS devices provide enhanced features over the standard TMS70Cx0 range. ROM sizes are available in 0, 4 and 8K bytes, and the RAM has been increased to 256 bytes. The I/O has been enhanced to provide 8 outputs and 24 bit-programmable I/O pins. The single 8-bit timer has been replaced by two independent 16-bit timer/event counters with 5-bit prescalers, each with 16-bit capture functions and event inputs. A hardware UART has been included which provides both asynchronous and clocked modes, allowing standard interprocessor and shift register peripheral device formats to be used. The UART has its own dedicated timer for baud rate generation. Prioritized interrupts from both timers and the UART are provided. The two external interrupt pins have also been enhanced to provide independent software programmed sense polarity. External memory expansion modes of 256 bytes and 64K bytes are available.

- The **TMS70Cx8** CMOS device has the same basic features as the TMS70Cx2 range. It is available with ROM sizes of 0 and 4K bytes. The I/O has been expanded to provide 8 output pins and 46 bit-programmable I/O pins. External memory expansion modes of 256 bytes and 64k bytes are available, with additional options to provide multiplexed/nonmultiplexed data/address and chip select signals.

- **Prototyping** devices are available to provide accurate form-factor emulation for all family members. For the TMS70Cx0 the SE70CP160 piggyback EPROM device should be used. For the TMS70CTx0 the SE70CP160 should also be used, with the addition of a 40 to 28 pin adaptor. For the TMS70Cx2 the TMP77C82JDL EPROM device should be used. This device can also be used with the TMS70Cx0 and TMS70CTx0 devices provided some minor software modifications are made which are included in Section 2.5.

Table 2–1. TMS7000 CMOS Family Feature Summary

	TMS70C40/ TMS70C20/ TMS70C00	**TMS70CT40/ TMS70CT20**	**TMS77C82/ TMS70C82/ TMS70C42/ TMS70C02**	**TMS70C48/ TMS70C08**
Max osc freq at 5V ± 10 %	5 MHz	5 MHz	6 MHz	6 MHz
Voltage	2.5 to 6 V	5 V ± 10 %	2.5 to 6 V	2.5 to 6 V
Operating temperature Industrial Commercial	 –40°C to 85°C 0°C to 70°C	 – 0°C to 70°C	 –40°C to 85°C 0°C to 70°C	 –40°C to 85°C 0°C to 70°C
On-chip ROM (Kbytes)	4 / 2 / 0	4 / 2	8 / 4 / 0	4 / 0
Internal RAM (bytes)	128	128	256	256
Interrupt levels: External Total	 2 4	 2 4	 2 6	 2 6
Timers/event counters: 21-bit 13-bit 10-bit	 – 1 –	 – 1 –	 2 – 1	 2 – 1
I/O lines: Bidirectional Input only Output only	16 8 8	12 4 4	24 – 8	46 – 8
Additional features	–	–	Serial Port	Serial Port
Package	40 pin DIP 44 pin PLCC	28 pin DIP –	40 pin DIP 44 pin PLCC	64 pin FLAT 68 pin PLCC
Development support: Prototyping: EPROM Piggyback XDS EVM Starter Kit	 – SE70CP160 Yes Yes –	 – SE70CP160† Yes Yes –	 TMS77C82 SE70CP162 Yes Yes Yes	 – SE70CP168 Yes Yes –

† Requires ATC70CT40 interface socket adaptor for pinout conversion (40 pin, 28 pin conversion).

- Register-to-register architecture
- Memory-mapped ports for easy addressing
- Eight addressing formats
- Single-instruction binary-coded decimal (BCD) add and subtract
- Two external maskable interrupts
- Flexible interrupt handling
- Wide voltage operating range, frequency range
- Two power-down modes
- Silicon-gate CMOS technology
- Warning, the RAM size of the SE70CP160 is not 128 but 256 bytes

2.2 TMS70Cx0 and TMS70CTx0 Key Features

2.2.1 TMS70CTx0 Key Features

	TMS70C40/ TMS70C20/ TMS70C00	TMS70CT40/ TMS70CT20	TMS77C82/ TMS70C82/ TMS70C42/ TMS70C02	TMS70C48/ TMS70C08
Max osc freq at 5V ± 10 %	5 MHz	5 MHz	6 MHz	6 MHz
Voltage	2.5 to 6 V	5 V ± 10 %	2.5 to 6 V	2.5 to 6 V
Operating temperature Industrial Commercial	 –40°C to 85°C 0°C to 70°C	 – 0°C to 70°C	 –40°C to 85°C 0°C to 70°C	 –40°C to 85°C 0°C to 70°C
On-chip ROM (Kbytes)	4 / 2 / 0	4 / 2	8 / 4 / 0	4 / 0
Internal RAM (bytes)	128	128	256	256
Interrupt levels: External Total	 2 4	 2 4	 2 6	 2 6
Timers/event counters: 21-bit 13-bit 10-bit	 – 1 –	 – 1 –	 2 – 1	 2 – 1
I/O lines: Bidirectional Input only Output only	16 8 8	12 4 4	24 – 8	46 – 8
Additional features	–	–	Serial Port	Serial Port
Package	40 pin DIP 44 pin PLCC	28 pin DIP –	40 pin DIP 44 pin PLCC	64 pin FLAT 68 pin PLCC
Development support: Prototyping: EPROM Piggyback XDS EVM Starter Kit	 – SE70CP160 Yes Yes –	 – SE70CP160† Yes Yes –	 TMS77C82 SE70CP162 Yes Yes Yes	 – SE70CP168 Yes Yes –

† Requires ATC70CT40 interface socket adaptor for pinout conversion (40 pin, 28 pin conversion).

- Register-to-register architecture
- Memory-mapped ports for easy addressing
- Eight addressing formats
- Single-instruction binary-coded decimal (BCD) add and subtract
- Two external maskable interrupts
- Flexible interrupt handling
- Voltage operating range; 5 V ± 10%
  - Frequency operating range 0.8 MHz to 5.0 MHz
- Two power-down modes:
  - Wake-up (160 µA at 1 MHz typical)
  - Halt, XTAL/CLKIN=GND (1 µA typical)
- Silicon-gate CMOS technology
- 28-pin, 400 mil, dual-inline package

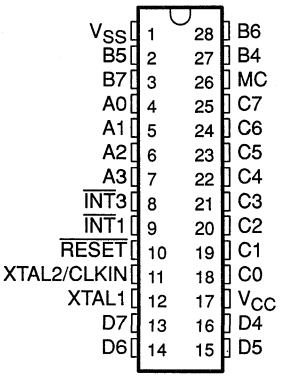
2.2.2 TMS70Cx0 Key Features

	TMS70C40/ TMS70C20/ TMS70C00	TMS70CT40/ TMS70CT20	TMS77C82/ TMS70C82/ TMS70C42/ TMS70C02	TMS70C48/ TMS70C08
Max osc freq at 5V ± 10 %	5 MHz	5 MHz	6 MHz	6 MHz
Voltage	2.5 to 6 V	5 V ± 10 %	2.5 to 6 V	2.5 to 6 V
Operating temperature Industrial Commercial	 –40°C to 85°C 0°C to 70°C	 – 0°C to 70°C	 –40°C to 85°C 0°C to 70°C	 –40°C to 85°C 0°C to 70°C
On-chip ROM (Kbytes)	4 / 2 / 0	4 / 2	8 / 4 / 0	4 / 0
Internal RAM (bytes)	128	128	256	256
Interrupt levels: External Total	 2 4	 2 4	 2 6	 2 6
Timers/event counters: 21-bit 13-bit 10-bit	 – 1 –	 – 1 –	 2 – 1	 2 – 1
I/O lines: Bidirectional Input only Output only	16 8 8	12 4 4	24 – 8	46 – 8
Additional features	–	–	Serial Port	Serial Port
Package	40 pin DIP 44 pin PLCC	28 pin DIP –	40 pin DIP 44 pin PLCC	64 pin FLAT 68 pin PLCC
Development support: Prototyping: EPROM Piggyback XDS EVM Starter Kit	 – SE70CP160 Yes Yes –	 – SE70CP160† Yes Yes –	 TMS77C82 SE70CP162 Yes Yes Yes	 – SE70CP168 Yes Yes –

† Requires ATC70CT40 interface socket adaptor for pinout conversion (40 pin, 28 pin conversion).

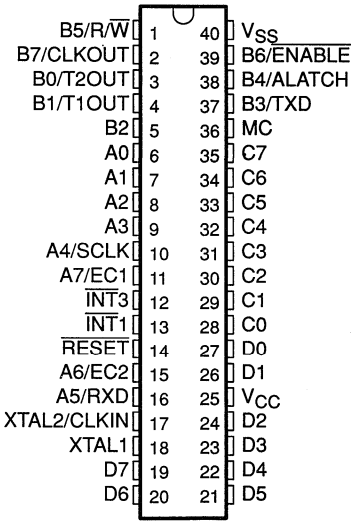
- Register-to-register architecture
- Memory-mapped ports for easy addressing
- Eight addressing formats
- Single-instruction binary-coded decimal (BCD) add and subtract
- Two external maskable interrupts
- Flexible interrupt handling
- Wide voltage operating range, frequency range:
  - 2.5 V – 0.8 MHz maximum
  - 6 V – 6.5 MHz maximum
- Two power-down modes:
  - Wake-Up (160 µA at 1 MHz typical)
  - Halt, XTAL/CLKIN=GND (1 µA typical)
- Silicon-gate CMOS technology
- 40-pin, 600 mil, dual-inline package
- 44-pin PLCC

Figure 2–1. Pinout for the TMS70CT20 and TMS70CT40

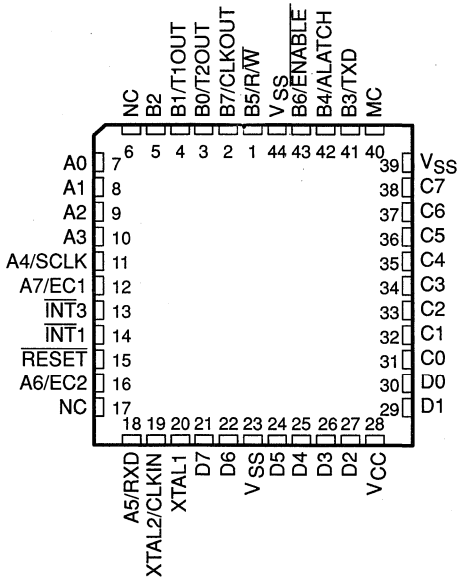


Plastic 28-Pin DIP

Figure 2–2. Pinouts for TMS70C00, TMS70C20, TMS70C40



A. 40-Pin DIP



B. 44-Pin PLCC

Table 2–2. TMS70x0 and TMS70Cx0 Pin Descriptions

Signal	Pin		I/O	Description
	PLCC	DIP		
A0 LSb	7	6	I	Port A. All pins may be used as high-impedance input-only lines. Pin A7/EC1 may also be used as the timer/event counter input.
A1	8	7	I	
A2	9	8	I	
A3	10	9	I	
A4	11	10	I	
A5	18	16	I	
A6	16	15	I	
A7/EC1	12	11	I	
B0	3	3	O	Port B. B0–B7 are general-purpose output-only pins. B4–B7 become memory-expansion control signals in peripheral-expansion, full-expansion, and microprocessor modes.
B1	4	4	O	
B2	5	5	O	
B3	41	37	O	
B4/ALATCH	42	38	O	Data output/memory interface address latch strobe
B5/R/$\overline{W}$	1	1	O	Data output/memory read/write signal
B6/$\overline{ENABLE}$	43	39	O	Data output/memory interface enable strobe
B7/CLKOUT	2	2	O	Data output/internal clockout
C0	31	28	I/O	Port C. C0–C7 can be individually selected in software as general-purpose input or output pins in Single-Chip mode. C0–C7 become the LSB address/data bus in Peripheral-Expansion, Full-Expansion, and Microprocessor modes.
C1	32	29	I/O	
C2	33	30	I/O	
C3	34	31	I/O	
C4	35	32	I/O	
C5	36	33	I/O	
C6	37	34	I/O	
C7	38	35	I/O	
D0	30	27	I/O	Port D. D0–D7 can be individually selected in software as general-purpose input or output pins in Single-Chip or Peripheral-Expansion modes. D0–D7 become the MSB address/data bus in Full-Expansion and Microprocessor modes.
D1	29	26	I/O	
D2	27	24	I/O	
D3	26	23	I/O	
D4	25	22	I/O	
D5	24	21	I/O	
D6	22	20	I/O	
D7	21	19	I/O	
$\overline{INT1}$	14	13	I	Highest priority maskable interrupt
$\overline{INT3}$	13	12	I	Lowest priority maskable interrupt
$\overline{RESET}$	15	14	I	Device reset
MC	40	36	I	Mode control pin, V_{CC} for microprocessor mode
XTAL2/CLKIN	19	17	I	Crystal input for control of internal oscillator
XTAL1	20	18	O	Crystal output for control of internal oscillator
V_{CC}	28	25		Supply voltage (positive)
V_{SS}	44 39 23	40		Ground reference

Table 2–3. TMS70CTx0 Pin Descriptions

Signal	Pin	I/O	Description
A0 A1 A2 A3	4 5 6 7	I I I I	Port A. High impedance input only pins.
B4 B5 B6 B7	27 2 28 3	O O O O	Port B. General purpose output only pins. B4–B7 become memory-expansion control signals in peripheral-expansion mode.
C0 C1 C2 C3 C4 C5 C6 C7	18 19 20 21 22 23 24 25	I/O I/O I/O I/O I/O I/O I/O I/O	Port C. Individually selectable in software as general purpose input or output pins. C0–C7 become the LSB address/data bus in the peripheral-expansion mode.
D4 D5 D6 D7	16 15 14 13	I/O I/O I/O I/O	Port D. Individually selectable in software as general purpose input or output pins.
$\overline{INT1}$	9	I	Highest priority software maskable interrupt
$\overline{INT3}$	8	I	Lowest priority software maskable interrupt
$\overline{RESET}$	10	I	Device reset
MC	26	I	V_{SS} for normal operation
$\overline{XTAL2}$/CLKIN	11	I	Crystal input
$\overline{XTAL1}$	12	O	Crystal output
V_{CC}	17		Supply voltage (positive)
V_{SS}	1		Ground reference

2.3 TMS70Cx2 Devices

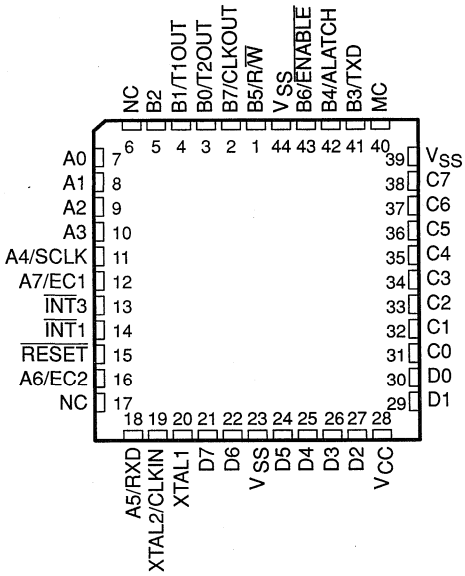
Table 2–4. TMS7000 CMOS Family Feature Summary

	TMS70C40/ TMS70C20/ TMS70C00	TMS70CT40/ TMS70CT20	TMS77C82/ TMS70C82/ TMS70C42/ TMS70C02	TMS70C48/ TMS70C08
Max osc freq at 5V ± 10 %	5 MHz	5 MHz	6 MHz	6 MHz
Voltage	2.5 to 6 V	5 V ± 10 %	2.5 to 6 V	2.5 to 6 V
Operating temperature Industrial Commercial	 –40°C to 85°C 0°C to 70°C	 – 0°C to 70°C	 –40°C to 85°C 0°C to 70°C	 –40°C to 85°C 0°C to 70°C
On-chip ROM (Kbytes)	4 / 2 / 0	4 / 2	8 / 4 / 0 / 8 (EPROM)	4 / 0
Internal RAM (bytes)	128	128	256	256
Interrupt levels: External Total	 2 4	 2 4	 2 6	 2 6
Timers/event counters: 21-bit 13-bit 10-bit	 – 1 –	 – 1 –	 2 – 1	 2 – 1
I/O lines: Bidirectional Input only Output only	16 8 8	12 4 4	24 – 8	46 – 8
Additional features	–	–	Serial Port	Serial Port
Package	40 pin DIP 44 pin PLCC	28 pin DIP –	40 pin DIP 44 pin PLCC	64 pin Flat 68 pin PLCC
Development support: Prototyping: EPROM Piggyback XDS EVM Starter Kit	 – SE70CP160 Yes Yes –	 – SE70CP160† Yes Yes –	 TMS77C82 SE70CP162 Yes Yes Yes	 – SE70CP168 Yes Yes –

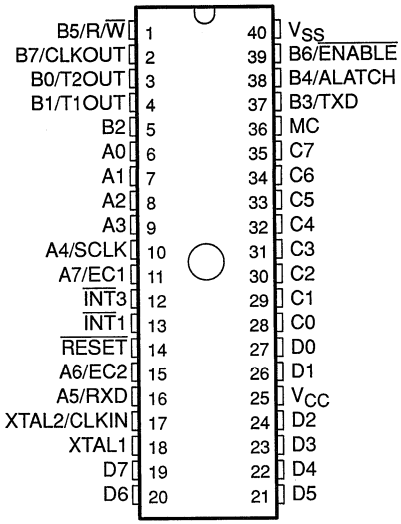
- Flexible on-chip serial port:
  - Asynchronous, isosynchronous, or serial I/O modes
  - Two multiprocessor communication formats
  - Error detection flags
  - Fully software programmable (bits/char, parity, and stop bits)
  - Internal or external baud-rate generator
  - Separate baud-rate timer useable as a third timer
- Memory-mapped ports for easy addressing
- Eight addressing formats
- Two external maskable interrupts and flexible interrupt handling
- Wide voltage operating range, frequency range
- Two power-down modes
  - Wake-up

- Halt
- Silicon-gate CMOS technology
- 40-pin dual-inline package, 44-pin PLCC

Figure 2–3. Pinouts for TMS70C02, TMS70C42 , TMS70C82, and TMS77C82 Devices



A. 44-Pin PLCC Package



B. 40-Pin Plastic DIP

Table 2–5. TMS70Cx2 and TMS77Cx2 Pin Descriptions

Signal	Pin		I/O	Description
	PLCC	DIP		
A0 LSb	7	6	I/O	A0–A4 and A7 are general-purpose bidirectional pins.
A1	8	7	I/O	
A2	9	8	I/O	
A3	10	9	I/O	
A4/SCLK	11	10	I/O	
A5/RXD	18	16	I/O	Data I/O/serial port receiver
A6/EC2	16	15	I/O	Data I/O/serial port clock/Timer 2 event counter
A7/EC1	12	11	I/O	Data I/O/Timer 1 event counter
B0/T2OUT	3	3	O	B0–B3 are outputs. B4–B7 are outputs in single-chip mode and memory interface pins in all other modes. B0 and B1 also contain the timer output functions.
B1/T1OUT	4	4	O	
B2	5	5	O	
B3/TXD	41	37	O	Data output/serial port transmitter
B4/ALATCH	42	38	O	Data output/memory interface address latch strobe
B5/$\overline{R/W}$	1	1	O	Data output/memory interface read/write signal
B6/$\overline{ENABLE}$	43	39	O	Data output/memory interface enable strobe
B7/CLKOUT	2	2	O	Data output/internal clockout
C0	31	28	I/O	Port C is a bidirectional data port. In microprocessor, peripheral-expansion, and full-expansion modes, Port C is a multiplexed low address/data bus.
C1	32	29	I/O	
C2	33	30	I/O	
C3	34	31	I/O	
C4	35	32	I/O	
C5	36	33	I/O	
C6	37	34	I/O	
C7	38	35	I/O	
D0	30	27	I/O	Port D is a bidirectional data port. In microprocessor and full-expansion mode, it is the high address bus.
D1	29	26	I/O	
D2	27	24	I/O	
D3	26	23	I/O	
D4	25	22	I/O	
D5	24	21	I/O	
D6	22	20	I/O	
D7	21	19	I/O	
$\overline{INT1}$	14	13	I	Highest priority external maskable interrupt
$\overline{INT3}$	13	12	I	Lowest priority external maskable interrupt
$\overline{RESET}$	15	14	I	Device reset
MC	40	36	I	Mode control pin, V_{CC} for microprocessor mode
XTAL2/CLKIN	19	17	I	Crystal input for control of internal oscillator
XTAL1	20	18	O	Crystal output for control of internal oscillator
V_{CC}	28	25		Supply voltage (5 V)
V_{SS}	44 39 23	40		Ground reference

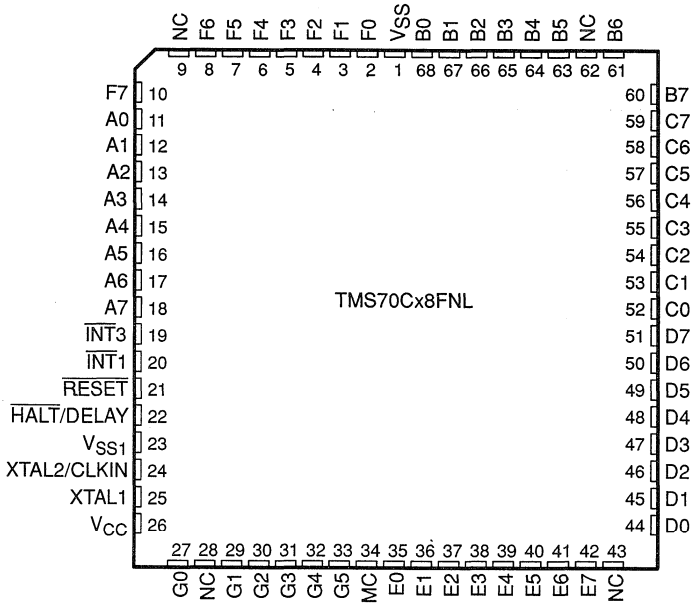
2.4 TMS70Cx8 Devices

2.4.1 TMS70Cx8 Key Features

	TMS70C40/ TMS70C20/ TMS70C00	TMS70CT40/ TMS70CT20	TMS77C82/ TMS70C82/ TMS70C42/ TMS70C02	TMS70C48/ TMS70C08
Max osc freq at 5V ± 10 %	5 MHz	5 MHz	6 MHz	6 MHz
Voltage	2.5 to 6 V	5 V ± 10 %	2.5 to 6 V	2.5 to 6 V
Operating temperature Industrial Commercial	 –40°C to 85°C 0°C to 70°C	 – 0°C to 70°C	 –40°C to 85°C 0°C to 70°C	 –40°C to 85°C 0°C to 70°C
On-chip ROM (Kbytes)	4 / 2 / 0	4 / 2	8 / 4 / 0 / 8 (EPROM)	4 / 0
Internal RAM (bytes)	128	128	256	256
Interrupt levels: External Total	 2 4	 2 4	 2 6	 2 6
Timers/event counters: 21-bit 13-bit 10-bit	 – 1 –	 – 1 –	 2 – 1	 2 – 1
I/O lines: Bidirectional Input only Output only	16 8 8	12 4 4	24 – 8	46 – 8
Additional features	–	–	Serial Port	Serial Port
Package	40 pin DIP 44 pin PLCC	28 pin DIP –	40 pin DIP 44 pin PLCC	64 pin Flat 68 pin PLCC
Development support: Prototyping: EPROM Piggyback XDS EVM Starter Kit	 – SE70CP160 Yes Yes –	 – SE70CP160† Yes Yes –	 TMS77C82 SE70CP162 Yes Yes Yes	 – SE70CP168 Yes Yes –

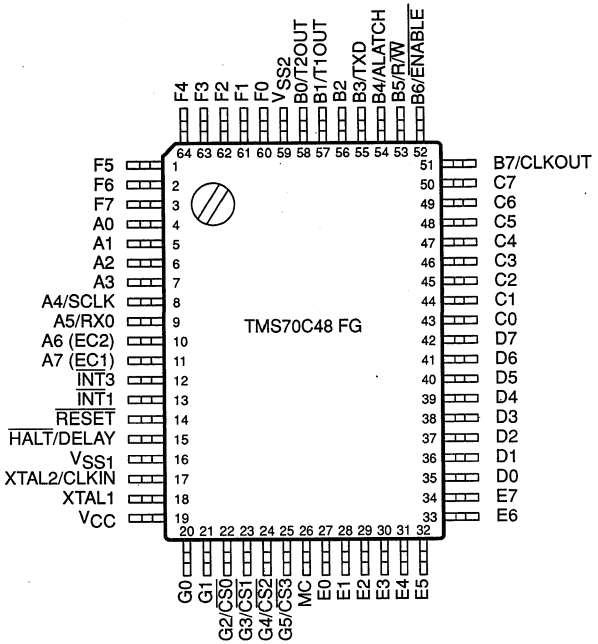
- Register-to-register architecture
- Memory-mapped ports for easy addressing
- Eight addressing formats
- Single-instruction binary-coded decimal (BCD) add and subtract
- Two external maskable interrupts
- Flexible interrupt handling
- Wide voltage operating range, frequency range
- Two power-down modes
- Silicon-gate CMOS technology
- Fully compatible with TMS70Cx8 devices
- 68-pin PLCC package, 64-pin quad flat package

Figure 2–4. Pinouts for TMS70C08 and TMS70C48



A. TMS70Cx8 PLCC Pinout

Figure 2–4. Pinouts for TMS70C08 and TMS70C48 (Concluded)



B. Quad Flat Package

Table 2–6. TMS70Cx8 Pin Descriptions

Signal	Pin	I/O	Description
F0	2	I/O	General-purpose bidirectional pins.
F1	3	I/O	
F2	4	I/O	
F3	5	I/O	
F4	6	I/O	
F5	7	I/O	
F6	8	I/O	
F7	9	I/O	
G0	26	I/O	General-purpose bidirectional pins.
G1	27	I/O	
G2	28	I/O	
G3	29	I/O	
G4	30	I/O	
G5	31	I/O	
$\overline{INT1}$	19	IN	High priority maskable interrupt
$\overline{INT3}$	18	IN	Low priority makable interrupt
$\overline{RESET}$	20	IN	Reset
$\overline{MC}$	32	IN	Memory mode pin
$\overline{HALT}$/DELAY	21	OUT	HALT/DELAY pin or halt mode status pin
XTAL2/CLKIN	23	IN	Crystal input for Control of internal OSC or external CLK input.
XTAL1	24	OUT	Crystal output for control of internal OSC
V_{CC}	25	IN	Supply voltage
V_{SS1}	22	IN	Ground reference 1
V_{SS2}	1	IN	Ground reference 2
A0	10	I/O	A0–A7 are general-purpose bidirectional pins.
A1	11	I/O	
A2	12	I/O	
A3	13	I/O	
A4/SCLK	14	I/O	Data I/O/serial port clock
A5/RXD	15	I/O	Data I/O/serial port receiver
A6(EC2)	16	I/O	Data I/O/Timer 2 event counter input
A7(EC1)	17	I/O	Data I/O/Timer 1 event counter input
B0/T2OUT	64	OUT	B0–B3 outputs. B4–B7 are output in single-chip mode and memory interface pins in all other modes.
B1/T1OUT	63	OUT	
B2	62	OUT	
B3/TXD	61	OUT	Data output/serial port transmitter
B4/ALATCH	60	OUT	Data output/address latch strobe
B5/R/W	59	OUT	Data output/R/W signal
B6/ENABLE	58	OUT	Data output/enable strobe
B7/CLKOUT	57	OUT	Data output/internal clock output
C0	49	I/O	General-purpose bidirectional pins
C1	50	I/O	Multiplexed low address and data bus in PE, FE, and MP modes
C2	51	I/O	
C3	52	I/O	
C4	53	I/O	
C5	54	I/O	
C6	55	I/O	
C7	56	I/O	

Table 2–6. TMS70Cx8 Pin Descriptions (Concluded)

Signal	Pin	I/O	Description
D0	41	I/O	General-purpose bidirectional pins
D1	42	I/O	The high address bus in FE and MP modes.
D2	43	I/O	
D3	44	I/O	
D4	45	I/O	
D5	46	I/O	
D6	47	I/O	
D7	48	I/O	
E0	33	I/O	General-purpose bidirectional pins.
E1	34	I/O	
E2	35	I/O	
E2	36	I/O	
E4	37	I/O	
E5	38	I/O	
E6	39	I/O	
E7	40	I/O	

2.5 SE70CP160, TMS77C82, and SE70CP168 Prototyping Devices

2.5.1 SE70CP160 (CMOS) Piggyback Prototyping Device Key Features

The SE70CP160 supports prototyping development for the TMS70C20, TMS70C40, TMS70CT20, and the TMS70CT40.

	TMS70C40/ TMS70C20/ TMS70C00	TMS70CT40/ TMS70CT20	TMS77C82/ TMS70C82/ TMS70C42/ TMS70C02	TMS70C48/ TMS70C08
Max osc freq at 5V ± 10 %	5 MHz	5 MHz	6 MHz	6 MHz
Voltage	2.5 to 6 V	5 V ± 10 %	2.5 to 6 V	2.5 to 6 V
Operating temperature Industrial Commercial	 –40°C to 85°C 0°C to 70°C	 – 0°C to 70°C	 –40°C to 85°C 0°C to 70°C	 –40°C to 85°C 0°C to 70°C
On-chip ROM (Kbytes)	4 / 2 / 0	4 / 2	8 / 4 / 0 / 8 (EPROM)	4 / 0
Internal RAM (bytes)	128	128	256	256
Interrupt levels: External Total	 2 4	 2 4	 2 6	 2 6
Timers/event counters: 21-bit 13-bit 10-bit	 – 1 –	 – 1 –	 2 – 1	 2 – 1
I/O lines: Bidirectional Input only Output only	16 8 8	12 4 4	24 – 8	46 – 8
Additional features	–	–	Serial Port	Serial Port
Package	40 pin DIP 44 pin PLCC	28 pin DIP –	40 pin DIP 44 pin PLCC	64 pin Flat 68 pin PLCC
Development support: Prototyping: EPROM Piggyback XDS EVM Starter Kit	 – SE70CP160 Yes Yes –	 – SE70CP160† Yes Yes –	 TMS77C82 SE70CP162 Yes Yes Yes	 – SE70CP168 Yes Yes –

† Requires ATC70CT40 interface socket adaptor for pinout conversion (40 pin/28 pin conversion).

- Uses '27C64, '27C128, or compatible EPROMs in a piggyback socket
- Register-to-register architecture
- Memory-mapped ports for easy addressing
- Eight addressing formats, including:
- Single-instruction binary-coded decimal (BCD) add and subtract
- Two external maskable interrupts and flexible interrupt handling
- Wide voltage operating range, frequency range:
  - 2.5 V – 0.8 MHz maximum
  - 6 V – 6.5 MHz maximum
- Two power-down modes:
  - Wake-up (160 μA at 1 MHz typical)

- Halt (10 μA typical)

- Fully compatible with TMS70Cx0 devices and can also be used for prototyping the TMS70CTx0 devices
- Silicon-gate CMOS technology
- 40-pin, 600 mil, dual-inline package

2.5.2 SE70CP168 Piggyback Prototyping Device Key Features

The SE70CP168 supports prototyping development for the TMS70C48.

	TMS70C40/ TMS70C20/ TMS70C00	TMS70CT40/ TMS70CT20	TMS77C82/ TMS70C82/ TMS70C42/ TMS70C02	TMS70C48/ TMS70C08
Max osc freq at 5V ± 10 %	5 MHz	5 MHz	6 MHz	6 MHz
Voltage	2.5 to 6 V	5 V ± 10 %	2.5 to 6 V	2.5 to 6 V
Operating temperature Industrial Commercial	 –40°C to 85°C 0°C to 70°C	 – 0°C to 70°C	 –40°C to 85°C 0°C to 70°C	 –40°C to 85°C 0°C to 70°C
On-chip ROM (Kbytes)	4 / 2 / 0	4 / 2	8 / 4 / 0 / 8 (EPROM)	4 / 0
Internal RAM (bytes)	128	128	256	256
Interrupt levels: External Total	 2 4	 2 4	 2 6	 2 6
Timers/event counters: 21-bit 13-bit 10-bit	 – 1 –	 – 1 –	 2 – 1	 2 – 1
I/O lines: Bidirectional Input only Output only	16 8 8	12 4 4	24 – 8	46 – 8
Additional features	–	–	Serial Port	Serial Port
Package	40 pin DIP 44 pin PLCC	28 pin DIP –	40 pin DIP 44 pin PLCC	64 pin Flat 68 pin PLCC
Development support: Prototyping: EPROM Piggyback XDS EVM Starter Kit	 – SE70CP160 Yes Yes –	 – SE70CP160† Yes Yes –	 TMS77C82 SE70CP162 Yes Yes Yes	 – SE70CP168 Yes Yes –

- Uses '27C64, '27C128, or compatible EPROMs in a piggyback socket
- Flexible on-chip serial port:
  - Asynchronous, isosynchronous, or serial I/O modes
  - Two multiprocessor communication formats
  - Error detection flags
  - Fully software programmable (bits/character, parity, and stop bits)
  - Internal or external baud-rate generator
  - Separate baud-rate timer useable as a third timer
- Register-to-register architecture
- Memory-mapped ports for easy addressing
- Eight addressing formats
- Single-instruction binary-coded decimal (BCD) add and subtract
- Two external maskable interrupts flexible interrupt handling
- Wide voltage operating range, frequency range:
  - 2.5 V – 3.5 MHz maximum

- 6 V – 7.5 MHz maximum
- Two power-down modes:
  - Wake-up
  - Halt
- Fully compatible with TMS70Cx8 devices
- Silicon-gate CMOS technology
- 64-pin, 900 mil, dual-inline package

2.5.3 TMP77C82 JDL CMOS EPROM Prototyping Device Key Features

The TMP77C82JDL supports prototyping development for the TMS70C42 and TMS70C82 directly. With minor software modifications it can also be used for the TMS70C20 and TMS70C40, and with a 28-pin adaptor, for the TMS70CT20 and TMS70CT40.

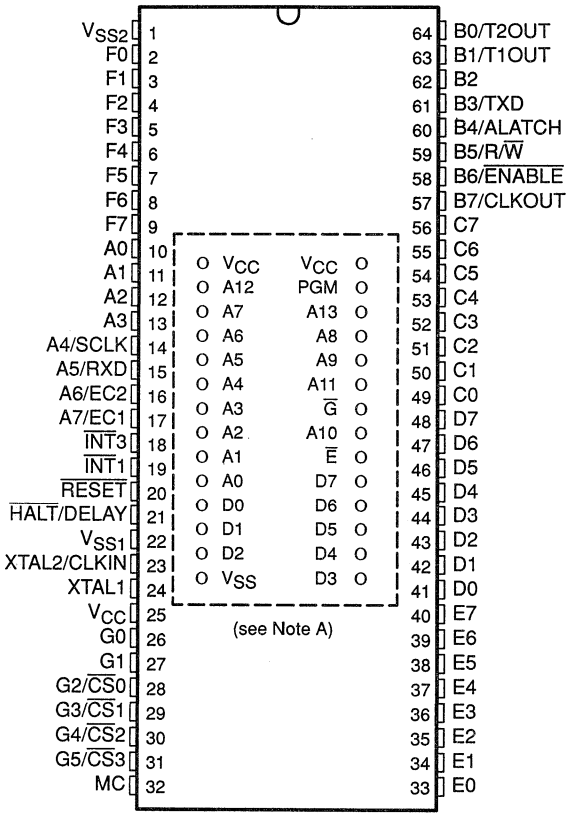
	TMS70C40/ TMS70C20/ TMS70C00	TMS70CT40/ TMS70CT20	TMS77C82/ TMS70C82/ TMS70C42/ TMS70C02	TMS70C48/ TMS70C08
Max osc freq at 5V ± 10 %	5 MHz	5 MHz	6 MHz	6 MHz
Voltage	2.5 to 6 V	5 V ± 10 %	2.5 to 6 V	2.5 to 6 V
Operating temperature Industrial Commercial	 –40°C to 85°C 0°C to 70°C	 – 0°C to 70°C	 –40°C to 85°C 0°C to 70°C	 –40°C to 85°C 0°C to 70°C
On-chip ROM (Kbytes)	4 / 2 / 0	4 / 2	8 / 4 / 0 / 8 (EPROM)	4 / 0
Internal RAM (bytes)	128	128	256	256
Interrupt levels: External Total	 2 4	 2 4	 2 6	 2 6
Timers/event counters: 21-bit 13-bit 10-bit	 – 1 –	 – 1 –	 2 – 1	 2 – 1
I/O lines: Bidirectional Input only Output only	16 8 8	12 4 4	24 – 8	46 – 8
Additional features	–	–	Serial Port	Serial Port
Package	40 pin DIP 44 pin PLCC	28 pin DIP –	40 pin DIP 44 pin PLCC	64 pin Flat 68 pin PLCC
Development support: Prototyping: EPROM Piggyback XDS EVM Starter Kit	 – SE70CP160 Yes Yes –	 – SE70CP160† Yes Yes –	 TMS77C82 SE70CP162 Yes Yes Yes	 – SE70CP168 Yes Yes –

† Requires ATC70CT40 interface socket adaptor for pinout conversion (40 pin/28 pin conversion).

- EPROM programming procedure compatible with '27C64 or '27C128
- Prototyping support for the TMS70C42
- Flexible on-chip serial port:
  - Asynchronous, isosynchronous, or serial I/O modes
  - Two multiprocessor communication formats
  - Error detection flags
  - Fully software programmable (bits/char, parity, and stop bits)
  - Internal or external baud-rate generator
  - Separate baud-rate timer useable as a third timer
- Memory-mapped ports for easy addressing

- Eight addressing formats
- Flexible interrupt handling
  - Priority servicing of simultaneous interrupts
  - Software calls through interrupt vectors
  - Precise timing of interrupts with the capture latch
  - Software monitoring of interrupt status
  - Two external maskable interrupts
- Two power-down modes:
  - Wake-up
  - Halt
- Silicon-gate CMOS technology, 40-pin, 600 mil, dual-inline package

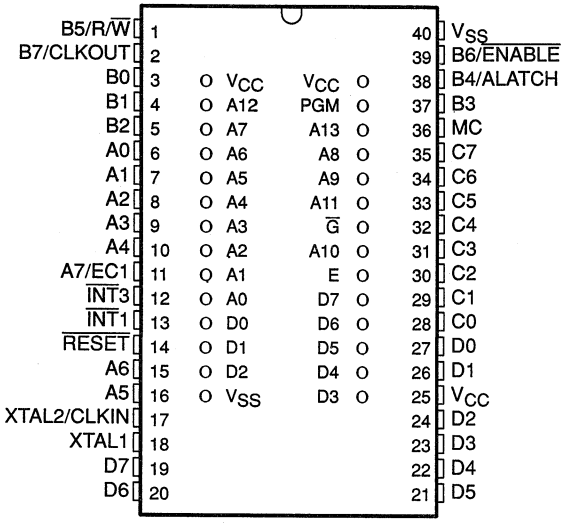
Figure 2–5. SE70CP168 Pinout



Ceramic 64-Pin DIP

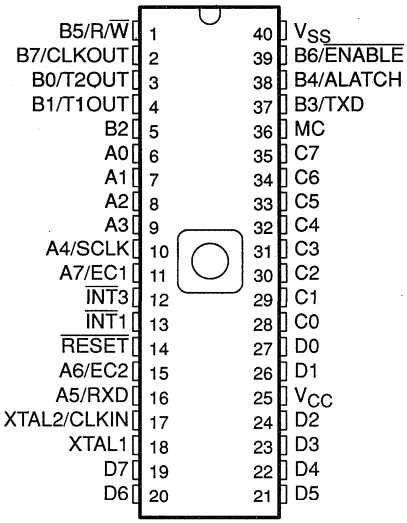
Note A: The broken line surrounds the pinout for the EPROM

Figure 2–6. SE70CP160 Pinout



Ceramic 40-Pin DIP

Figure 2–7. TMP77C82JDL Pinout



A. 40-Pin JD Package

Table 2–7. TMS77C82 Pin Descriptions

Operation Modes					EPROM Mode		
Signal	**Pin No.**		**I/O**	**Description**	**Signal**	**I/O**	**Description**
	PLCC	**DIP**					
A0 LSb	7	6	I/O	A0–A7 are general-purpose bidirectional pins.	A7	I	A3–A7 and A12 are address lines.
A1	8	7	I/O		A6	I	
A2	9	8	I/O		A5	I	
A3	10	9	I/O		A4	I	
A4/SCLK	11	10	I/O	Data I/O/serial port clock	A3	I	
A5/RXD	18	16	I/O	Data I/O/serial port receiver	A12	I	
A6/EC2	16	15	I/O	Data I/O/Timer 2 event counter	$\overline{PGM}$	I	Program
A7/EC1	12	11	I/O	Data I/O/Timer 1 event counter	$\overline{G}$	I	Output enable
B0/T2OUT	3	3	O	B0–B3 are outputs. B4–B7 are outputs in single-chip mode and memory interface pins in all other modes. B0 and B1 are outputs for Timer 2 and Timer 1.			
B1/T1OUT	4	4	O				
B2	5	5	O				
B3/TXD	41	37	O	Data output/serial port transmitter			
B4/$\overline{ALATCH}$	42	38	O	Data output/memory interface address latch strobe			
B5/$\overline{R/W}$	1	1	O	Data output/memory read/write signal			
B6/$\overline{ENABLE}$	43	39	O	Data output/memory interface enable strobe			
B7/CLKOUT	2	2	O	Data output/internal clockout			
C0	31	28	I/O	Port C is a bidirectional data port. In microprocessor, peripheral-expansion, and full-expansion modes, port C is a multiplexed low address and data bus.	Q1	I/O	Q1–Q8 are bidirectional data lines.
C1	32	29	I/O		Q2	I/O	
C2	33	30	I/O		Q3	I/O	
C3	34	31	I/O		Q4	I/O	
C4	35	32	I/O		Q5	I/O	
C5	36	33	I/O		Q6	I/O	
C6	37	34	I/O		Q7	I/O	
C7	38	35	I/O		Q8	I/O	
D0	30	27	I/O	Port D is a bidirectional data port. In microprocessor or full-expansion mode, it is the high address bus.	A8	I	A0–A2 and A8–A11 are address lines.
D1	29	26	I/O		A9	I	
D2	27	24	I/O		A11	I	
D3	26	23	I/O		A10	I	
D4	25	22	I/O		$\overline{E}$	I	Chip enable
D5	24	21	I/O		A0	I	
D6	22	20	I/O		A1	I	
D7	21	19	I/O		A2	I	
$\overline{INT1}$	14	13	I	Highest priority external maskable interrupt			
$\overline{INT3}$	13	12	I	Lowest priority external maskable interrupt			
$\overline{RESET}$	15	14	I	Device reset	GND		V_{SS} for EPROM mode
MC	40	36	I	Mode control pin, V_{CC} for microprocessor mode	V_{PP}		Program enable (12.5 V to program, (0 V to verify)
XTAL2/CLKIN	19	17	I	Crystal input for control of internal oscillator	GND		V_{SS} for EPROM mode
XTAL1	20	18	O	Crystal output for control of internal oscillator			
V_{CC}	28	25		Supply voltage (positive)	V_{CC}		Supply voltage (5 V)
V_{SS}	23 39 44	40		Ground reference	GND		Ground reference

Note: For the programming truth table, refer to Section 9.2 in Chapter 9, *Design Aids.*

Chapter 3

TMS7000 Family Architecture

This chapter discusses the internal architecture of the TMS7000 family devices. Topics in this section include:

Section		Page
3.1	On-Chip RAM and Registers	3-7
3.2	On-Chip General Purpose I/O Ports	3-10
3.3	Memory Modes	3-16
3.4	System Clock Options	3-31
3.5	CMOS Low-Power Modes	3-36
3.6	Interrupts and System Reset	3-39
3.7	Programmable Timer/Event Counters	3-53
3.8	Serial Port (TMS70Cx2, TMS77C82, and TMS70Cx8 Devices Only)	3-68

Figure 3–1 to Figure 3–5 show the major components of the TMS7000 family devices' internal architecture.

Note:

TMS7000 and *TMS7000 family* refer to all TMS7000 devices as described in Chapter 2.

Figure 3–1. TMS77C82 Block Diagram

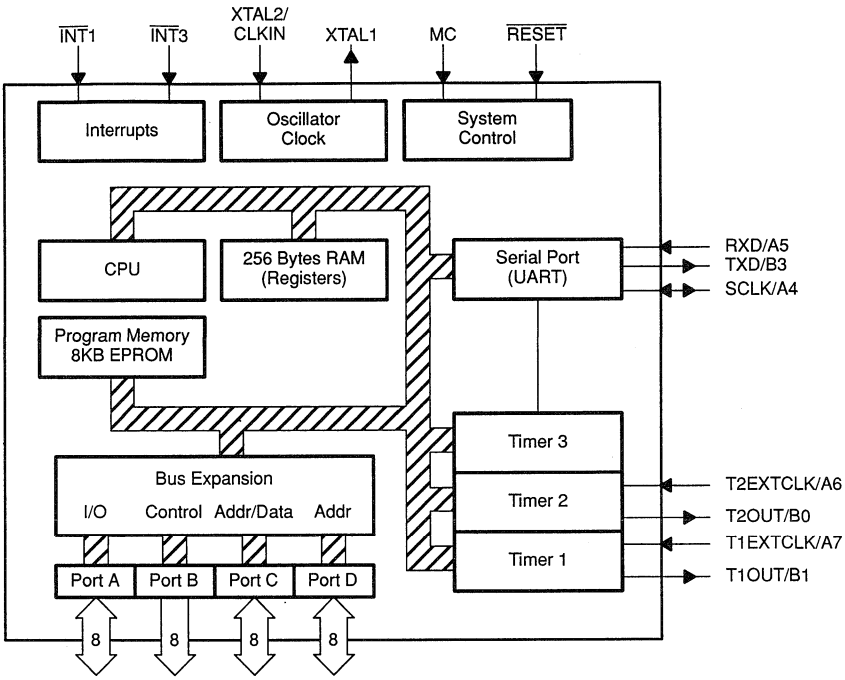


Figure 3–2. TMS70Cx2 Block Diagram

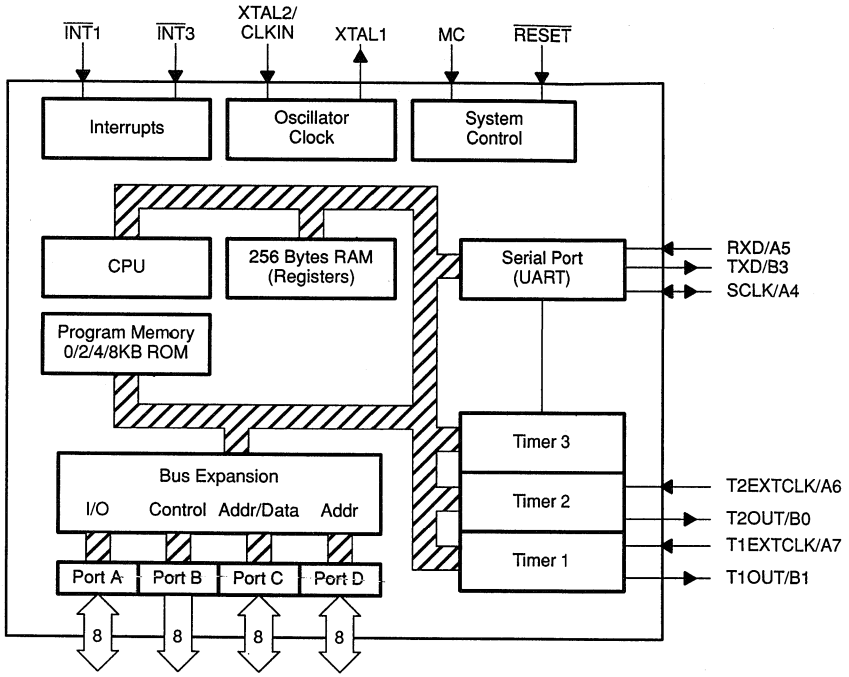


Figure 3–3. TMS70CTx0 Block Diagram

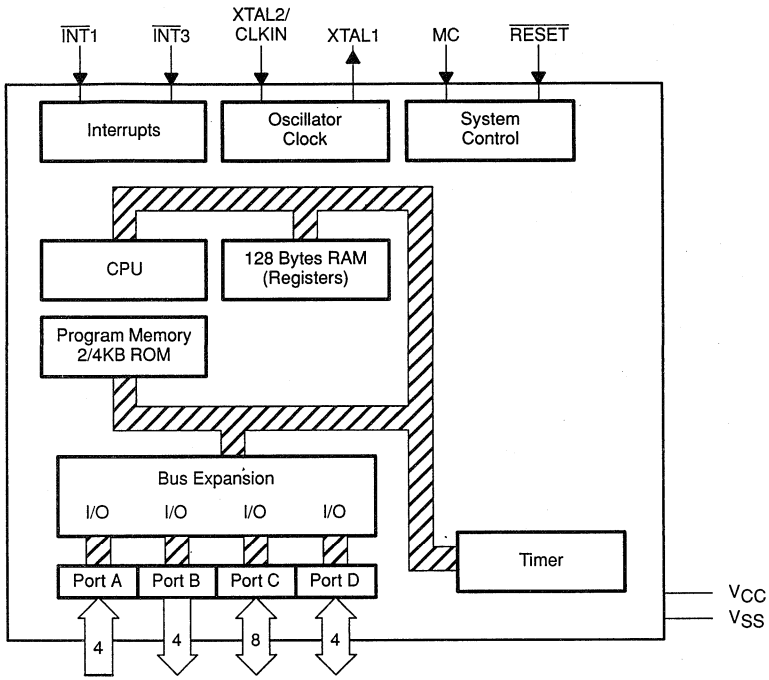


Figure 3–4. TMS70Cx0 Block Diagram

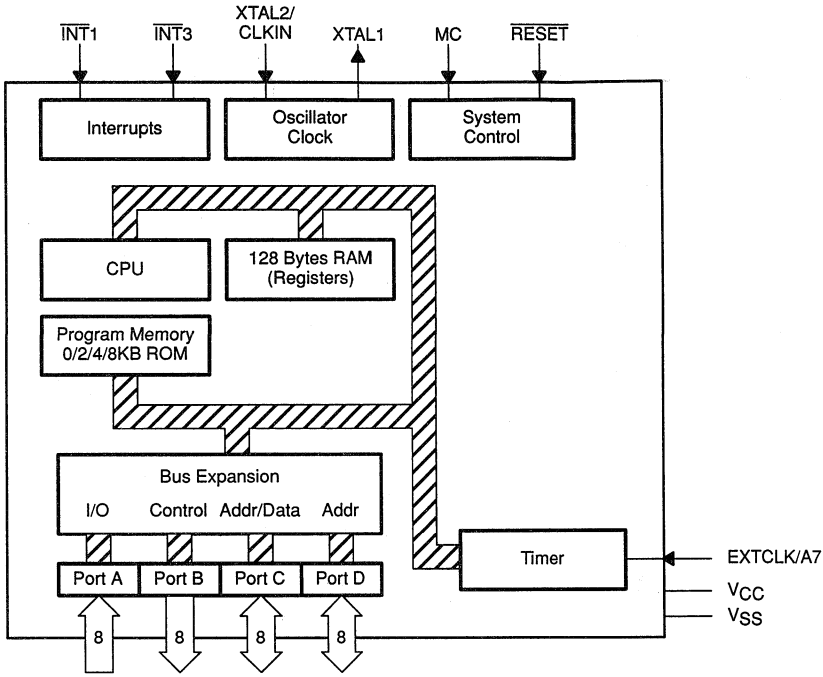
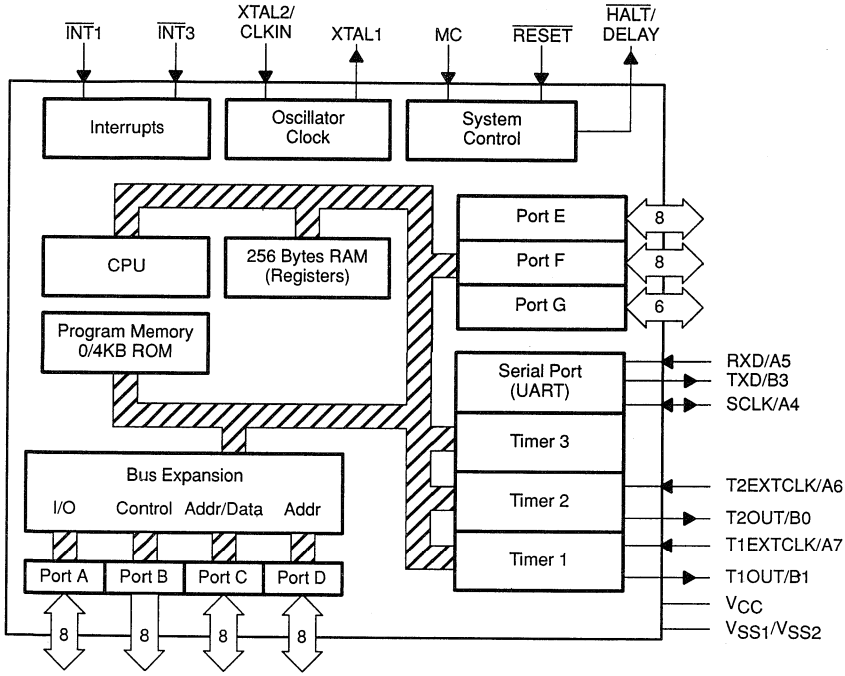


Figure 3–5. TMS70Cx8 Block Diagram



3.1 On-Chip RAM and Registers

TMS7000 family devices have a 64K-byte maximum memory address space. On-chip and off-chip memory address space varies according to the particular family member used and mode selected (see Section 3.2, Memory Modes). The following sections discuss the register file (RF), the peripheral file (PF), and three CPU registers: the stack pointer (SP), the status register (ST), and the program counter (PC).

3.1.1 Register File (RF)

On-chip RAM is called the **register file** (RF). Depending upon the device used, the RF has either 128 or 256 bytes of memory treated as registers R0-R127 or R0-R255. These are located in lower memory as follows:

Device	Number of Registers	Register Range	Memory Address
TMS70Cx0	128	R0–R127	>0000 – >007F
TMS70CTx0	128	R0–R127	>0000 – >007F
TMS70Cx2	256	R0–R255	>0000 – >00FF
TMS77C82	256	R0–R255	>0000 –>00FF
TMS70Cx8	256	R0–R255	>0000 –>00FF

The first two registers, **R0** and **R1**, are also referred to as **Register A** and **Register B**, respectively. Several instructions use register A or B implicitly as either the source or destination register. For example, the STSP instruction stores the contents of the stack pointer in register B. Other instructions may use registers A or B to save memory or increase execution speed. Unless otherwise indicated, any register in the register file can be used as a source or destination register.

3.1.2 Peripheral File (PF)

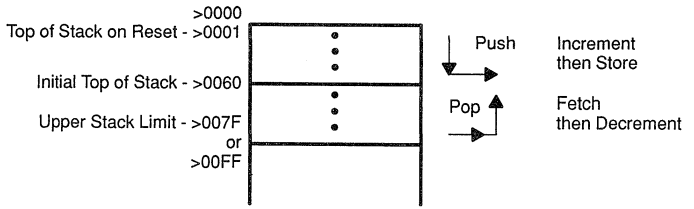
The **peripheral file** (PF) is mapped into locations >0100 to >01FF, which are referred to as P0–P255. These peripheral-file locations contain the 8-bit PF registers, used for interrupt control, parallel I/O ports, timer control, memory-expansion control, and serial port control. All PF addresses not used onboard the TMS7000 are mapped externally in all modes except single-chip. Several instructions, called peripheral-file instructions, communicate with the PF registers, allowing easy use of externally-mapped peripheral devices.

3.1.3 Stack Pointer (SP)

The **stack pointer** (SP) is an 8-bit CPU register that points to the top of the stack. The stack is physically located in the on-chip RAM, or RF. When the stack is used, the SP points to the last or top entry on the stack. During reset, the SP is loaded with >01. The SP is loaded from register B (R1) via the LDSP

instruction and initialized to any other value by executing a stack initialization program such as the one illustrated in Figure 3–6. This feature allows the stack to be located anywhere in the register file. The SP is loaded into register B via the STSP command. The SP is automatically incremented when data is pushed onto the stack and automatically decremented after data is popped from the stack.

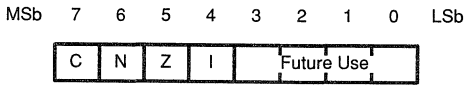
Figure 3–6. Example of Stack Initialization in the Register File



3.1.4 Status Register (ST)

The **status register** (ST) is an 8-bit CPU register that contains three conditional status bits — carry (C), sign (N), zero (Z) — and a global interrupt enable bit (I). The C, N, and Z bits are used for arithmetic operations, bit rotating, and conditional branching.

Figure 3–7. Status Register (ST)



Carry (C) Bit Used as carry-in/carry-out for most rotate and arithmetic instructions.

Negative (N) Bit Contains the most significant bit of the destination operand contents after instruction execution.

Zero (Z) Bit Contains a 1 when the destination operand equals zero after instruction execution.

Global Interrupt Enable (I) Bit
Enables/disables all interrupts. The EINT (enable interrupts) instruction sets this bit to 1; the DINT (disable interrupts) instruction clears it.

This bit must be set to a 1 for interrupts to be acknowledged. However, the individual interrupt flag bits can be set whether this bit is set to a 1 or a 0.

Jump-on-condition instructions are also associated with the C, N, and Z status bits to provide conditional program-flow options.

During reset all bits in the status register are cleared. During other interrupts, the status register is saved on the stack and can be accessed via the PUSHST and POPST instructions.

3.1.5 Program Counter (PC)

The 16-bit **program counter** (PC) consists of two 8-bit registers in the CPU. These registers contain the MSB and the LSB of a 16-bit address: the **program counter high** (PCH) and **program counter low** (PCL).

The PC acts as the 16-bit address pointer of the opcodes and operands in memory of the currently executing instruction. During reset, the MSB and the LSB of the PC are loaded into register A and register B, respectively.

3.2 On-Chip General Purpose I/O Ports

TMS7000 devices have up to 32 I/O pins organized as four 8-bit parallel ports A ,B, C, and D, except TMS70Cx8 which has 3 additional ports E, F, and G, for a total of 54 I/O pins.

Port A — **TMS70Cx0** and **TMS70CTx0** devices – Port A is an input-only port (only port pins A0–A3 available on TMS70CTx0 devices).

TMS70Cx2, **TMS70Cx8,** and **TMS77C82** devices – Port A is fully bidirectional.

Port B — **All devices** – Port B is an output only port (only port pins B4–B7 available on TMS70CTx0 devices).

Port C — **TMS70Cx0**, **TMS70Cx2** and **TMS77C82** – Port C is bidirectional. It is also used as the address/data bus for memory expansion.

TMS70Cx8 – Depending on the mask option, MUX or NMUX, port C is multiplexed addr/data or LSB address bus, respectively. (See subsection 11.1.2, Manufacturing Mask Options). In single chip mode, the port C is a bidirectional I/O port as in the TMS70Cx2.

Port D — **All devices** – Port D is a bidirectional I/O port; it is also used as the address/data bus for memory expansion. (The TMS70CTx0 devices operate in the single chip and in the peripheral expansion mode only, since only port pins D4–D7 are available on port D.)

Port E — **TMS70Cx8** – Depending on the mask option, MUX or NMUX, port E is a bidirectional I/O port, or a data bus for memory and external peripheral device, respectively. In single chip mode, port E remains a bidirectional I/O port regardless of the option.

Port F — **TMS70Cx8** – Port F is bidirectional.

Port G — **TMS70Cx8** – Port C function depends on the mask option I/O or C/S. In I/O option, G is a 6-bit parallel bidirectional port. In the C/S option, the port G data register and the data direction register function identically to the I/O type above, but the most significant 4 bits, G2–G5, cannot be assigned to actual pins because they are used as the chip select signal outputs:

The standard CS-PLA is shown below:

CS0	0100–01FF	for peripheral devices
CS1	4000–7FFF	for up to 16K-byte memory

CS2	8000–BFFF	for up to 16K-byte memory
CS3	C000–FFFF	for up to 16K-byte memory

The port G chip select signals timing is the same as 8-bit MSB address bus timing which exists on port D in full expansion mode or microprocessor mode. During assertion of $\overline{\text{RESET}}$, the CS0–CS3 pins may be kept inactive (high level) without being affected by clock timing.

Ports A, C, D, E, F and G are each controlled and accessed via individual **data-direction registers** and **data registers** in the peripheral file. Output-only port B has only a data register. The *data register* contains the value to be input or output; the *data-direction register* indicates whether the individual port pin is an input or an output. I/O pins can be individually designated as input or output by writing a 1 or 0 to a corresponding bit in their PF data-direction register. A **1** makes the pin an *output*, a **0** makes the pin an *input*.

Writing to the data-direction register does not affect the value in the data register. This allows all bidirectional pins to be used for either input or output by only changing the data-direction register.

During a hardware reset, all data-direction registers are cleared, forcing all bidirectional ports to their high-impedance input state. It is good practice to load Ports A, C, D, E, F, and G data registers before programming any bidirectional bits as outputs. During a hardware reset, port B is set to all 1s.

When any port is configured as an output-only port, applying an external potential to its pins may affect system reliability. The value read at the port pins of Ports C or D will be the same as the last value internally written to the port. However, reading Port B returns the value at the pins, which can override the data written to the port.

Figure 3–8 (page 3-12) shows the logic for each bidirectional I/O line.

Figure 3–8. Bidirectional I/O Logic

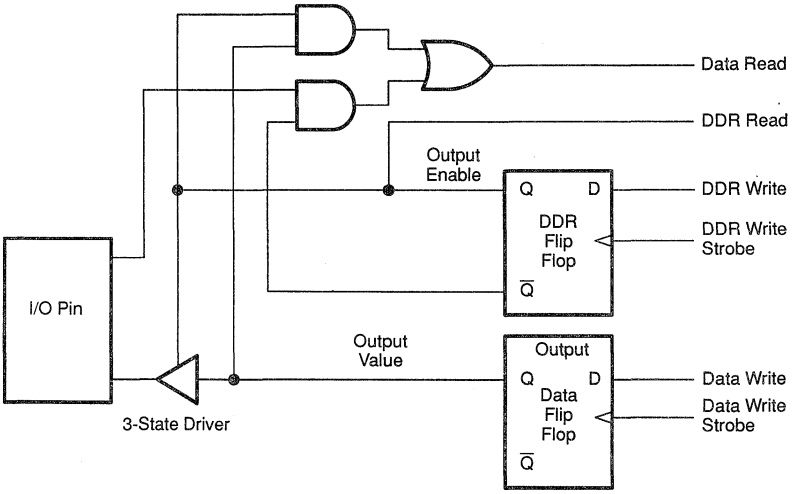


Table 3–1. TMS70Cx0 Port Configuration

I/O	Single-Chip Mode	Peripheral-Expansion Mode	Full-Expansion Mode	Microprocessor Mode
Port A	8 input pins A7=A7/EC1	8 input pins A7=A7/EC1	8 input pins A7=A7/EC1	8 input pins A7=A7/EC1
Port B	8 output pins	4 output pins 4 bus control signals	4 output pins 4 bus control signals	4 output pins 4 bus control signals
Port C	8 I/O pins	8-bit address/data bus	8-bit low address/data bus (LSB)	8-bit low address/data bus (LSB)
Port D	8 I/O pins	8 I/O pins	8-bit high address bus (MSB)	8-bit high address bus (MSB)
Total I/O Pins Available	8 input pins 8 output pins 16 I/O pins	8 input pins 4 output pins 8 I/O pins	8 input pins 4 output pins	8 input pins 4 output pins
Total Memory Pins	None	8 address/data (multiplexed) 4 memory control	16 address/data 4 memory control	16 address/data 4 memory control

Table 3–2. TMS70CTx0 Port Configuration†

I/O	Single-Chip Mode	Peripheral-Expansion Mode
Port A	4 Input Pins	4 Input Pins (A0-A3)
Port B	4 Output Pins	4 Output Pins (B4-B7)
Port C	8 I/O Pins	8 Mux Address/Data Bus
Port D	4 I/O Pins	4 I/O Pins (D4-D7)
Total I/O Pins Available	4 Input Pins 4 Output Pins 12 I/O Pins	4 Input Pins 4 I/O Pins
Total Memory Pins	None	8 Addr/Data Multiplexed 4 Memory Ctl

† The TMS70CTx0 devices operate in the single-chip mode and peripheral-expansion mode only.

Table 3–3. TMS70Cx8 Port Configuration

I/O	Single-Chip Mode	Peripheral-Expansion Mode	Full-Expansion & Microprocessor Modes
Port A	8 I/O Pins A4=A4/SCLK, A5=A5/RXD, A6=A6/EC2, A7=A7/EC1		
Port B	8 Output Pins B3=B3/TXD B0,B1=B0,B1/PWM	4 Output Pins 4 Bus Control signals B0,B1=PWM Output B3=TX	
Port C	8 I/O pins	8 Multiplexed Addr(LSB)/Data Bus	
Port D	4 I/O pins		8 Addr(MSB) Bus
Port E	8 I/O pins	8 I/O Pins	
Port F	8 I/O Pins		
Port G	6 I/O Pins		
Total I/O Pins Available	46 I/O Pins 8 Output Pins	38 I/O Pins 4 Output Pins	30 I/O Pins 4 Output Pins
Total Memory Pins	None	8 Addr/Data Multiplexed 4 Memory Ctl	8 Addr/Data Multiplexed 8 Addr Bus 4 Memory Ctl

Table 3–4. TMS70Cx2 and TMS77C82 Port Configuration

I/O	Single-Chip Mode	Peripheral-Expansion Mode	Full-Expansion Mode	Microprocessor Mode
Port A	8 I/O pins A4=A4/SCLK A5=A5/RXD A6=A6/EC2 A7=A7/EC1	8 I/O pins A4=A4/SCLK A5=A5/RXD A6=A6/EC2 A7=A7/EC1	8 I/O pins A4=A4/SCLK A5=A5/RXD A6=A6/EC2 A7=A7/EC1	8 I/O pins A4=A4/SCLK A5=A5/RXD A6=A6/EC2 A7=A7/EC1
Port B	8 output pins B3=B3/TXD B1=B1/T1OUT B0=B0/T2OUT	4 output pins 4 bus control signals B3=B3/TXD B1=B1/T1OUT B0=B0/T2OUT	4 output pins 4 bus control signals B3=B3/TXD B1=B1/T1OUT B0=B0/T2OUT	4 output pins 4 bus control signals B3=B3/TXD B1=B1/T1OUT B0=B0/T2OUT
Port C	8 I/O pins	8-bit address/data bus	8-bit low address/data bus (LSB)	8-bit low address/data bus (LSB)
Port D	8 I/O pins	8 I/O pins	8-bit high address bus (MSB)	8-bit high address bus (MSB)
Total I/O Pins Available	8 output pins 24 I/O pins	4 output pins 16 I/O pins	4 output pins 8 I/O pins	4 output pins 8 I/O pins
Total Memory Pins	None	8 address/data (multiplexed) 4 memory control	16 address/data 4 memory control	16 address/data 4 memory control

3.2.1 Port A

On **TMS70Cx0** parts, port A is an 8-bit high impedance input-only port, providing eight general purpose input lines. Pin A7/EC1 may also be used to clock the on-chip timer/event counter (see Section 3.7, Programmable Timer/Event Counters).

On **TMS70CTx0** parts, port A is a 4-bit high impedance input-only port, providing four general purpose input lines (A0–A3).

On **TMS70Cx2**, **TMS70Cx8**, and **TMS77C82** devices, port A is a fully bidirectional I/O port. However, pins A5/RXD and A4/SCLK serve as the serial data receive pin and serial clock, respectively, when the serial port is used. Pins A6/EC2 and A7/EC1 may be used to clock the on-chip timer/event counters, Timer 2 and Timer 1, respectively.

3.2.2 Port B

In ***Single-Chip mode***, port B is an 8-bit general-purpose output port. Reading port B returns the value written to the pins unless modified by an external value at the pins. (The TMS70CTx0 devices contain only pins B4-B7.)

In ***all other memory modes***, port B is split into two parts. The lower nibble (pins B0–B3) are general-purpose output-only pins. The most significant

nibble (pins B4–B7) contains the bus control signals: ALATCH, R/$\overline{W}$, $\overline{ENABLE}$ and CLKOUT. (The TMS70CTx0 devices operate in single-chip and peripheral expansion modes only.)

On **TMS70Cx2**, **TMS70Cx8,** and **TMS77C82** devices, pin B3 is also the serial output line (TXD) for the serial port. In addition, a pulse width modulation function exists on both Timer 1 and Timer 2 that allows B1 and B0 outputs, respectively, to be toggled every time the timer decrements through zero (see Section 3.7, TXCTL1 description).

3.2.3 Port C

In ***Single-Chip mode***, port C is an 8-bit bidirectional I/O port. Any of its eight pins may be individually programmed as an input or output line.

In ***all other memory modes***, port C becomes a multiplexed address/data port for the off-chip memory bus. In this case, port C provides the least significant byte of a 16-bit address, followed by eight bits of read or write data. (Port D provides the most significant byte of the 16-bit address.)

The TMS70CTx0 devices operate in single-chip and in the peripheral-expansion modes only.

3.2.4 Port D

In ***single-chip*** or ***peripheral-expansion mode***, port D is an 8-bit bidirectional I/O port. Any of its eight pins may be individually programmed as an input or output line under software control.

In ***full-expansion*** and ***microprocessor modes***, port D becomes a multiplexed address/data port for the off-chip memory bus. In this case, port D provides the most significant byte of a 16-bit address. (Port C provides the least significant byte of the 16-bit address.)

3.2.5 Ports E and F

On TMS70Cx8 parts, ports E and F are fully bidirectional I/O ports in all modes.

3.2.6 Port G

On TMS70Cx8 parts, port G is a fully bidirectional I/O port in all modes. Alternatively, pins G2–G5 can be masked as chip-select pins (see subsection 11.1.2, Manufacturing Mask Options).

3.3 Memory Modes

The TMS7000 can address up to 64K bytes. Four memory modes can be selected by a combination of software and hardware: the **single-chip**, **peripheral-expansion**, **full-expansion**, and **microprocessor modes**. The TMS70CTx0 devices operate in the single-chip and in the peripheral-expansion modes only.

The **mode control** (MC) input pin forces the TMS7000 into Microprocessor mode when set to a V_{CC}. If the MC pin is held at V_{SS}, the remaining memory modes can be selected by bits 6 and 7 of the peripheral file I/O control register (IOCNT0 – P0), as shown in Table 3–5.

Table 3–5. Mode Selection Conditions (MC Pin)

Mode	Mode Select Conditions	
	Mode Control Pin (MC)	IOCNT0 Bits 7,6
Single-Chip	V_{SS}	0 0
Peripheral-Expansion	V_{SS}	0 1
Full-Expansion	V_{SS}	1 0
Microprocessor	V_{CC}	X X

Note: X = Don't care

During reset the IOCNT0 register is set to a 0. (Refer to Section 3.6 for a detailed description of reset and the initialization procedure for the IOCNT0 register.) Table 3–6 and Table 3–7 summarize the four memory modes.

Table 3–6. TMS70Cx0 and TMS70CTx0 Memory Map

Address	Single-Chip Mode	Peripheral-Expansion Mode	Full Expansion Mode	Microprocessor Mode
>0000 >007F	Register File	Register File	Register File	Register File
>0080 >00FF	Reserved	Reserved	Reserved	Reserved
>0100 >010B	On-Chip I/O	On-Chip I/O	On-Chip I/O	On-Chip I/O
>010C >01FF		Peripheral Expansion	Peripheral Expansion	Peripheral Expansion
>0200	Not Available	Not Available	Memory Expansion	Memory Expansion
>F000 >F800 >FFFF	4K ROM 2K ROM	4K ROM 2K ROM	4K ROM 2K ROM	

Table 3–7. TMS70Cx2 and TMS77C82 Memory Map

Address '70C42	Single-Chip Mode		Peripheral Expansion Mode		Full Expansion Mode		Microprocessor Mode	'70C82 '77C82
>0000 >00FF	Register File		Register File		Register File		Register File	>0000 >00FF
>0100 >0123	On-Chip File		On-Chip I/O File		On-Chip I/O File		On-Chip I/O File	>0100 >0123
>0124 >01FF	Not Available		Peripheral Expansion		Peripheral Expansion		Peripheral Expansion	>0124 >01FF
>0200			Not Available		Memory Expansion		Memory Expansion	>0200 >DFFF
>EFFF								>E000
>F000 >FFFF	4K ROM	8K ROM/ EPROM	4K ROM	8K ROM/ EPROM	4K ROM	8K ROM/ EPROM		>FFFF

Table 3–8. TMS70C48 Memory Map

Address	Single-Chip Mode	Peripheral Expansion Mode	Full Expansion Mode	Micro-processor Mode
>0000 >00FF	Register File	Register File	Register File	Register File
>0100 >0121	On-Chip File	On-Chip I/O File	On-Chip I/O File	On-Chip I/O File
>0122 >01FF	Not Available	Peripheral Expansion	Peripheral Expansion	Peripheral Expansion
>0200 >EFFF				Memory Expansion
>F000 >FFFF	4 K ROM	4 K ROM	4 K ROM	

Table 3–9. TMS70Cx0 and TMS70CTx0 Peripheral Memory Map

Port	Address	Name	Single-Chip Mode	Peripheral Expansion Mode	Full Expansion Mode	Micro-processor Mode
P0	>0100	IOCNT0	I/O Control register			
P1	>0101	–	Reserved			
P2	>0102	T1DATA	Timer 1 data			
P3	>0103	T1CTL	Timer 1 control			
P4	>0104	APORT	Port A data			
P5	>0105	–	Reserved			
P6	>0106	BPORT	Port B Data	†		
P7	>0107	–	Reserved			
P8	>0108	CPORT	Port C Data	Peripheral Expansion		
P9	>0109	CDDR	Port C Data-Direction Register			
P10	>010A	DPORT	Port D Data			
P11	>010B	DDDR	Port D Data-Direction Register			
P12-P255	>010C->01FF		Not available	Peripheral Expansion		

† In expansion modes, Port B is referenced in a special manner. See the Port B discussion on page 3-26.

Table 3–10. TMS70Cx2 and TMS77C82 Peripheral Memory Map

Port	Address	Name	Single-Chip Mode	Peripheral-Expansion Mode	Full Expansion Mode	Microprocessor Mode
P0	>0100	IOCNT0	I/O Control register 0			
P1	>0101	IOCNT2	I/O Control register 2			
P2	>0102	IOCNT1	I/O Control register 1			
P3	>0103	–	Reserved			
P4	>0104	APORT	Port A Data			
P5	>0105	ADDR	Port A Data-Direction Register			
P6	>0106	BPORT	Port B data	†		
P7	>0107	–	Reserved			
P8	>0108	CPORT	Port C data	Peripheral Expansion		
P9	>0109	CDDR	Port C Data-Direction Register			
P10	>010A	DPORT	Port D Data			
P11	>010B	DDDR	Port D Data-Direction Register			
P12	>010C	T1MSDATA	Timer 1 MSB decrementer reload register/MSB readout latch			
P13	>010D	T1LSDATA	Timer 1 LSB reload register/LSB decrementer value			
P14	>010E	T1CTL1	Timer 1 control register 1/MSB readout latch			
P15	>010F	T1CTL0	Timer 1 control register 0/LSB capture latch value			
P16	>0110	T2MSDATA	Timer 2 MSB decrementer reload register/MSB readout latch			
P17	>0111	T2LSDATA	Timer 2 LSB reload register/LSB decrementer value			
P18	>0112	T2CTL1	Timer 2 control register 1/MSB readout latch			
P19	>0113	T2CTL0	Timer 2 control register 0/LSB capture latch value			
P20	>0114	SMODE	Serial port mode control register			
P21	>0115	SCTL0	Serial port control register 0			
P22	>0116	SSTAT	Serial port Status Register			
P23	>0117	T3DATA	Timer 3 reload register/decrementer value			
P24	>0118	SCTL1	Serial port control register 1			
P25	>0119	RXBUF	Receiver buffer			
P26	>011A	TXBUF	Transmitter buffer			
P27-P35	>011B->0123		Reserved			
P36-P255	>0124->01FF		Not available	Peripheral Expansion		

Note: In expansion modes, Port B is referenced in a special manner. See the Port B discussion on page 3-26.

Table 3–11. TMS70C48 Peripheral Memory Map

Port	Address	Name	Single-Chip Mode	Peripheral Expansion Mode	Full Expansion Mode	Micro-processor Mode	Reset Value
P0	>0100	IOCNT0	Interrupt Control/Memory Mode Control				00000000
P1	>0101	IOCNT2	Sense & Edge/Level CTL for INT1/INT3				XX00XX00
P2	>0102	IOCNT1	Interrupt Control (INT4/INT5)				XXXX0000
P3	>0103	–	Reserved				
P4	>0104	APORT	Port A Data Value				XXXXXXXX
P5	>0105	ADDR	Port A Direction Register				00000000
P6	>0106	BPORT	Port B Data Value				11111111
P7	>0107	–	Reserved				
P8	>0108	CPORT	Port C Data Value				XXXXXXXX
P9	>0109	CDDR	Port C Direction Register				00000000
P10	>010A	DPORT	Port D Data Value				XXXXXXXX
P11	>010B	DDDR	Port D Direction Register				00000000
P12	>010C	T1MSDATA	R/Timer 1 current timer value (MSB)				
			W/Timer 1 timer latch (MSB)				XXXXXXXX
P13	>010D	T1LSDATA	R/Timer 1 current timer value (LSB)				
			W/Timer 1 timer latch (LSB)				XXXXXXXX
P14	>010E	T1CTL1	R/Timer 1 capture latch (MSB)				
			W/Control register 1				X0XXXXXX
P15	>010F	T1CTL0	R/Timer 1 capture latch (LSB)				
			W/Control register 0				0X0XXXXX
P16	>0110	T2MSDATA	R/Timer 2 current timer value (MSB)				
			W/Timer 2 timer latch (MSB)				XXXXXXXX
P17	>0111	T2LSDATA	R/Timer 2 current timer value (LSB)				
			W/Timer 2 timer latch (LSB)				XXXXXXXX
P18	>0112	T2CTL1	R/Timer 2 capture latch (MSB)				
			W/Control register 1				00XXXXXX
P19	>0113	T2CTL0	R/Timer 2 capture latch (LSB)				
			W/Control register 0				0X0XXXXX
P20	>0114	SMODE	Serial Port Mode Control Register				XXXXXXXX
P21	>0115	SCTL0	Serial Port Control Register 0				01XXX000
P22	>0116	SSTAT	R/Serial Port Status Register				X0XXX101
P23	>0117	T3DATA	R/Timer 3 current timer value				
			W/Timer 3 timer latch				XXXXXXXX
P24	>0118	SCTL1	Serial Port Control Register				X00000XX
P25	>0119	RXBUF	R/Serial Port Receiver Data Buffer				XXXXXXXX
P26	>011A	TXBUF	W/Serial Port Transmitter Data Buffer				XXXXXXXX
P27	>011B	–	Reserved				

Table 3–11. TMS70C48 Peripheral Memory Map (Continued)

Port	Address	Name	Single-Chip	Peripheral Expansion	Full Expansion	Micro-processor	Reset Value
P28	>011C	EPORT	Port E Data Value				XXXXXXXX
P29	>011D	EDDR	Port E Direction Register				00000000
P30	>011E	FPORT	Port F Data Value				XXXXXXXX
P31	>011F	FDDR	Port F Direction Register				00000000
P32	>0120	GPORT	Port G Data Value				XXXXXXXX
P33	>0121	GDDR	Port G Direction Register				XX000000
P34–P255	>0122 >01FF		Not Available	Peripheral Expansion			

3.3.1 Single-Chip Mode

Single-Chip mode is selected when:

MC = V_{SS} and PF Register IOCNT0 = 00XX XXXX

In single-chip mode, the TMS7000 family devices function as standalone microcomputers with no off-chip memory-expansion bus. User memory consists of the RAM register file and ROM. All available I/O lines may be used for various purposes, such as scanning keyboards, driving displays, and controlling other mechanisms. The four ports are configured as shown in Figure 3–9.

Figure 3–9. I/O Ports — Single-Chip Mode

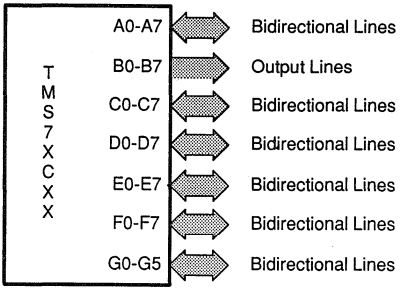
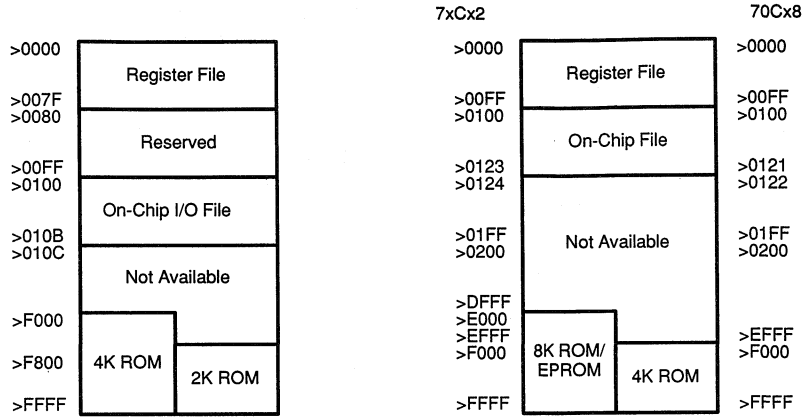


Figure 3–10 shows the single-chip mode memory map. The unused peripheral file (PF) locations and off-chip memory addresses cannot be addressed. If you attempt to read one of these locations, an undefined value is returned. Writing to these addresses has no effect. peripheral-file registers P0–P11 reference the I/O ports and other on-chip functions. Table 3–9, Table 3–10, and Table 3–11 list the peripheral-file registers that are available in single-chip mode.

Figure 3–10. Single-Chip Mode Memory Map



A. TMS70Cx0 and TMS70CTx0 Devices **B. TMS7xCx2 and TMS70Cx8 Devices**

Port A is accessed via PF register **P4** (APORT). When P4 is read, such as with a MOVP (Move from PF) instruction, the value on the port A input pins is returned. The input data is read approximately two machine cycles before the completion of the instruction.

- On the **TMS70Cx0** device, bit 7 (A7) is the MSb and bit 0 (A0) is the LSb. When the on-chip timer/event counter is placed in the external event-counter mode, bit A7/EC1 serves as the external clock input, triggering the event counter on every positive-going transition.
- On the **TMS70CTx0** devices, bit 3 (A3) is the MSb and bit 0 (A0) is the LSb.
- On **TMS70Cx2**, **TMS77C82**, and **TMS70Cx8** devices, all pins are bidirectional I/O pins. Each of these pins can become an input or an output pin, depending upon the value in the data direction register (ADDR) P5.

  P5 bit = 1 Corresponding port A pin becomes an output.

  P5 bit = 0 Corresponding port A pin becomes a high-impedance input.

  Pins A4/SCLK, A5/RXD, A6/EC2, and A7/EC1 have multiple functions. Pins A4/SCLK and A5/RXD are the serial clock I/O pin and the serial data receiver pin, respectively, when the serial port is used. Pins A6/EC2 and A7/EC1 may be used to clock the on-chip timer/event counter, TM2 and TM1, respectively. (See the serial port section for more information, Section 3.8). Pin A6 can also be the external clock input for Timer 2.

  Writing to the data registers will modify the output latch, whether the data direction registers are set for input or output. If set to output, the data latch state will be transferred directly to the pins.

Port B output pins always assert the value of the port B data register, PF register **P6** (BPORT). Writing to P6 loads the port B register, modifying the port B output pins. Reading from P6 provides the current value of the port B pins. When $\overline{\text{RESET}}$ goes active, port B register contents are set to 1s by the on-chip circuitry. (Only pins B4-B7 are available on the TMS70CTx0 devices.)

Port C, Port D (CPORT and DPORT) are bidirectional I/O pins. Data registers are **P8** and **P10** of the peripheral file. Each of these pins can become either an output or an input pin depending upon the value in the port C and D data-direction register, locations P9 and P11 (CDDR and DDDR). A 1 causes an output and a 0 causes a high-impedance input. Writing to the data registers will modify the output latch, whether the data direction registers are set for input or output. If set to output, the data latch state will be transferred directly to the pins. If set to input, the new states will be stored in the output latch and will not affect the states of the input pins. The output latch will be transferred to the pins only if they are subsequently changed back to outputs. Reading the data register returns either the current value at the pin (when the pin is an input) or the current value of the data register (for pins configured as outputs). Refer to Figure 3–8 (page 3-12) for a diagram of the bidirectional I/O logic. (Port pins C0-C7 and D4-D7 are available on the TMS70CTx0 devices.)

Ports E, F, G Bidirectional I/O pins for the TMS70Cx8 devices only. Data registers are P28, P30 and P32. Data direction registers are P29, P31 and P33 respectively. The information concerning ports C and D above apply similarly to E, F, and G.

Peripheral-file instructions ANDP, ORP, and XORP perform a read/modify/write cycle on PF registers. When applied to a port's data register, these instructions can clear, set, or complement the output pins on the port.

The following program segment illustrates a use of the I/O lines in the single-chip mode for most family members. (The TMS70CTx0 devices do not contain all 32 I/O pins initialized below.)

```
IOCNT0   EQU    P0              I/O control register 1
APORT    EQU    P4              Port A data register
BPORT    EQU    P6              Port B data register
CPORT    EQU    P8              Port C data register
CDDR     EQU    P9              Port C data-direction register
DPORT    EQU    P10             Port D data register
DDDR     EQU    P11             Port D data-direction register
*
RESET    MOVP   %>3F,IOCNT0     Set Single-Chip mode, enable all
*                               interrupts, clear all pulse
*                               flip-flops
L1       MOVP   %>02,DPORT      Load Port D with 0000 0010
*                               (D7-D0)
L2       MOVP   %>00,CPORT      Load Port C with 0000 0000
*                               (C7-C1)
         MOVP   %>F0,CDDR       Config C7-C4 outputs, C3-C0 inputs
         MOVP   %>0F,DDDR       Config D7-D4 inputs, D3-D0 outputs
         ORP    %>04,DPORT      Set pin D2 to 1
         ANDP   %>7F,CPORT      Clear pin C7
```

```
BTJZP   %>08,CPORT,L1   Jump if C3 is 0
MOVP    %>55,BPORT      Set Port B to 0101 0101 (B7-B0)
XORP    %1,BPORT        Toggle bit B0
BTJOP   %>41,APORT,L2   Jump if either A6 or A1 is a 1
```

Note:

The percent sign (%) indicates the immediate addressing mode. The instruction set is described in Chapter 6.

3.3.2 Peripheral-Expansion Mode

Peripheral-expansion mode is selected when:

MC = V_{SS} and PF Register IOCNT0 = 01XX XXXX

Peripheral-expansion mode incorporates features of both the I/O-intensive single-chip mode and the memory-intensive full-expansion mode. References to peripheral-file addresses (locations >0100 to >01FF) not corresponding to on-chip PF registers produce off-chip memory cycles. During peripheral-file instructions, a PF port is read, even if the value is not needed, such as in a `MOVP A,P6`. If a hardware configuration makes this read undesirable, use a STA (store A) instruction with the memory-mapped address of the PF register. The ability to reference off-chip addresses allows the TMS7000 to be directly connected to most of the popular peripheral devices developed for 8-bit microprocessors. The TMS7000 PF instructions reference these off-chip peripherals just as easily as they access on-chip PF registers.

Figure 3–11. I/O Ports — Peripheral-Expansion Mode

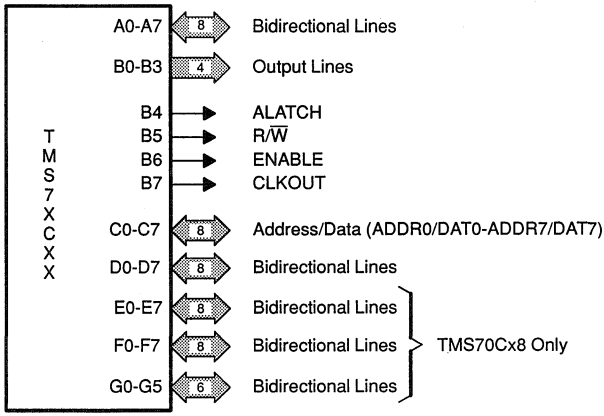
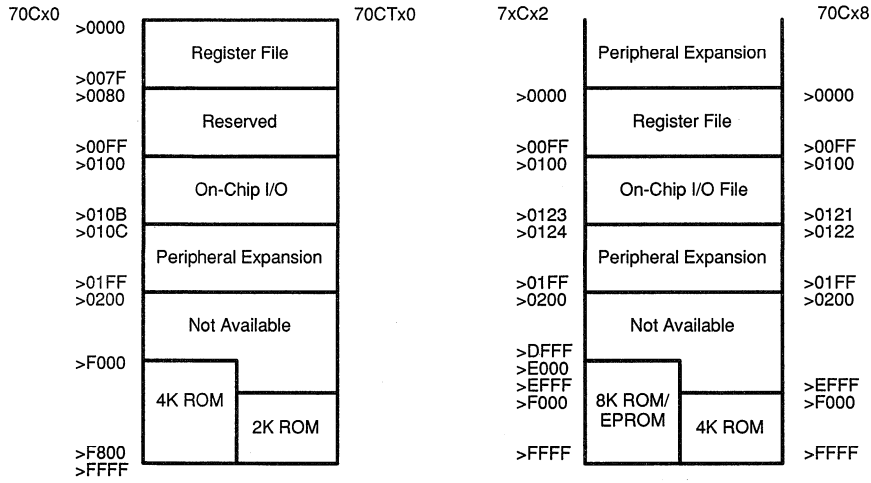


Figure 3–12. Peripheral-Expansion Mode Memory Map



A. TMS70Cx0 and TMS70CTx0 Devices

B. TMS70Cx2, TMS77C82, and TMS70Cx8 Devices

Port A functions the same as in single-chip mode.

Port B is divided into two sections: pins B0–B3 function as individual output pins, the same as in single-chip mode; pins B4–B7, however, function as external memory bus controls:

- Pin B4/ALATCH is strobed to logic 1 while port C asserts the memory address.
- Pin B5/R/$\overline{W}$ is driven to logic 1 for a read cycle and to logic zero for a write cycle.
- Pin B6/$\overline{ENABLE}$ is asserted at logic 0 whenever an external memory cycle is in progress.
- Pin B7/CLKOUT is an output clock intended for general memory control timing.

Exact signal timing is described in Chapter 4.

References to the port B data register, P6, are handled in a special manner. When a value is written to P6, pins B0–B3 output the new value. Pins B4–B7 ignore the new value and continue to output memory bus signals. An external memory write cycle will also write the entire 8 bits of the new value to the external address >0106. When P6 is read, the least significant nibble (B0–B3) is taken from the current value on pins B0–B3. The most significant nibble is obtained by reading the external address >0106.

Port C functions as a multiplexed address/data port for the memory-expansion bus. In normal configurations, port C is attached to the input of an 8-bit latch such as an SN74LS373. The B4/ALATCH signal drives the G input of the latch, so that the latch's Q outputs follow the D inputs while B4/ALATCH is high, and ouputs become latched when it falls. After B4/ALATCH falls and data (such as a memory address) is latched, port C either becomes a high-impedance input for read cycles or it asserts the output data for write cycles.

Ports D, E, F, and G function identically to a bit-programmable, bidirectional I/O port, as in the single-chip mode.

Note:

1) The port C data-direction register is mapped into external memory. The port C input or output function can be recreated externally by mapping a latch at location >0108.
2) Because B4/ALATCH, B5/R/$\overline{W}$, and port C are active for both external and internal (ROM and RAM) memory cycles, it is recommended that B6/$\overline{ENABLE}$ be gated with the chip-select input of all external memory devices to prevent external bus conflicts.

3.3.3 Full-Expansion Mode

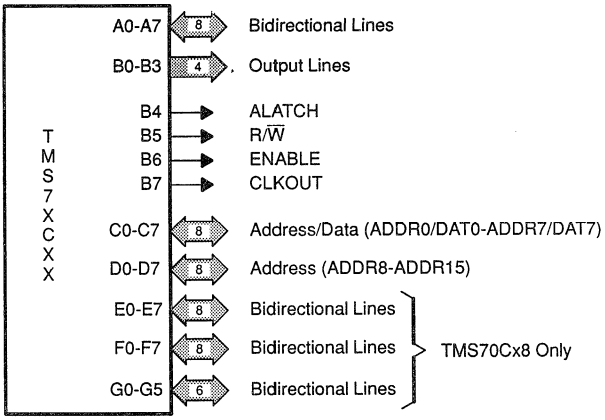
Full-expansion mode is selected when:

MC = V_{SS} and PF Register IOCNT0 = 10XX XXXX

Full-expansion mode uses a 16-bit address to extend the memory addressing capability of the TMS7000 to its full 64K-byte limit. External memory may be accessed with instructions using the direct, register file indirect, and indexed addressing modes of the instruction set. This meets a variety of application requirements by expanding the external program or data storage.

Full-expansion mode I/O is identical to the peripheral-expansion mode *except that port D is used to output the most significant byte (MSB) of the 16-bit address.* Thus, port D is not available as an I/O port. The four ports are configured as shown in Figure 3–13. Figure 3–14 shows the I/O memory assignments for the full-expansion mode.

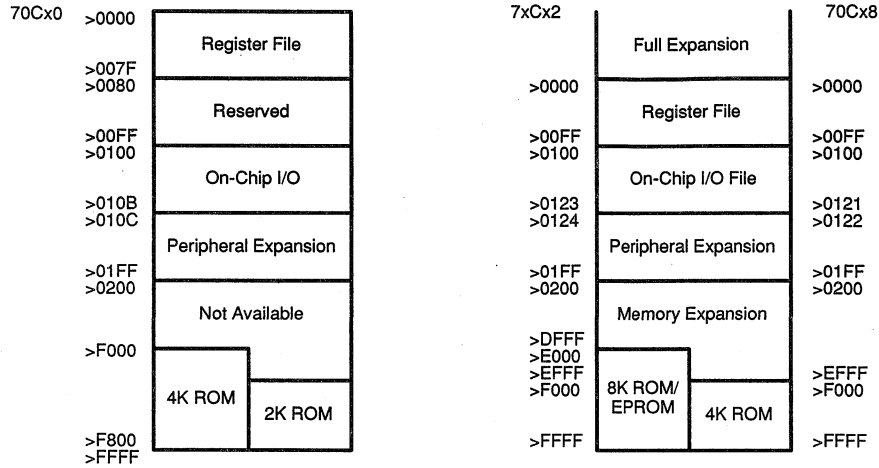
Figure 3–13. I/O Ports — Full-Expansion Mode



TMS70Cx2, TMS77C82, and TMS70Cx8 Devices

As in the peripheral-expansion mode, accesses to peripheral-file registers (locations >0100 to >01FF) which are not directly implemented as on-chip registers produce off-chip memory cycles. The on-chip peripheral-file registers are listed in Table 3–9 through Table 3–10. Note that the port D data register (DPORT) and the port D data-direction register (DDDR) are implemented as off-chip addresses in the full-expansion mode. The port D input or output function can be recreated externally by mapping a latch at location >010A.

Figure 3–14. Full-Expansion Mode Memory Map



A. TMS70Cx0 Devices

B. TMS70Cx2, TMS77C82, and TMS70Cx8 Devices

3.3.4 Microprocessor Mode

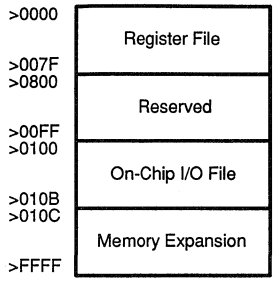
Microprocessor mode is selected when:

MC = V_{CC} and PF Register IOCNT0 = XXXX XXXX

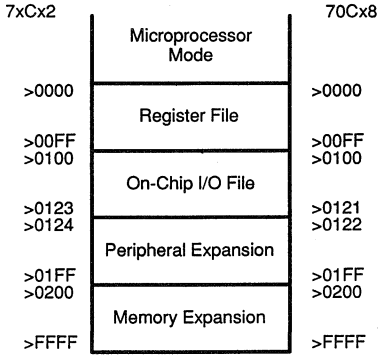
Microprocessor mode is intended for applications that do not justify the use of on-chip ROM. The port pins are configured exactly as in full-expansion mode (see Figure 3–13). Unlike full-expansion mode, no on-chip ROM is referenced in microprocessor mode. All memory accesses except for internal RAM and on-chip peripheral-file locations are now addressed externally.

The MC pin must be held at logic 1 (V_{CC}) to place the device in this mode. There are no restrictions on when the value of the MC pin may change, but it is recommended that the value be changed only when the device is in reset. Indeterminate results can occur if the MC pin is changed while the device is accessing memory locations whose internal/external status may change.

Figure 3–15. Microprocessor Mode Memory Map



A. TMS70Cx0 Devices



B. TMS70Cx2, TMS77C82, and TMS70Cx8 Devices

3.4 System Clock Options

The internal state cycle period, called $t_{c(c)}$, is derived from the clock system that is applied on the XTAL pins of the circuit. The internal clock then divides the external clock source frequency by two to produce the internal state frequency; for example, a 5 MHz crystal produces an internal frequency of 2.5 MHz, which drives a 400-ns machine cycle. TMS7000 devices can use a crystal, ceramic resonator, or another approximately 50% duty cycle clock as an external clock source. If the TMS7000 contains the RC mask option (low power mode, see subsection 3.4.2), it shall use an R-C circuit or a ceramic resonator in the range described in subsection 3.4.1.

3.4.1 System Clock Connections

The TMS7000 devices use the following methods to implement the system clock options:

Crystals: Parallel resonant crystals are connected between pins XTAL1 and XTAL2/CLKIN. To optimize the crystal waveform unbalanced capacitors should be connected, 15-pF between XTAL1 and Ground, and 33 pF between XTAL2 and Ground. The crystal and components should be mounted as close as possible to the input pins to minimize output distortion and start-up stabilization time. This connection is illustrated in Figure 3–16a. Crystals can be used with XTAL mask option only.

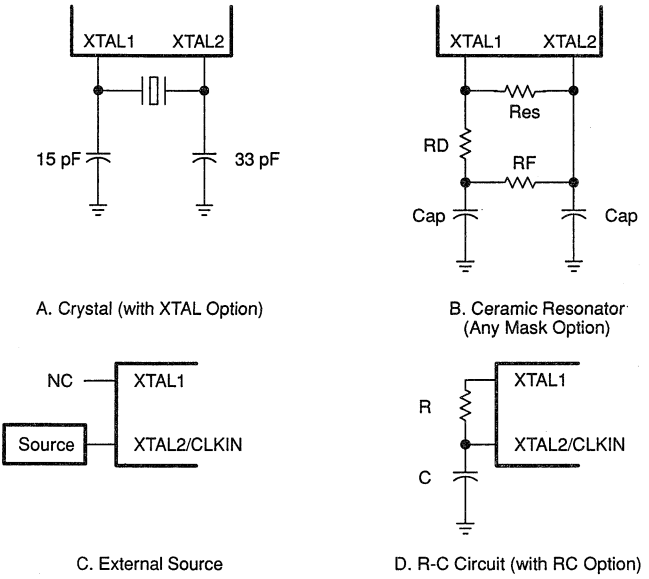
Ceramic Resonators:

Ceramic resonators are connected between pins XTAL1 and XTAL2/CLKIN. A resistor and two capacitors, with values determined by the selected ceramic resonator, must be connected as shown in Figure 3–16b. Values vary by manufacturer and type.

Ceramic resonators can be used with both XTAL and RC mask options. The following values are recommended by the Murata manufacturer:

Mask Option	Ceramic Resonator	Frequency (Hz)	RF (MOhm)	RD (KOhm)	Cap (pF)	V_{CC} Range (V)
RC	CSB600P	600 K	1	2.7	150	2.0 — 5.5
	CSB1100JT	1100 K	1	2.7	100	2.0 — 5.5
	CSA3.58MG	3.58 M	1	0	30	3.5 — 5.5
	CSA8.00MT	8 M	1	0	22	4.0 — 5.5
XTAL	CSB500E	500 K	1	5.6	100	2.0 — 5.5
	CSA5.00MG	5 M	1	0	30	4.0 — 5.5
	CSA8.00MT	8 M	1	0	30	4.0 — 5.5

Figure 3–16. System Clock Connections



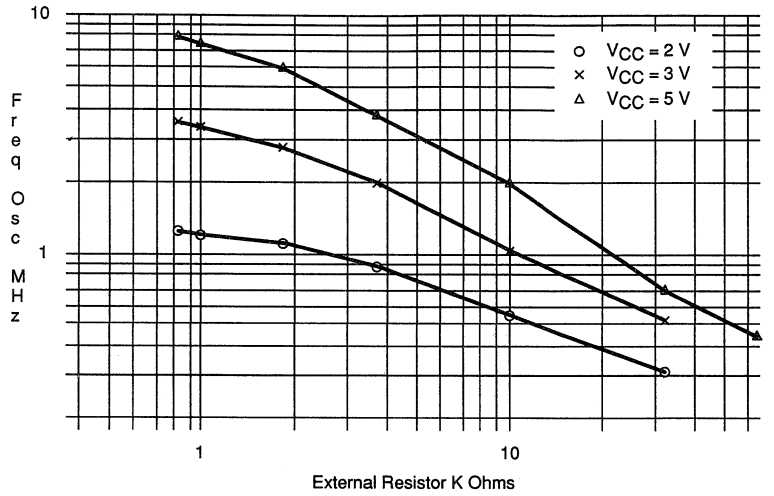
External Clock Source:

As shown in Figure 3–16c, external clock sources are connected to XTAL2/CLKIN, and XTAL1 is not connected. External clock sources can be used with both XTAL and RC mask options.

R-C Circuits: R-C circuits provide a simple, low-cost oscillator for applications in which frequency toleration is not a concern. R-C circuits also provide immediate start-up oscillation upon exiting the halt-off mode operation (see subsection 3.4.2).

R-C circuits are connected as shown in Figure 3–16d. The recommended value for the capacitor C is 47 pF. The value of the resistor R required for the desired frequency must be selected with respect to V_{CC}, ambient temperature, and the tolerance of the R-C components. Recommended values for the resistor in the R-C network fall in the range of 1KΩ–100KΩ. Refer to Figure 3–17 for the frequency/resistance relationships. R-C circuit can be used with R-C mask option only.

Figure 3–17. Frequency Versus Resistance

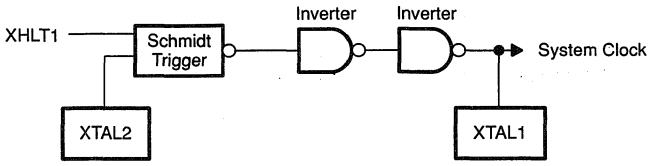


NOTE: TMS7xCxx RC Option/C = 47 pF.

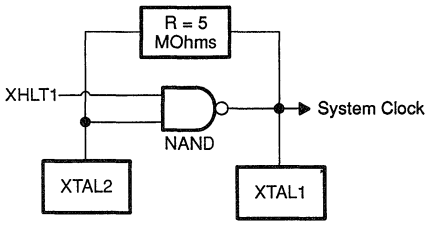
3.4.2 Low-Power Mask Option

The TMS7000 devices may use oscillator mask options which provide different levels of functionality and power consumption during the halt low-power mode. These oscillator options are called RC/OSC-Off and XTAL/OSC-On. The TMS70Cx8 devices offer a third oscillator option called ceramic. For these three options, the internal-clock-circuit block diagrams are shown in Figure 3–18.

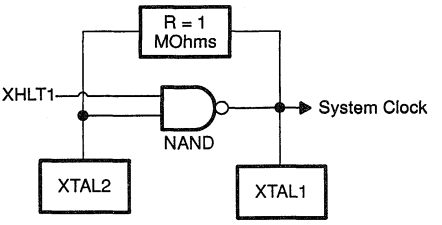
Figure 3–18. Internal Clock Circuit Block Diagrams



A. R-C Option Internal Clock Circuit



B. XTAL Option Internal Clock Circuit



C. Ceramic Option Internal Clock Circuit

The OSC-On options (XTAL) will keep the on-chip oscillator active during the halt mode. When the device is brought out of halt mode, there will be no delay in restoring the full operation since the oscillator is already running. This option is useful in applications where no delay in restoring full operation after halt mode is more important than the lower power consumption.

The OSC-Off option (RC) is useful in applications where very low power consumption is required in halt mode. This option causes the oscillator to cease oscillation when halt mode is entered. This offers the lowest power consumption typically below 1μA. The RC mask programmable option supports an R-C circuit as well as a ceramic resonator or other external 50% duty cycle CLKIN signal. If an R-C network is used, it will restart full oscillation immediately upon exiting halt mode. If a ceramic resonator is used, there will be a period before the oscillations stabilize, causing a delay in the response to reset of about 10 milliseconds.

In the TMS70Cx8 devices, the *ceramic* option can cause the oscillator to cease oscillation when the halt mode is entered. When the device is brought out of halt mode, the clock start-up delay can be set with the $\overline{\text{HALT}}$/Delay pin capacitance value. Therefore in the TMS70Cx8 devices we recommend to use the ceramic option rather than the RC option as an OSC-Off solution with a ceramic resonator.

Note:

RC, XTAL and ceramic are mask options, which means the option is placed on a manufacturing template, or mask, that copies the actual circuit onto the silicon device. This means the oscillator option is finalized at the start of manufacture and cannot be changed by software or hardware.

3.5 Low-Power Modes

The TMS7000 CMOS microcomputers can be programmed to enter low-power modes of operation (wake-up and halt) when the IDLE instruction is executed. For information concerning mask options associated with the halt low-power mode, see subsection 3.4.2.

3.5.1 TMS70Cx0 and TMS70CTx0 Low-Power Modes

The TMS70Cx0 and TMS70CTx0 devices support the wake-up and halt low-power modes. These modes are entered when:

1) Bit 5 of the Timer 1 control register (T1CTL) is set (0 for wake-up mode, 1 for halt mode), **and**
2) The IDLE instruction is executed.

Activating $\overline{\text{RESET}}$ or acknowledging an enabled interrupt releases the device from either mode except for the halt OSC-Off mode where the use of either a crystal or ceramic resonator requires activating $\overline{\text{RESET}}$. Both low-power modes freeze the I/O ports, retaining their conditions before the IDLE instruction was executed. Complete RAM data retention is also maintained through both low-power modes as long as power is applied. Table 3–12 describes the low-power options.

Table 3–12.Low-Power Options for TMS70Cx0 and TMS70CTx0 Devices

Mode	CPU Status	Timer 1 Status	OSC Status	Enter Mode Via	Exit Mode Via	Clock Source
Wake-Up	Halted	Active	Active	IDLE	RESET, INT1, INT2, INT3 (if enabled)	Crystal, R-C Circuit, Ceramic Resonator, External Clock
Halt (XTAL)	Halted	Halted	Active	IDLE	RESET,INT1, INT3 (if enabled)	Crystal, Ceramic Resonator, External Clock
Halt (RC)	Halted	Halted	Halted	IDLE	RESET, INT1, INT3 (if enabled)	R-C Circuit, External Clock
Halt (RC)	Halted	Halted	Halted	IDLE	RESET	Ceramic Resonator

In wake-up mode, the oscillator and timer logic remain active. The on-chip timer may be used to release the device from the low-power state. The I_{CC} current requirements in wake-up mode are frequency dependent for both the OSC-On and the OSC-Off options.

3.5.2 TMS70Cx2 Devices

The TMS70Cx2 devices support the wake-up and halt-low power modes. These modes are entered when the IDLE instruction is executed. An enabled interrupt must be executed to allow the device to return to normal operation. The TMS70Cx2 devices have the ability to disable the individual onboard timers during the low power modes. In halt-OSC-Off mode, to ensure that the oscillator is halted, the following bits in the respective registers must be programmed.

Register	Binary Value	
T1CTL0	xx1xxxxx	x= don't care
T2CTL0	xx1xxxxx	
SCTL0	10xxxxxx	

The important point to note is that bit 6 of SCTL0 must be at *zero* which enables the serial port and not at 1 which resets the serial port.

By definition, whenever the Timer 1, Timer 2, or Timer 3 are active when the IDLE instruction is executed, the device is in a wake-up mode. When, and only when the above register configuration is set, and when the IDLE is executed, the device is in halt mode.

Table 3–13. Low-Power Options for TMS7xCx2 and TMS70Cx8 Devices

Mode	CPU Status	T1/T2/T3 UART	OSC Status	Enter Via	Exit Mode Via	Clock Source
Wake-up	Halted	Each programmed active or halted	Active	Idle	RESET, INT1, INT2, INT3, INT4, INT5 if enabled	Crystal, R-C circuit, ceramic resonator, external clock
Halt (XTAL)	Halted	Halted TIHALT = 1 T2HALT = 1 SPH = 1, and UR=0	Active	Idle	RESET, INT1, INT3 if enabled	Crystal, Ceramic Resonator, External Clock
Halt (RC)	Halted	Halted T1HALT = 1 T2HALT = 1 SPH = 1, and UR=0	Halted	Idle	RESET, INT1, INT3 if enabled	R-C Circuit External Clock
Halt (RC)	Halted	Halted T1HALT = 1 T2HALT = 1 SPH = 1, and UR=0	Halted	Idle	RESET	Ceramic Resonator (for 7xCx2) External Clock
Halt (Ceramic)	Halted	Halted T1HALT = 1 T2HALT = 1 SPH = 1, and UR=0	Halted	Idle	RESET	Ceramic Resonator (for 7xCx8) External Clock

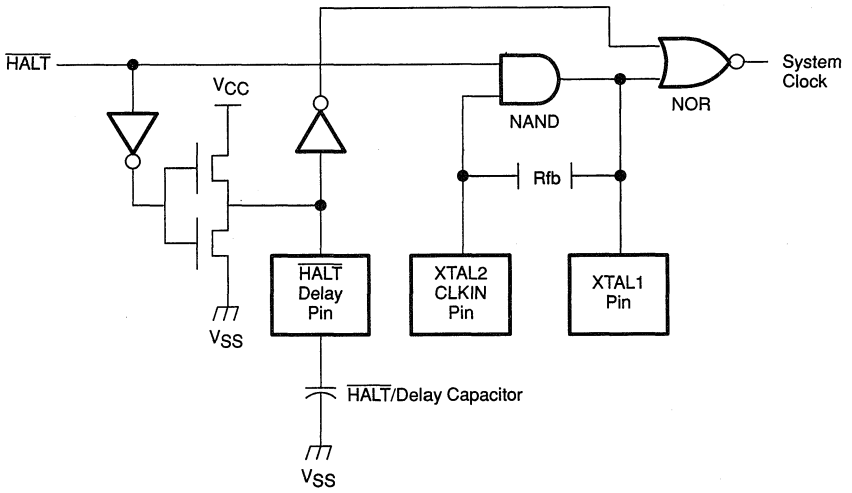
3.5.3 TMS70Cx8 Devices

The TMS70Cx8 supports the two low-power modes, using the same T1CTL0, T2CTL0 and SCTL0 register programming as for the TMS70Cx2.

3.5.3.1 $\overline{\text{HALT}}$/DELAY Pin

To avoid function error by an unsettled oscillation, the TMS70Cx8 provides the $\overline{\text{HALT}}$/Delay pin. This output pin is also used as status pin which indicates the halt mode set or not. This pin keeps low during IDLE in halt mode. At wake-up or active modes, this pin keeps high, and is capable of driving an LS-TTL gate.

Figure 3–19. $\overline{\text{HALT}}$/DELAY Block Diagram



1) Determine the capacitor at the $\overline{\text{HALT}}$/Delay pin using a graph or a formula of capacitance versus delay time.

Tr	I	Capacitance Value			Conditions
V_{CC} voltage	I	10 µF	3.3 µF	1 µF	All timings in ms
5 V— 5.5 V	I	2.4	1	0.26	Room Temperature; Tr is not affected by F_{OSC}
4 V	I	3	1.2	0.37	
3 V	I	5	1.8	0.55	

The rise time Tr, is the delay between the interrupt to release halt mode and the clockout from B7.

2) Connect this capacitor between the $\overline{\text{HALT}}$/Delay pin and V_{SS} pin.

3.6 Interrupts and System Reset

All TMS7000 family devices have a non-maskable system reset pin, $\overline{\text{RESET}}$. This signal has the highest priority in the interrupt hierarchy. $\overline{\text{RESET}}$ immediately initializes the device.

The **TMS70Cx0** and **TMS70CTx0** devices have three separate, maskable interrupts that are triggered from three sources. The **TMS70Cx2**, **TMS77C82**, and **TMS70Cx8** devices have five separate maskable interrupts that can be triggered from as many as seven sources. Each interrupt has a specific priority level; if two or more interrupts occur simultaneously, they are serviced according to priority—highest first, lowest last. Table 3–14 summarizes the interrupts.

Table 3–14. Interrupt Summary

Interrupt	External/ Internal	Source	Priority	Vector Address MSB	Vector Address LSB
$\overline{\text{RESET}}$	E	$\overline{\text{RESET}}$ pin low	Immediate (highest priority)	>FFFE	>FFFF
$\overline{\text{INT1}}$	E	$\overline{\text{INT1}}$ pin active†	Priority 1	>FFFC	>FFFD
INT2	E/I	Timer/Event counter 1 ‡ count-down past 0	Priority 2	>FFFA	>FFFB
$\overline{\text{INT3}}$	E	$\overline{\text{INT3}}$ pin active †	Priority 3	>FFF8	>FFF9
INT4	I	RX Buffer Loaded, or TX Buffer Empty, or Timer 3 countdown past 0	Priority 4	>FFF6	>FFF7
INT5	E/I	Timer/Event counter 2 countdown thru 0	Priority 5	>FFF4	>FFF5

† The external interrupts on the TMS7xCx2 and TMS70Cx8 devices can be programmed for level and sense detection.

‡ The TMS70CTx0 devices do not contain the external event counter pin.

Note: INT4 and INT5 apply to TMS70Cx2, TMS77C82, and TMS70Cx8 devices only.

3.6.1 Device Initialization

$\overline{\text{RESET}}$, interrupt level 0, cannot be masked. The processor recognizes a reset immediately, even in the middle of an instruction execution. To execute the reset function, the $\overline{\text{RESET}}$ pin must be held low for a minimum of 1.25 × $t_{c(C)}$ internal state clock periods. While the $\overline{\text{RESET}}$ pin is asserted (0):

1) The data-direction registers for the I/O ports are cleared.

2) Port A's output data flip-flop is set to all 1s; ports C and D output data flip-flops are not altered during a reset.

3) This places ports C and D (and port A on TMS7xCx2 and TMS70Cx8 devices) in high-impedance input mode, and port B outputs all 1s (>FF), regardless of the internal machine clock state.

4) On TMS70Cx8 devices, ports E and F are placed in high impedance input mode. When port G is masked with chip select option, at device initialization, G2–G5 pins are all 1s and G0–G1 are high impedance inputs (1111ZZ). When port G is I/O masked option, port G is placed in high impedance input mode.

The reset function does not change the INTn flag bits in the IOCNT0 register (since all zeros are written). If any of the bits in a peripheral file data-direction register (DDR) are set to a 1, the corresponding port pin would become an output, producing a 1 level. (Remember, data-direction registers are set to all 0s on $\overline{\text{RESET}}$.)

It is generally a good practice to initialize the output data flip-flop with the desired output value (by writing to the port data value register) before writing to the DDR flip-flop to make the corresponding pin an output. Figure 3–20 and Figure 3–21 show examples of possible initialization routines after the assertion of $\overline{\text{RESET}}$. Device initialization requires 17 state cycles after $\overline{\text{RESET}}$ goes inactive.

When $\overline{\text{RESET}}$ returns to its inactive condition (1), the following operations are performed before the first instruction acquisition:

1) All 0s are written to the status register. This clears the global interrupt enable bit (I), disabling all interrupts.

2) All 0s are written to the IOCNT0 register. This disables $\overline{\text{INT}}$1, INT2, and $\overline{\text{INT}}$3 and leaves the INTn flag bits unchanged.

3) All 0s are written to the IOCNT1 register in the TMS70Cx2, TMS70Cx8, and TMS77C82 devices. This disables INT4 and INT5.

4) The PC's MSB and LSB values before $\overline{\text{RESET}}$ was asserted are stored in R0 and R1 (registers A and B), respectively.

5) The stack pointer is initialized to >01.

6) The MSB and LSB of the RESET interrupt vector are fetched from locations >FFFE and >FFFF, respectively (see Table 3–14 , page 3-39), and loaded into the program counter.

7) Program execution begins from the address placed in the program counter.

Figure 3–20. Sample Initialization Routine for TMS70Cx0 Devices

```
RESET   MOVP    %>2E,P0       CLEAR INT,INT2,INT3 FLAGS
*                             PLACE DEVICE IN SINGLE CHIP MODE
*                             ENABLE INT2 ( TIMER )
        MOVP    %VALUE1,P4    LOAD APORT DATA REGISTER
        MOVP    %MASK1,P5     LOAD APORT DATA DIRECTION REGISTER
        MOVP    %VALUE2,P6    LOAD BPORT DATA REGISTER
        MOVP    %VALUE3,P8    LOAD CPORT DATA REGISTER
        MOVP    %MASK2,P9     LOAD CPORT DATA DIRECTION REGISTER
        MOVP    %VALUE4,P10   LOAD DPORT DATA REGISTER
        MOVP    %MASK3,P11    LOAD DPORT DATA DIRECTION REGISTER
*
        MOVP    %>0F,P2       LOAD TIMER RELOAD REGISTER
        MOVP    %>9F,P3       SET CLOCK SOURCE TO INTERNAL
*                             PRE-SCALER WITH >IF
*                             SELECT WAKE-UP MODE
*                             START TIMER
        EINT                  SET GLOBAL INTERRUPT ENABLE BIT
```

Figure 3–21. Sample Initialization Routine for TMS70Cx2 and TMS77C82 Devices

```
RESET   MOVP    %>2E,P0       Clear INT1-, INT2, and INT3- flags,
*                             place device in Single-Chip mode, and
*                             enable INT2
        MOVP    %>22,P1       Select falling edge only for INT1 & INT3
        MOVP    %>0F,P2       Clear and enable INT4 and INT5
        MOVP    VALU1,P4      Load Port A Data Register
        MOVP    MASK1,P5      Load Port A Data-Direction Register
        MOVP    VALU2,P8      Load Port C Data Register
        MOVP    MASK2,P9      Load Port C Data-Direction Register
        MOVP    VALU3,P10     Load Port D Data Register
        MOVP    MASK3,P11     Load Port D Data-Direction Register
        MOVP    VALU4,P12     Load Timer 1 MSB reload register
        MOVP    VALU5,P13     Load Timer 1 LSB reload register
        MOVP    %>40,P14      Enable the timer output on B1
        MOVP    MASK4,P15     Initialize clock start, source, halt
*                             bit and prescaler value
        MOVP    VALU6,P16     Load Timer 2 MSB reload register
        MOVP    VALU7,P17     Load Timer 2 LSB reload register
        MOVP    %>40,P18      Enable the timer output on B0
        MOVP    MASK5,P19     Initialize clock start, source, halt
*                             bit and prescaler value
        MOVP    MASK6,P20     Initialize serial port format
        MOVP    MASK7,P21     Configure serial port
        MOVP    MASK8,P23     Load Timer 3 reload register
        MOVP    MASK9,P24     Configure serial port control
        EINT                  Set global interrupt enable bit
*                             to allow interrupts
```

the stack pointer can also be re-initialized following reset by executing a program similar to the one below.

```
STACK   MOV   %VALUE,B      Load Register B with the stack
*                           starting point in the Register
*                           File
        LDSP                Put this value into the Stack
*                           Pointer register
```

A simple R-C circuit can provide a power-up reset, automatically resetting the TMS7000 when power is applied. The capacitor and resistor values are selected according to the clock frequency used, the minimum voltage at which the $\overline{\text{RESET}}$ signal is at logic 1, and the ramp-up time of the power to the device. The following formula calculates the minimum time required for an adequate device reset:

$$t_{rst} = 2 \frac{V_{CC}}{V_{IL}} (1.25 t_{c(C)}) + t_{pwr} = RC$$

where:

t_{rst} = Total time $\overline{\text{RESET}}$ pin is held at logical level 0
V_{CC} = Supply voltage
V_{IL} = Low-level input voltage
$t_{c(C)}$ = Internal machine clock period
t_{pwr} = Ramp-up time for V_{CC}
R = Resistor value in ohms (no more than 1 megohm)
C = Capacitor value in farads

3.6.2 Interrupt Operation

The TMS7000 family's interrupts can be falling-edge sensitive, falling-edge and level sensitive, rising-edge sensitive, or rising-edge and level sensitive. Table 3–15 illustrates the interrupt configurations supported by each TMS7000 family device.

Table 3–15. External Interrupt Operation

TMS7000 Device and Interrupts		Falling Edge	Falling Edge and Level	Rising Edge	Rising Edge and Level
TMS70Cx0, TMS70CTx0	INT1 INT3	X	 X		
SE70CP160	INT1 INT3	X	 X		
TMS70Cx2, TMS70CX8†	INT1 INT3	X X	X X	X X	X X
TMS77C82	INT1 INT3†	X X	X X	X X	X X

† The TMS70Cx2, TMS77C82, and TMS77C82 devices' external interrupts edge/level sensitive polarity are software programmable. This is set via the I/O control 1 register (P1).

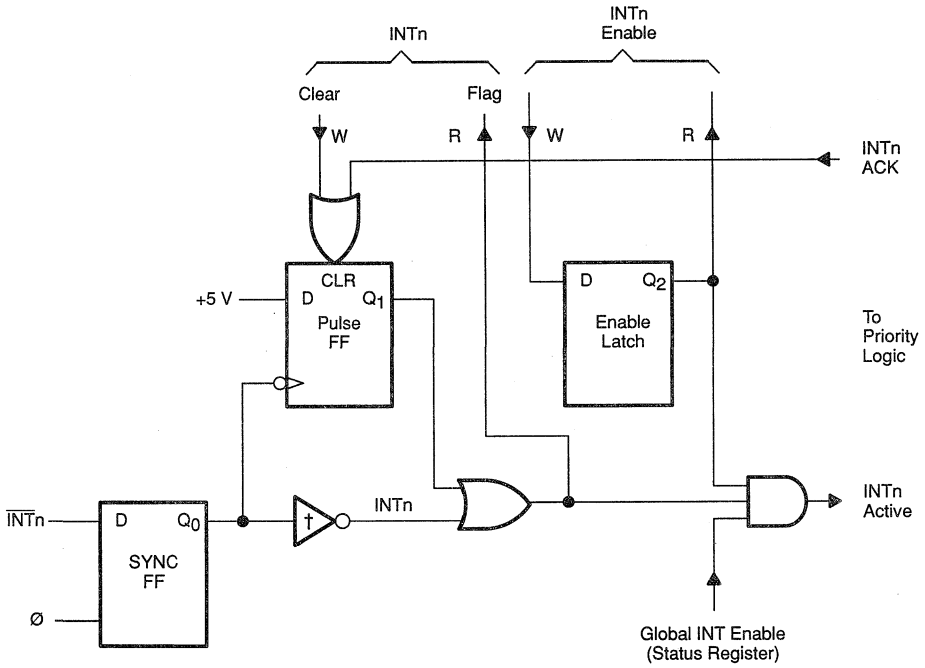
1) When an interrupt is first asserted, its level is gated into the sync flip-flop by the internal state clock, $t_{c(C)}$, which has a cycle period of $2/F_{osc}$. To detect an interrupt, the INTn signal must be active for a minimum of $1.25 \times t_{c(C)}$ clock periods.

2) The negative output edge of the sync flip-flop clocks a 1 into the pulse flip-flop. This is the "edge" detection of the interrupt signal and is the only time a 1 is loaded into the pulse flip-flop. The pulse flip-flop will be set within 1.25 state clock cycles of the interrupt assertion. If the signal is removed before the CPU recognizes the interrupt, its occurrence is latched on the pulse flip-flop output, Q1.

3) Edge-sensitive interrupts detect only the pulse flip-flop Q1 output, not the INTn level. Once an interrupt has been asserted (INTn goes low), it becomes active if the INTn enable bit and the global interrupt enable bit (I) register are set to one.

   The "level path" logic shown in Figure 3–22 applies only to external interrupts that are both edge- and level-sensitive; it is not implemented for interrupts that are only edge-sensitive. For more information, refer to Table 3–15.

Figure 3–22. CPU Interface to Interrupt Logic



† Available only for level-sensitive interrupts.

4) As shown in Figure 3–22, when the TMS7000's on-chip logic detects an active interrupt, it sends an INTn ACTIVE signal to the CPU. When the currently executing instruction is completed, the CPU acknowledges the active interrupt and routes INTA back to that interrupt's INTn ACK (interrupt acknowledge) line. If simultaneous interrupts occur, that is, more than one interrupt is active within the same instruction boundary, the interrupts are acknowledged by the CPU according to the priority levels. For example, if both INT2 and $\overline{\text{INT3}}$ occur within the same instruction boundary, INT2 is serviced first.

5) After the CPU acknowledges the interrupt, the INTn ACK line, as shown in Figure 3–22, clears the corresponding pulse flip-flop. The CPU then pushes the status register contents and the program counter onto the stack, and clears the status register, including the global interrupt enable (I) bit. The CPU reads an interrupt code from the interrupt priority logic to determine which interrupt requires servicing. The 16-bit vector value is read from the two vector addresses associated with the interrupt being serviced, and is loaded into the program counter. The interrupt vector val-

ue is the address of the first instruction in the interrupt service routine. The interrupt vector addresses are shown in Table 3–14 on page 3-39. Instruction execution then proceeds at the new address value in the program counter.

Nineteen internal state clock cycles [$t_{c(C)}$] are required between the end of an instruction in the interrupted program and the start of the first instruction of the interrupt service routine. Interrupting out of the Idle state requires 17 state clock cycles.

3.6.3 Interrupt Control

The I/O control registers, IOCNT0, IOCNT1, and IOCNT2 contain the interrupt control bits. All TMS7000 members have an IOCNT0 register. Only TMS70Cx2, TMS77C82 and TMS70Cx8 devices have an IOCNT1 register, because they have two more interrupts, INT4 and INT5; and an IOCNT2 register because only they can change the polarity of their external interrupts. The I/O control registers are mapped in to PF locations as follows:

Table 3–16. I/O Control Registers

Peripheral File	TMS70Cx0 TMS70CTx0	TMS70Cx2 TMS70Cx8 TMS77C82
IOCNT0	P0	P0
IOCNT1	NA	P2
IOCNT2	NA	P1

Figure 3–23. IOCNT0 — I/O Control Register 0 (P0 for All Devices)

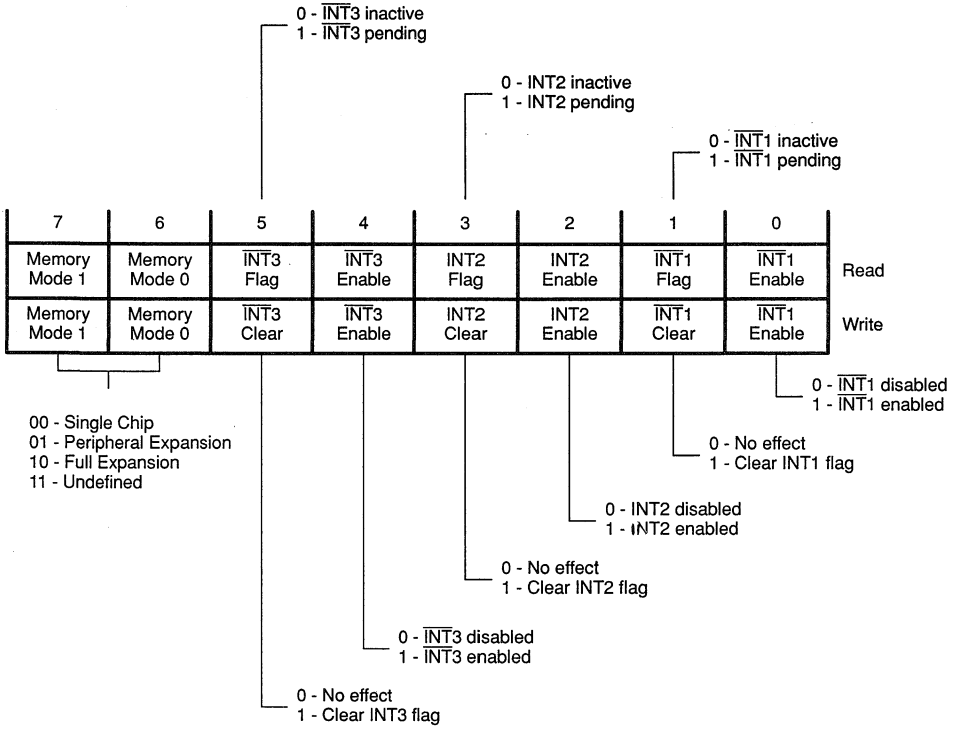


Figure 3–24. IOCNT1 — I/O Control Register 1

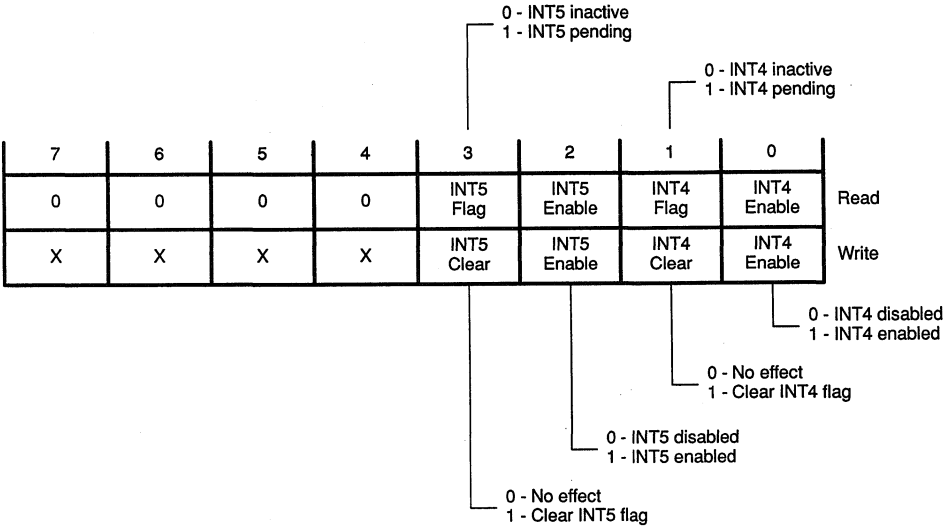
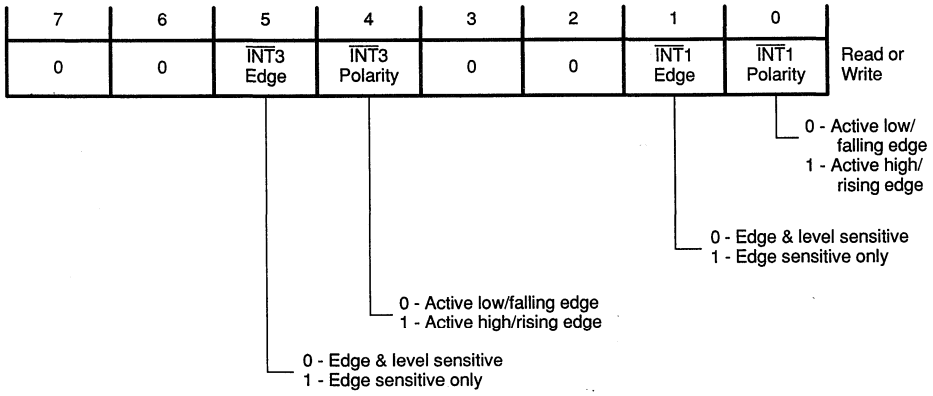


Figure 3–25. IOCNT2 — I/O Control Register 2 (P1 for TMS70Cx2 and TMS70Cx8 Only)



In the I/O control registers, each interrupt is associated with a flag bit (INTn flag) and enable bit (INTn enable). The global interrupt enable (I) bit in the status register allows all interrupts to be enabled or disabled at the same time. Three conditions must be met before the CPU will recognize an interrupt:

1) A 1 must be written to the INTn enable bit in the IOCNT0 or IOCNT1 register.

2) The global interrupt enable (I) bit in the status register must be set to 1 by the EINT instruction.

3) The interrupt must be the highest priority interrupt asserted within an instruction boundary.

Through software, the INTn enable bits can be read and written to:

- Writing a 0 individually masks the corresponding interrupt.
- Writing a 1 allows the interrupt to be recognized.

The reading of the INTn flag is handled differently (see Figure 3–22 on page 3-44):

- An active signal applied to INTn is read as a 1 from one side of an OR gate.
- INTn going active latches a 1 to the other side of the OR gate which stays latched when the signal goes inactive.

Thus INTn going active is returned as both a latched and edge sensitive signal on INT3 of the TMS70Cx0 devices, or any external interrupts selected for edge and level sensitivity on TMS7xCx2 and TMS70Cx8 devices.

The INTn flag bit may be tested in software, regardless of whether the interrupt is enabled or disabled. For example, the following program statement waits for the active edge of the interrupt input on the $\overline{\text{INT1}}$ pin by testing INT1 flag:

```
WAIT    BTJZP   %>02,P0,WAIT    Wait for INT1-
*                                (INT1 flag = 1)
```

This allows external interrupt pins to be polled as inputs. Interrupt input pins have an advantage over the other general-purpose inputs if the input signal is a short pulse. The pulse flip-flop of the interrupt input will always capture a pulse with a width of at least 1.25 × $t_{c(C)}$ cycles, allowing software to detect that the condition occurred, even after the pulse is gone.

If the level is required to be tested (for example, for a debounce routine) then either $\overline{\text{INT3}}$ on TMS70Cx0 devices or edge and level sensitivity on TMS7xCx2 and TMS70Cx8 should be used. The flag should be tested *after* an attempt is made to clear it. If the active level is still present on the input, the flag will be reasserted by the level. This is necessary because if the active level is removed, the pulse flip-flop will still retain the original active edge information.

```
WAIT    MOVP    %>22,P0          ATTEMPT TO CLEAR INT1 AND INT3
        BTJOP   %>02,P0,WAIT     JUMP TO WAIT UNTIL INPUT REMOVED
```

Due to the read/modify/write nature of the bit manipulation instructions (ANDP, ORP, and XORP), it is possible that the INTn flag bits in the IOCNT0 and IOCNT1 registers could be unintentionally cleared. To avoid these occurrences, *use the MOVP and the STA instructions when writing data to IOCNT0 and IOCNT1.*

Because the INTn flag and INTn clear bits are in the same bit positions, use caution when accessing these bits. For example, you may be able to use XORP to set INT1 enable without altering the state of the INT1 flag (XORP %>03,PO), as long as the INT1 flag does not change state during the instruction execution. However, if a short INT1 pulse sets the pulse flip-flop between the read and write portions of the instruction execution, a 0 would be read from INT1 flag and a 1 would be written to INT1 clear to reclear INT1 flag. In this case, the $\overline{\text{INT1}}$ pulse would be undetected by the processor. This same instruction would also affect the INT2 flag and $\overline{\text{INT3}}$ flag in a similar manner as they are also located in the IOCNT0 register.

Immediately following $\overline{\text{RESET}}$, all interrupts are globally disabled because the I bit (interrupt enable) in the status register is reset to 0. Also, the IOCNT0 register is cleared. This clears the INTn enable bits, disabling $\overline{\text{INT1}}$, INT2, and $\overline{\text{INT3}}$ individually and putting the TMS7000 in single-chip mode. This does not affect the INTn flag bits from their previous condition before $\overline{\text{RESET}}$.

3.6.4 Multiple Interrupt Servicing

When an interrupt is recognized, the global interrupt enable (I) status register bit is automatically cleared while the interrupt is serviced. This prevents all other interrupts from being recognized during the execution of the interrupt service routine. Once the service routine is completed by executing the RETI (Return from Interrupt) instruction, the old status register contents are popped from the stack. This returns the I bit back to 1, allowing any pending interrupts to be recognized.

An interrupt service routine can explicitly allow nested interrupts by executing the EINT instruction to directly set the I bit in the status register to a 1, thus permitting other interrupts to be recognized during service routine execution. When a nested interrupt service routine completes, it returns to the previous interrupt service routine when the RETI instruction is executed.

Interrupt priorities are only evaluated when interrupts are enabled and more than one source is pending. Thus, for example, if an INT1 is received and the subroutine is entered, and an EINT is encountered allowing nested interrupts, then INT2 and INT3 occur simultaneously, the following action would occur:

1) Following the EINT instruction, INT2 and $\overline{\text{INT3}}$ would be prioritized.
2) INT2 would be immediately serviced (even though it is lower priority than the currently serviced $\overline{\text{INT1}}$) and $\overline{\text{INT3}}$ would be left pending. Interrupts would be globally disabled.
3) On returning from INT2 the $\overline{\text{INT1}}$ status register would be restored, which would re-enable global interrupts.
4) $\overline{\text{INT3}}$ would then be serviced (even though it is lower priority than the original $\overline{\text{INT1}}$) and interrupts would be globally disabled.
5) On returning from $\overline{\text{INT3}}$, the original $\overline{\text{INT1}}$ status would be restored which would re-enable global interrupts.
6) As no other interrupts are pending, the $\overline{\text{INT1}}$ service routine would be resumed and completed before returning to the background routine.

3.6.5 External Interrupt Servicing

The external interrupt interface consists of three discrete input lines that require no external synchronization: $\overline{\text{RESET}}$, $\overline{\text{INT1}}$, and $\overline{\text{INT3}}$.

TMS70Cx0, TMS70CTx0, SE70CP160

The external interrupt $\overline{\text{INT1}}$ on the TMS70Cx0, TMS70CTx0 and SE70CP160 devices is a high-impedance falling-edge sensitive only interrupt, while $\overline{\text{INT3}}$ is a high-impedance falling-edge and level-sensitive interrupt.

TMS70Cx2, TMS77C82, TMS70Cx8, SE70CP168

The external interrupts on these devices can be individually programmed as falling-edge sensitive only, falling-edge and level sensitive, rising-edge sensitive only, or rising-edge and level sensitive.

Certain safeguards should be observed for external interrupts that are both edge- and level-sensitive. The logical-OR of both the pulse flip-flop output (Q1) and $\overline{\text{INTn}}$ (inverted INTn) affect the state of INTn flag, and can therefore activate the interrupt (see Figure 3–22 on page 3-44). The pulse flip-flop is automatically cleared when the CPU acknowledges the interrupt. However, as long as the INTn pin is low, the interrupt will remain active even when the pulse flip-flop output is 0. This is how an external interrupt source is detected as a level signal. If INTn is active longer than the shortest path through the interrupt service routine, this same interrupt will be serviced again upon return from the service routine if no higher priority interrupts are active. In many applications this interrupt re-servicing is acceptable; however, in applications where this is a potential problem, the associated INTn enable bit must be disabled before exiting the interrupt service routine. Upon return from the service routine, INTn

flag must be periodically software-polled to determine when INTn has gone inactive, and then INTn enable may be re-enabled. Note that devices with edge-sensitive only interrupts do not require the previously mentioned safeguards.

To prevent an interrupt signal from being detected as a level signal, the maximum pulse (time low) of the signal cannot exceed the following:

$$(16 + N) \times t_{c(C)}$$

where:

N = the total number of state clock cycles in the interrupt service routine, up to and including the EINT or RETI instruction

$t_{c(C)}$ = the internal state clock cycle period

This ensures that the INTn flag is cleared before the first possible instruction boundary in which the interrupt could be re-serviced. Note that this is not of any concern for INT1 on the TMS70Cx0 and TMS70CTx0 devices since it is edge-sensitive only, not level sensitive.

3.6.6 External Interrupt Signals

Some applications may cause an incorrect interrupt vector to be accessed when using edge- and level-sensitive interrupts on the *TMS70Cx0 and TMS70CTx0 devices only*. This may happen when an INTn pulse goes inactive on the boundary condition when interrupts are being enabled. Two events are necessary for this to occur:

1) First, the pulse flip-flop is cleared upon entry to the interrupt service routine; since the INTn pin is still active, INTn flag and INT active remain active.

2) Second, the INTn pin goes inactive on the boundary condition when interrupts are being enabled (RETI and EINT instructions or a write to IOCNT0 to enable interrupts).

When the INTn pin goes inactive, INTn flag becomes inactive and some time later INT active becomes inactive. This results in INT active being acknowledged by the CPU, but INTn flag becomes inactive before interrupt decode logic can determine which interrupt was pending. Note that INTn has already been serviced, so that reservicing of the interrupt is not required. If this condition occurs, interrupt vector fetches from locations >FFF8 and >FFF9 (for $\overline{\text{INT3}}$) will occur for TMS70Cx0 and TMS70CTx0 devices. This situation does not exist for edge-sensitive only interrupts (such as $\overline{\text{INT1}}$ on the TMS70CTx0 and TMS70Cx0 devices, and the interrupts on the TMS70Cx2, TMS70Cx8, and TMS77C82 devices).

In applications where the system design cannot guarantee that the duration of the pulsed interrupt is outside this critical window, three system solutions should be considered.

- A system hardware solution uses an external D-type flip-flop or a one-shot in the interrupt path, providing a level interrupt which the TMS7000 would externally clear as part of the service routine.
- Prevent the re-servicing of the interrupt as described earlier by setting the associated INTn enable bit to 0 in the interrupt service routine.
- If only one external interrupt has the potential to cause this boundary condition, for TMS70Cx2/77C82/70Cx8 devices with edge and level sensitivity enabled, a trap vector should be placed in location >FFF0 and >FFF1 which points to a RETI instruction. This will return the program to normal program flow if this condition occurs. For TMS70Cx0 devices, use INT1 since this interrupt is only edge sensitive and will not exhibit the condition.

3.7 Programmable Timer/Event Counters

The programmable timer/event counters are 8-bit or 16-bit counters with programmable, prescaled clock source. TMS70Cx0 and TMS70CTx0 devices contain one timer/event counter. TMS70Cx2, TMS77C82, and TMS70Cx8 devices contain two timer/event counters and one timer. The data and control registers for these two timer/event counters are shown in Figure 3–26 through Figure 3–31.

All TMS7000 timer/event counters are down counters, and start decrementing from the value stored in the timer reload registers at a rate determined by the system clock and the value stored in the prescaler reload registers. On reaching zero, an interrupt can be generated to the CPU. The prescaler and timer reload registers are automatically downloaded to prescaler and decrementer registers as the timer passes through zero, allowing a new time period to be loaded by the software without disturbing an ongoing count.

- **Timer 1** is available on all **TMS7000** devices.
  - **TMS70Cx0** and **TMS70CTx0**

    Timer 1 is an 8-bit timer/event counter with a 5-bit programmable prescaler. It contains an 8-bit capture latch and is accessed through PF registers P2 and P3. Note that the TMS70CTx0 devices do not contain the A7/EC1 pin necessary for external event counting.
  - **TMS70Cx2**, **TMS77C82**, and **TMS70Cx8**

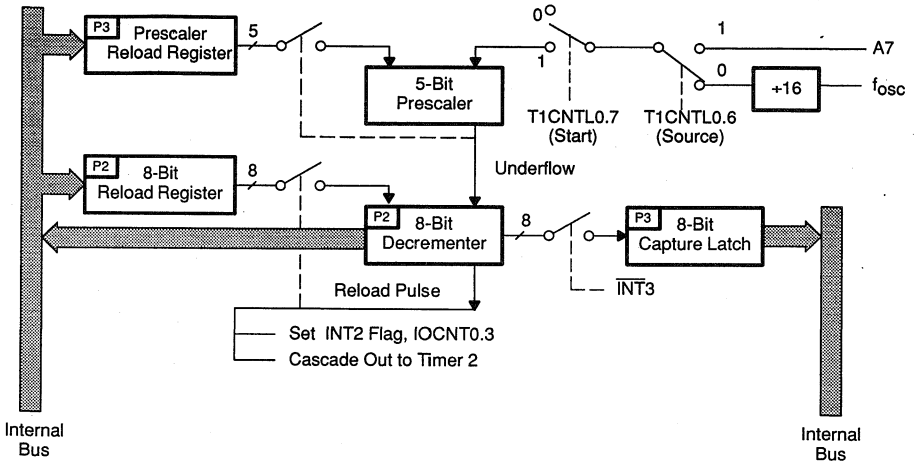
    Timer 1 is a 16-bit timer/event counter that contains a 5-bit programmable prescaler and a 16-bit capture latch. It is accessed through PF registers P12, P13, P14, and P15.
- **Timer 2** is available on the **TMS70Cx2/TMS77C82/TMS70Cx8** devices.

  Timer 2 is a 16-bit timer/event counter that contains a 5-bit programmable prescaler and 16-bit capture latch. It is accessed through PF registers P16, P17, P18, and P19.
- **Timer 3** is available on the **TMS70Cx2/TMS77C82/TMS70Cx8** devices and can be used as an independent timer or as the clock source for the on-chip serial port. Because of this function, Timer 3 is described in more detail in Section 3.8, The Serial Port.

Note:

The contents of all registers associated with the timers are not affected by a hardware $\overline{\text{RESET}}$. These registers must be initialized by software.

Figure 3–26. 8-Bit Programmable Timer/Event Counters — Timer 1 (TMS70Cx0 and TMS70CTx0)



Note: Pin A7/EC1 not available on the TMS70CTx0 devices.

Figure 3–27. 16-Bit Programmable Timer/Event Counters — Timer 1 (TMS70Cx2, TMS77C82, and TMS70Cx8)

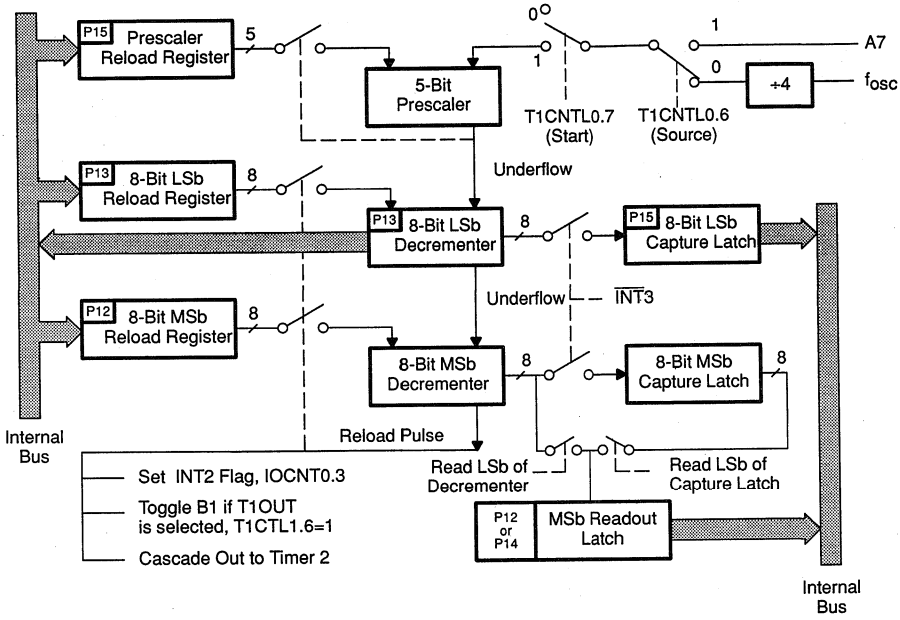
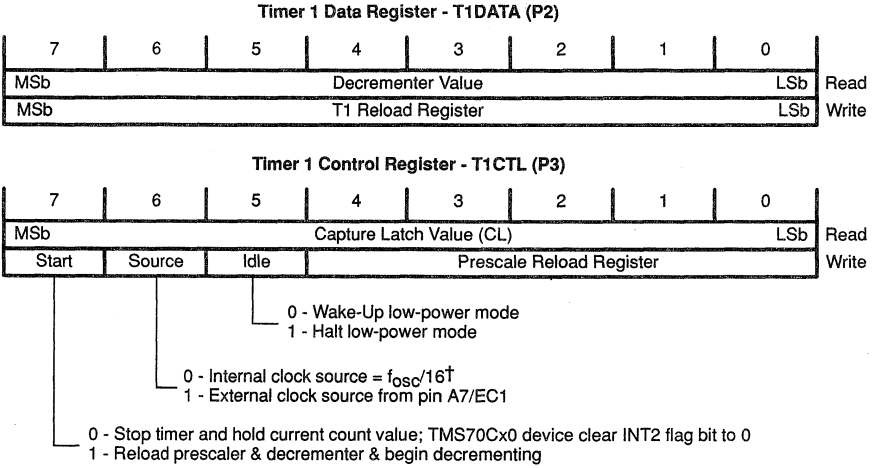


Figure 3–28. Timer 1 Data and Control Registers (TMS70Cx0 and TMS70CTx0)



† Only internal clock source available on the TMS70CTx0 devices.

Figure 3–29. Timer 1 Data and Control Registers (TMS70Cx2, TMS77C82, and TMS70Cx8)

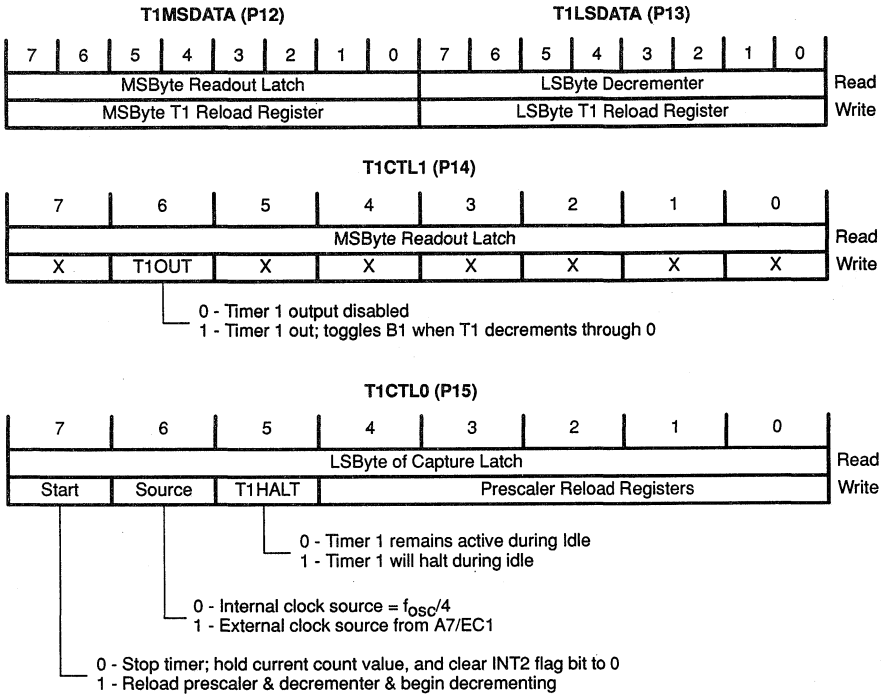


Figure 3–30. 16-Bit Programmable Timer/Event Counters — Timer 2 (TMS70Cx2, TMS77C82, and TMS70Cx8)

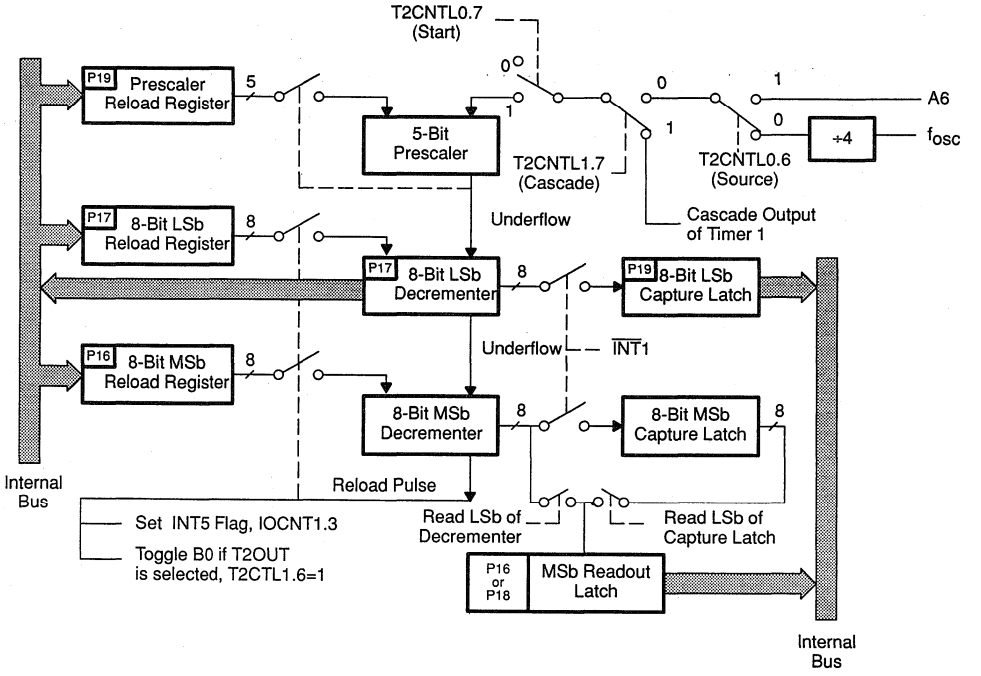
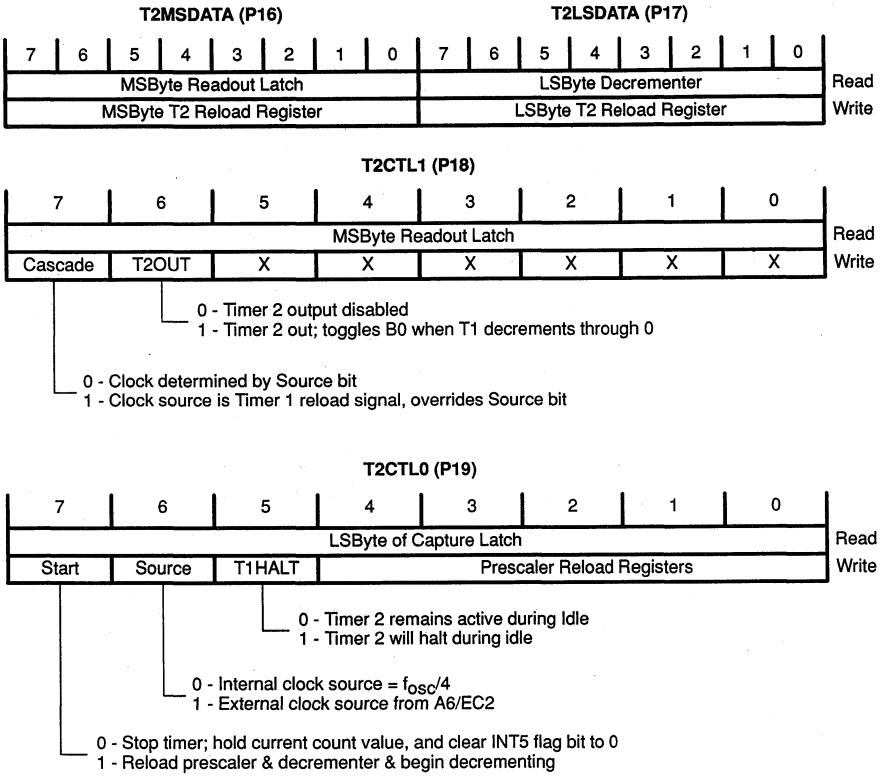


Figure 3–31. Timer 2 Data and Control Registers (TMS70Cx2, TMS77C82, and TMS70Cx8)



3.7.1 Control Registers for Timer/Event Counters 1 and 2 (TMS70Cx0 and TMS70CTx0 Devices)

The control bits and prescaling value of Timer 1 are determined by the control register bits in T1CTL. These bits can only be written to the control registers and cannot be read by a program. When T1CTL is read, the 8-bit capture latch value is returned. Since the control and prescale bits are write only and the capture latch is read only, both existing in T1CTL, the read/modify/write instructions such as ANDP, ORP, and XORP should not be used. The following instructions should be used for timer control-bit manipulations.

```
MOVP   %>XX,Pn     STA    %>01xx
MOVP   A,Pn        STA    *Rn
MOVP   B,Pn        STA    >01xx(B)
```

where:

%>XX = Immediate 8-bit hexadecimal data value
>01xx = 16-bit Peripheral-File hexadecimal address
A = Register A
B = Register B
Pn = Peripheral-File register number
Rn = General-purpose register pair number

When writing to the timer data register T1DATA, the value is actually being stored in the timer reload register. If the timer is still running, it will continue to decrement from its original value. This allows the timers to be preloaded with the next period to be timed without affecting the current time-out period. This can be used in PWM generation. When reading the timer data register, the value directly from the decrementer is returned, and can be read if the timer is running or halted. As the reload register and decrementer values are likely to be different, the read/modify/write instructions described above should also not be used for the timer data registers.

3.7.2 Control Registers for Timer/Event Counters 1 and 2 (TMS70Cx2, TMS77C82, and TMS70Cx8 Devices)

The control bits and prescaling value of Timers 1 and 2 of the TMS70Cx2, TMS77C82, and TMS70Cx8 devices are determined by the timer control registers T1CTL0 (P15), T1CTL1 (P14), T2CTL0 (P19), and T2CTL1 (P18). Data can only be written to these control registers, and cannot be read back by a program. When Timer 1 control register T1CTL0 is read, the least significant (LS) byte of the capture latch value associated with Timer 1 is returned. When T1CTL1 is read, the most significant (MS) byte of the Timer 1 readout latch is returned. When T2CTL0 is read, the least significant (LS) byte of the Timer 2 capture latch is returned. When T2CTL1 is read, the most significant (MS) byte of the Timer 2 readout latch is returned. Since the control and prescale bits are write only, the read/modify/write instructions such as ANDP, ORP, and XORP

should not be used. The following instructions should be used for timer control-bit manipulations.

```
MOVP    %>XX,Pn     STA  %>01xx
MOVP    A,Pn        STA  *Rn
MOVP    B,Pn        STA  >01xx(B)
```

where:

%XX = Immediate 8-bit hexadecimal data value
>01xx = 16-bit Peripheral-File hexadecimal address
A = Register A
B = Register B
Pn = Peripheral-File register number
Rn = General-purpose register pair number

When writing to the timer data registers T1LSDATA,T1MSDATA,T2LSDATA and T2MSDATA the values are actually being stored in the timer reload registers. If the timer is still running, it will continue to decrement from its original value. This allows the timers to be preloaded with the next period to be timed without affecting the current time-out period. This can be used in PWM generation. When reading the timer data registers, the value directly from the decrementer is returned, and can be read if the timer is running or halted. As the reload register and decrementer values are likely to be different, the read/modify/write instructions described above should also not be used for the timer data registers.

3.7.3 Timer Start/Stop (Bit 7) and Capture Latch

Bit 7 of the timer control registers contain a start/stop bit for the timer/event counters.

Bit 7 = 0 A start bit of **0** disables or freezes the timer chain at the current count value.

Bit 7 = 1 A start bit of **1**, regardless of whether the bit was a 0 or a 1 before, loads the prescaler and counter decrementers with the corresponding reload register values, and the timer/event counter operation begins.

3.7.3.1 Timer 1 Capture Latch (TMS70Cx0 and TMS70CTx0 Devices)

The Timer 1 8-bit capture latch can be accessed by reading the Timer 1 control register T1CTL (P3). T1CTL will contain the "captured" current Timer 1 value whenever $\overline{\text{INT3}}$ is triggered even if $\overline{\text{INT3}}$ is disabled. Please note that when $\overline{\text{INT3}}$ is used to exit a low-power mode on the TMS70Cx0 or TMS70CTx0 CMOS parts, the capture latch may store an indeterminate value. This is due to the logic design of the CMOS devices. Since the value in the capture latch may not be valid when leaving either of the low-power modes via $\overline{\text{INT3}}$, it is recommended that the capture latch not be used in this situation.

3.7.3.2 Timer 1 and Timer 2 Capture Latches (TMS70Cx2, TMS77C82, and TMS70Cx8 Devices)

The TMS70Cx2, TMS77C82, and TMS70Cx8 devices contain two 16-bit capture latches, one each for Timer 1 and Timer 2. The Timer 1 16-bit capture latch can be accessed by reading the Timer 1 control registers T1CTL0 (P15) and T1CTL1 (P14). The Timer 2 16-bit capture latch can be accessed by reading the Timer 2 control registers T2CTL0 (P19) and T2CTL1 (P18). The capture latch values for Timer 1 and Timer 2 are loaded on the active edges of $\overline{\text{INT3}}$ and $\overline{\text{INT1}}$, respectively, whether the interrupts are enabled or not. Both capture latches are disabled during the IDLE instruction when their corresponding HALT bits are 1.

Reading the Timer 1 control register T1CTL1 or the Timer 2 control register T2CTL will return the value of the MSB readout latch of the respective timer. This latch is shared between MSB of the timer latch and the MSB of the capture latch. It allows the complete 16-bit value of the timer latch or the capture latch to be sampled at one moment. The LSB must be read first, which causes the MSB to be simultaneously loaded into the readout latch. This latch physically exists in only one location for each timer; however, each latch can be read from two different locations. Timer 1 MSB readout latch can be read from T1MSDATA (P12) or T1CTL1 (P14). Timer 2 MSB readout latch can be read from T2MSDATA (P16) or T2CTL1 (P18).

Reading the LSB of the decrementer or capture latch will update the contents of the readout latch. In order to correctly read the entire 16-bit value of the decrementer or capture latch, the LSB must be read first, which will load the MSB readout latch. The MSB readout latch must be read and stored before reading the LSB of either the decrementer or capture latch. The order of 16-bit read operations should be:

Timer 1: *Decrementer*: Read P13 then P12 ***or*** read P13 then P14
Capture Latch: Read P15 then P12 ***or*** read P15 then P14

Timer 2: *Decrementer*: Read P17 then P16 ***or*** read P17 then P18
Capture Latch: Read P19 then P16 ***or*** read P19 then P18

3.7.4 Clock Source Control (Bit 6) (See note below.)

For the **TMS70Cx0** and **TMS70CTx0** devices, bit 6 (source) of T1CTL (P3) selects the clock source for Timer 1. For the TMS70Cx2, TMS77C82, and TMS70Cx8 devices, bit 6 (source) of T1CTL0 and T2CTL0 selects the Timer 1 and Timer 2 clock sources, respectively.

For the **TMS70Cx2** devices, bit 6 (SOURCE) of T1CTL0 and T2CTL0 selects the Timer 1 and Timer 2 clock sources, respectively.

Bit 6 = 0 A source bit of **0** selects the internally generated clock and places the timer/event counter in the realtime clock mode using the internal clock source. Each positive transition of the timer clock signal decrements the count chain. Realtime clock mode allows a program to periodically interrupt and call a service routine, such as a display refresh, by simply setting the prescale reload register and the timer reload register so the routine is called at the desired frequency.

Bit 6 = 1 A source bit of **1** selects the external clock source and places the timer/event counter in the event-counter mode. In this mode, each positive transition at the port A event counter pins decrements the count chain (when the prescaler is decremented to zero, it is reloaded with the prescaler reload register value and the counter is decremented by one).

Summary for all TMS7000 devices (see note below):

	Event Counter Input Pin	Interrupt Level
Timer 1	Pin A7/EC1	INT2
Timer 2	Pin A6/EC2	INT5

The event-counter mode allows INT2 and INT5 to function as positive edge-triggered external interrupts by loading a start value of 0 into both the prescaler and timer reload register. A positive transition on A7/EC1 or A6/EC2 decrements the corresponding timer through zero and generates an INT2 or INT5. Event-counter mode can also be used as an externally provided realtime clock if an external clock is input on the I/O pin. The minimum clock period on pins A7/EC1 or A6/EC2 must not be less than $f_{osc}/16$ for TMS70Cx0 devices, or $f_{osc}/4$ for TMS70Cx2 devices. The minimum pulse width of the external signal must not be less than 1.25 state clock cycles [$1.25 \times t_{c(C)}$] to be properly detected by the device.

Note:

The TMS70CTx0 devices do not contain an external event counter input pin. Therefore, the clock source must be selected as internal.

3.7.5 Idle/Timer Halt Bit (Bit 5)

The function of the Idle bit (bit 5) in the timer control registers varies depending on the device type.

- **TMS70Cx0** and **TMS70CTx0**

  Bit 5 of T1CTL (P3) register is the Idle bit. This bit selects either of two low-power modes on these devices when the IDLE instruction is executed. (See subsection 3.4.2 about CMOS low-power modes.)

  Bit 5 = 0 Wake-up low-power mode
  Bit 5 = 1 Halt low-power mode

- **TMS70Cx2, TMS77C82, and TMS70Cx8**

  Bit 5 of the T1CTL0 (P15) and T2CTL0 (P19) registers acts as a timer-halt bit. This bit selects either of two timer operational modes when the IDLE instruction is executed.

  Bit 5 = 0 Timer active mode
  Bit 5 = 1 Halt timer mode

3.7.6 Cascading Timers

The TMS70Cx2, TMS77C82, and TMS70Cx8 devices can have their timers cascaded together to form one large timer. The external clock input for Timer 2 is the port A pin A6/EC2.

- **Cascade Bit**

  Bit 7 of the T2CTL1 (P18) register is the Cascade bit. This bit is used in conjunction with the T2CTL0 (P19) Source (bit 6) to determine the Timer 2 clock source.

  Bit 7 = 0 A Cascade bit of **0** allows bit 6 of T2CTL0 to determine the clock source.

  Bit 7 = 1 A Cascade bit of **1** selects the output generated by the Timer 1 reload pulse as the clock input to the prescaler of Timer 2. The Cascade bit overrides the Source bit; that is, if the Cascade bit is 1, the Source bit of Timer 2 has no effect.

Note that the Timer 2 output (T2OUT) cannot be used if Timer 1 and Timer 2 are cascaded together.

3.7.7 Timer and Prescaler Operation

The timer clock, whether internal or external, is prescaled by a 5-bit modulo-N counter. The prescaling value is determined by the least significant five bits of the timer control register. The timers decrement and an underflow occurs on

the transition from 0 to >FF. Thus, a prescale value of >7 will produce an $f_{osc}/128$ clock input into the timer for a TMS70Cx0 device with a timer clock source of $f_{osc}/16$.

- **TMS70Cx0**, and **TMS70CTx0**

  Timer 1 Bits 0–4 of Timer 1 control register T1CTL comprise the Timer 1 prescale reload register value.

- **TMS70Cx2**, **TMS77C82**, and **TMS70Cx8**

  Timer 1 Bits 0–4 of Timer 1 control register T1CTL0 comprise the Timer 1 prescale reload register value.

  Timer 2 Bits 0–4 of Timer 2 control register T2CTL0 comprise the Timer 2 prescale reload register value.

These steps occur during timer operation:

1) Upon starting the timer, the prescaler and timer are loaded from the prescaler reload register and timer reload register, respectively.

2) Each pulse decrements the prescaler by one.

3) When the prescaler countdown decrements through zero, the timer is decremented by one. After the prescaler is decremented,

   If timer ≠ 0 Reload prescaler and go back to step 2.

   If timer = 0 When both the timer and the prescaler decrement through zero together, an interrupt occurs. An INT2 for Timer 1 (INT5 for Timer 2) is momentarily pulsed when both the prescaler and counter decrement past the zero value together. This sets the INT2 or INT5 pulse flip-flop, as described in subsection 3.6.2, Interrupt Operation.

4) The 5-bit prescaler and decrementer are then immediately reloaded with the contents of the prescale reload register and the timer reload register, and the timer will start decrementing with the new reload register values.

- **TMS70Cx0 and TMS70CTx0**

  The 8-bit timer reload register is loaded through the Timer 1 data register T1DATA (P2). This value is write only. When read, T1DATA returns the current value of the 8-bit decrementer and **not** the current value of the timer reload register. When read, the T1CTL returns the capture latch value for Timer 1 and **not** the prescaler reload register.

- **TMS70Cx2, TMS77C82, and TMS70Cx8**

  The 16-bit timer reload registers are loaded through the Timer 1 data registers T1LSDATA (P13) and T1MSDATA (P12), and the Timer 2 data regis-

ters T2LSDATA (P17) and T2MSDATA (P16). This value is write only. When read, T1LSDATA and T2LSDATA return the current value of the LSB of the Timer 1 and Timer 2 decrementers, respectively, and not the LSB timer reload register value. For this reason, the read/modify/write I/O instructions should not be used to alter the data value in the timer reload registers. T1MSDATA and T2MSDATA will return the value of the MSB readout latch for Timers 1 and 2, respectively. To read the **Timer 1** capture latch, first read T1CTL0 (P15) to obtain the LSB, then read T1CTL1 (P14) to obtain the MSB. To read the **Timer 2** capture latch, first read T2CTL0 (P19) to obtain the LSB, then read T2CTL1 (P18) to obtain the MSB.

3.7.8 Timer Interrupts

When the prescaler and decrementers pass through zero together, an interrupt flag (INTn flag) is set and the prescaler and counter decrementers are immediately and automatically reloaded with the corresponding reload register values. The interrupt levels generated by the timers are INT2 for Timer 1 and INT5 for Timer 2. The period between successive timer interrupts may be calculated by the following formula:

- **TMS70Cx0** and **TMS70CTx0**

  $t_{INT} = t_{CLK} \times (PR+1) \times (TR+1)$

  where:

  t_{INT} = Period between timer interrupts
  t_{CLK} = Period of the timer input clock which is $16/f_{osc}$ for real time clock mode or the period of the external input pin for event-counter mode
  PR = 5-bit prescaler reload register value
  TR = 8-bit timer reload register value

  At the falling edge of the $\overline{INT3}$ input, the Timer 1 counter value is loaded into the capture latch. This feature provides the capability to determine when an external event occurred relative to the current Timer 1 decrementer value.

- **TMS70Cx2**, **TMS77C82**, and **TMS70Cx8**

  $t_{INT} = t_{CLK} \times (PR+1) \times (TR+1)$

  where:

  t_{INT} = Period between timer interrupts
  t_{CLK} = Period of the timer input clock which is $4/f_{osc}$ for realtime clock mode or the period of the external input pin for event-counter mode
  PR = 5-bit prescaler reload register value
  TR = 16-bit timer reload register (value written to the MSB and LSB timer reload registers)

The falling edge of the $\overline{INT3}$ input will cause the 16-bit decrementer value of Timer 1 to be loaded into the Timer 1 capture latch. Likewise, the falling edge of the $\overline{INT1}$ input will cause the 16-bit decrementer value of Timer 2 to be loaded into the Timer 2 capture latch. This feature provides the capability to determine when an external event occurred relative to the current timer/counter value.

3.7.9 Pulse Width Modulation, Timer Output Function (TMS70Cx2, TMS77C82, and TMS70Cx8)

A pulse width modulation (PWM) function exists on both Timer 1 and Timer 2, that allows the B1 and B0 outputs, respectively, to be toggled every time the timer decrements through zero. This function is controlled by the PWM bit, (or T1OUT and T2OUT bits), the bit 6 in the timer control registers T1CTL1 and T2CTL1.

When operating in the timer output mode, the B0 and/or B1 output cannot be changed by writing to the Port B Data Register. Writing to the appropriate timer's Start bit will reload and start the timer, and will not toggle the output. The output will toggle only when the timer decrements through zero. The timer output feature is independent of INT2 and INT5; therefore, it will operate with INT2 and INT5 enabled or disabled. Also, if the timer is active during the IDLE instruction, the timer output feature will continue to operate.

Whenever the T2OUT or T1OUT bit is returned to 0, B0 or B1 will become an output-only pin, like B2. The value in the B0 or B1 data register will be the last value output by the timer output function, to that B0 or B1 will not change as the T1OUT or T2OUT bit is returned to 0.

Whenever port B is read, the value on the B0 pin will always be returned, so the current timer output value can be read by reading port B.

The T1OUT and T2OUT bits are set to 0 by a reset, so the timer output function will not be enabled unless the user sets T1OUT or T2OUT to 1.

The Timer 2 output (T2OUT) cannot be used if Timer 1 and Timer 2 are cascaded together (Cascade bit of T2CTL1 set to 1).

3.8 Serial Port (TMS70Cx2, TMS77C82, and TMS70Cx8 Devices Only)

The TMS70Cx2, TMS77C82, and TMS70Cx8 devices contain a serial port, greatly enhancing their I/O and communications capabilities. Including a hardware serial port on chip saves ROM code and allows much higher transmission rates than could be achieved through software.

The full-duplex serial port consists of a receiver (RX), transmitter (TX), and a third timer called Timer 3 (T3). The functional operation of the serial port is configured through software initialization. A set of control words are first sent out to the serial port to initialize the desired communications format. These control words will determine the baud rate, character length, even/odd/off parity, number of stop bits, and so forth.

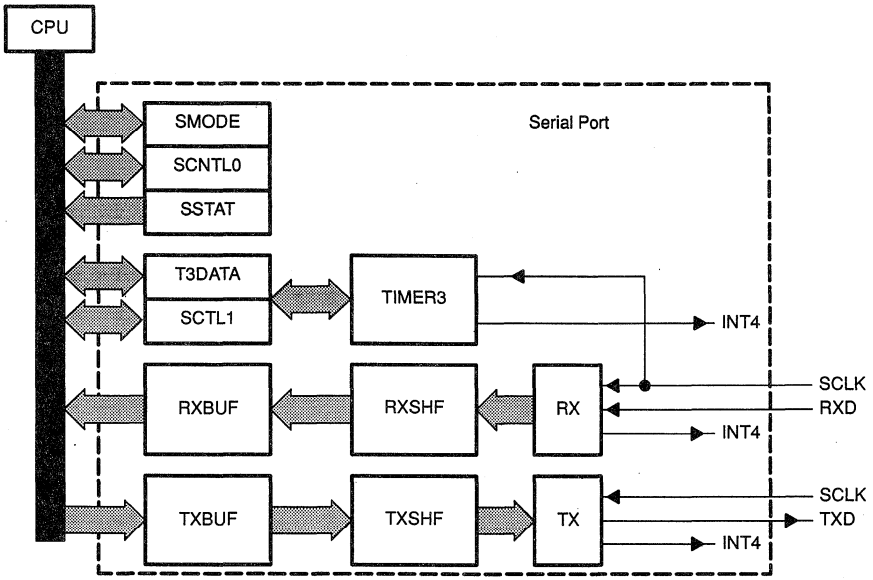
Figure 3–32 (page 3-69) illustrates the serial port functional blocks.

The serial port provides universal synchronous asynchronous receiver/transmitter (USART) communications:

- **Asynchronous mode**, discussed in subsection 3.8.2.1 (page 3-83) interfaces with many standard devices such as terminals and printers using RS-232-C formats.
- **Isosynchronous mode**, discussed in subsection 3.8.2.2 (page 3-84) permits very high transmission rates and requires a synchronizing clock signal between the receiver and transmitter.
- **Serial I/O mode**, discussed in subsection 3.8.2.3 (page 3-85) can be used to expand I/O lines and to communicate with peripheral devices requiring a non-UART serial input such as A-to-D converters, display drivers, and shift registers.

The serial port has also two multiprocessor protocols, compatible with the Motorola (MC6801, MC6811, HD6301, ...) and INTEL (i8051, i80C51, i8096, ...). These protocols allow efficient data transfer between multiple processors. They are implemented using isosynchronous or standard asynchronous formats.

Figure 3–32. Serial Port Functional Blocks



Note:

The INT4 sources are effectively wire-ORed together to generate only one INT4 input. The SCLK sources are wired together to generate only one SCLK input.

3.8.1 Serial Port Registers

The serial port is controlled and accessed through registers in the peripheral file. These registers are listed in Table 3–17. Figure 3–32 contains a block diagram of the serial port registers and functional blocks.

Table 3–17.Serial Port Control Registers

Register TMS70Cx2	Name	Type	Function
P20 > 0114	SMODE	READ/WRITE	Format/Communication Mode
P21 > 0115	SCTL0	READ/WRITE	Serial Port Control 0
P22 > 0116	SSTAT	READ	Serial Port Status
P23 > 0117	T3DATA	READ/WRITE	Timer 3 Data
P24 > 0118	SCTL1	READ/WRITE	Serial Port Control 1
P25 > 0119	RXBUF	READ	Receiver Buffer
P26 > 011A	TXBUF	WRITE	Transmitter Buffer

The serial mode register, **SMODE**, is the RX/TX control register that describes the character format and type of communication mode (asynchronous, isosynchronous, or serial I/O).

The serial port control 0 register, **SCTL0**, is the RX/TX control register used to control the serial port functions, TX and RX enable, clearing of error flags, and S/W enable.

The serial port status register, **SSTAT**, is the read-only serial status register used to report the serial port status.

The **T3DATA** register is the read/write Timer 3 data register.

The serial port control register 1, **SCTL1**, is used to control the source of SCLK, multiprocessor communication, Timer 3 interrupt, and the Timer 3 prescaler value.

RXBUF is a read-only register containing data from RX. RXBUF is double-buffered with the internal shift register (**RXSHF**) so that the the CPU has at least a full frame to read the received data before RX can overwrite it with new data.

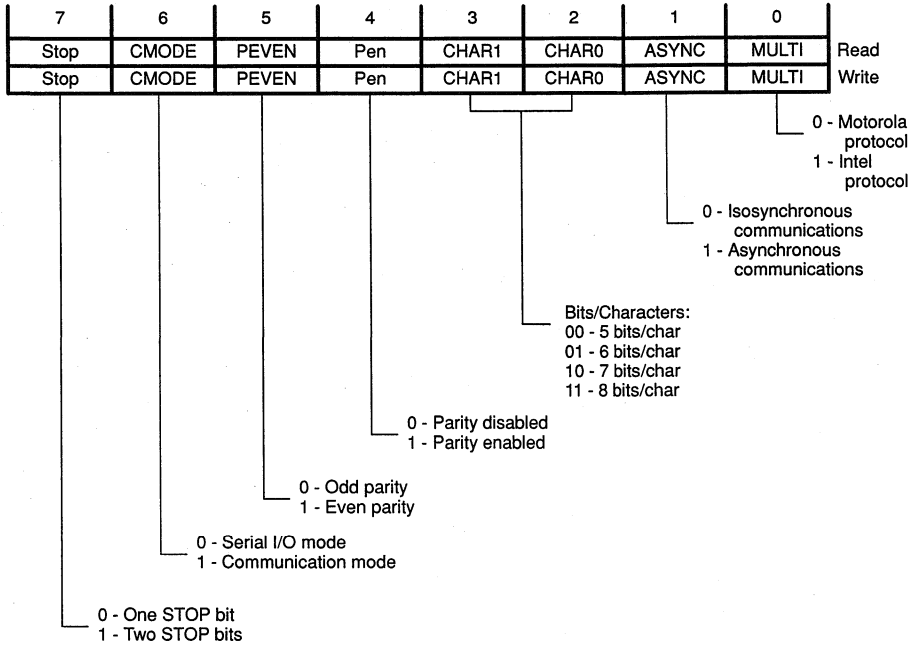
TXBUF is a write-only register from which TX takes the data it transmits. It is double-buffered with the TX shift register (**TXSHF**), so that the CPU has a full frame to write new data before TXBUF becomes empty.

The TXD and RXD lines use I/O pins B3/TXD and A5/RXD, respectively. This configuration allows the TXD and RXD pins to be used as I/O pins if desired. If serial port transmission is disabled, then TXD follows B3. If reception is disabled, then no receiver interrupts occur and A5 functions as a general-purpose I/O pin. The B3 I/O pin must be set to a 1 in order to enable the TXD pin.

3.8.1.1 Serial Mode Register (SMODE)

The SMODE register is the RX/TX control register that describes the character format and type of communication mode (asynchronous, isosynchronous, or serial I/O). SMODE is accessed anytime through P20 in the peripheral file; SMODE is not affected by $\overline{\text{RESET}}$. When configured in the serial I/O mode, bits 7, 5, 4, 1 and 0 are don't care.

Figure 3–33. Serial Mode Register — SMODE



Multiprocessor Mode (MULTI) Bit 0

There are two possible multiprocessor protocols, Motorola (subsection 3.8.3.1) and Intel (subsection 3.8.3.2).

0 — Selects the Motorola protocol.
1 — Selects the Intel protocol.

The Motorola mode is typically used for normal communications since the Intel mode adds an extra bit to the frame. The Motorola mode does not add this extra bit and is compatible with RS-232-type communications. Multiprocessor communication is different from the other communication modes because it uses wake-up and sleep functions.

Communications Mode (ASYNC) Bit 1

This bit determines the serial port communication mode.

0 — Selects isosynchronous mode (subsection 3.8.2.2). In this mode, the bit period is equal to the SCLK period; bits are read on a single value basis.
1 — Selects asynchronous mode (subsection 3.8.2.1). In this mode the bit period is 8 times the SCLK period and bits are read on a two out of three majority basis.

Number of Bits per Character (CHAR1, CHAR2) Bits 2, 3

Character length is programmable to 5, 6, 7 or 8 bits. Characters less than 8 bits are right-justified in buffers RXBUF and TXBUF and padded with leading zeros. The unused leading bits in TXBUF may be written as don't cares. The RXBUF and TXBUF register formats are illustrated in Figure 3–38 and Figure 3–39.

Parity Enable (PEN) Bit 4

If parity is disabled (PEN set to 0), then no parity bit is generated during transmission or expected during reception. A received parity bit is not transferred to RXBUF with the received data because it is not considered one of the data bits when programming the character field. The parity error flag may be set even though parity is disabled.

Parity Even (PEVEN) Bit 5

If PEN is set, then this bit defines odd or even parity according to an odd or even number of 1 bits in both transmitted and received characters.

0 — Sets odd parity.
1 — Sets even parity.

Serial I/O or Communication Mode (CMODE) Bit 6

This bit determines whether the serial port operates in serial I/O mode or one of the communication modes.

0 — Puts the serial port in serial I/O mode which allows easy I/O expansion by using external shift registers.
1 — Selects communication mode. The ASYNC bit (bit 1) determines whether the serial port is in asynchronous or isosynchronous mode. The MULTI bit (bit 0) determines if the communication uses the Motorola or Intel protocol.

Number of Stop Bits (STOP) Bit 7

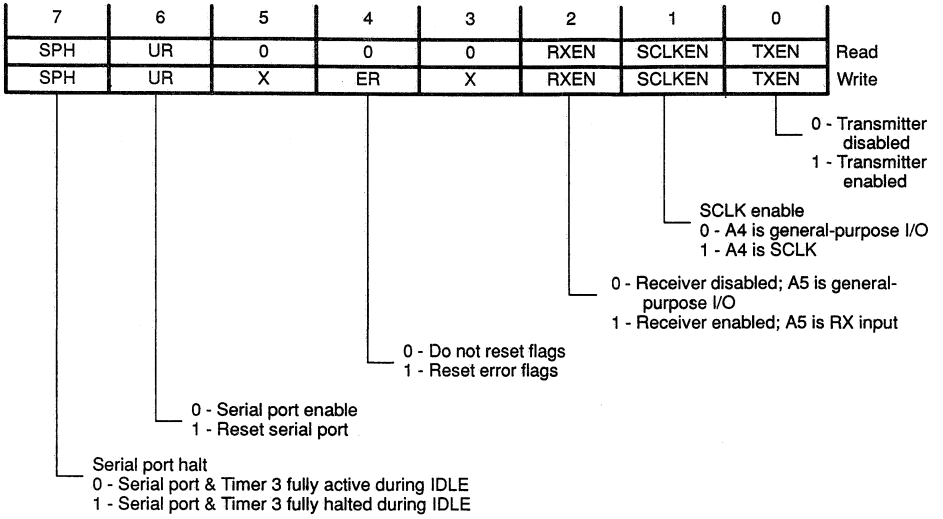
This bit determines the number of stop bits sent when the serial port is in isosynchronous or asynchronous mode.

0 — Selects one stop bit.
1 — Selects two stop bits. The receiver checks for one stop bit only.

3.8.1.2 Serial Control Register 0 (SCTL0)

The SCTL0 register is the RX/TX control register used to control the serial port functions, TX and RX enable, clearing of error flags, and S/W reset. SCTL0 is cleared by a hardware or software reset. SCTL0 is a *read/write* register and is accessed anytime through P21 of the peripheral file.

Figure 3–34. Serial Control 0 Register — SCTL0



Transmit Enable (TXEN) Bit 0

Data transmission through TXD (pin B3) cannot take place unless TXEN is set to a 1 and data register B3 is set to a 1.

When TXEN is reset to 0, transmission does not halt until all the data previously written to TXBUF is sent. Thereafter, B3/TXD can be used as general-purpose output. TXEN is set to 0 by a hardware or software reset.

In isosynchronous mode, if an internally generated SCLK is used, the SCLK output at pin A4 is enabled. When the entire frame is transmitted, TX disables SCLK and sets TXRDY and INT4 flag to a 1, and TXEN to 0.

Serial Clock Enable (SCLKEN) Bit 1

This bit determines if the A4/SCLK pin will be used as general-purpose I/O (bit 1 = 0), or as the serial clock SCLK pin (bit 1 = 1).

Receive Enable (RXEN) Bit 2

In the **communication modes** (asynchronous and isosynchronous):

0 — Prevents received characters from being transferred into RXBUF, and no RXRDY interrupt is generated. However, the receiver shift register (RXSHF) continues to assemble characters. Thus, if RXEN is set during character reception, the complete character will be transferred into RXBUF.

1 — Enables RX (receiver) to set INT4 flag and enable RXRDY.

In **Serial I/O mode**:

0 — The UR bit sets RXEN to 0.
1 — Enables RX operation.

In isosynchronous mode, if an internally generated SCLK is used, the SCLK output at pin A4 is enabled. When the entire frame is received, RX disables SCLK and sets RXRDY and INT4 flag to a 1, and RXEN to 0. RXEN has no direct affect on RXRDY or INT4 flag in this mode.

Error Reset (ER) Bit 4

The error reset bit is used to reset any error flags during serial port operation.

0 — No error flags are affected.
1 — Clears all three error flags in the SSTAT register (PE, OE, FE).

Software UART Reset (UR) Bit 6

Writing a 1 to this bit either directly or as a result of any write to the SMODE register, puts the serial port into the reset condition. SCLK (pin A4) is put in the high-impedance input state. The TXD signal is held at 1 so the B3 pin may be used as a general-purpose output line. The A5/RXD signal becomes a general-purpose I/O line. To go into low-power mode, during an IDLE instruction, UR must be set to zero.

Until a 0 is written to UR, all affected logic is held in the reset state. UR must be set to 0 before the CPU can write a 1 to CLK and output SCLK on port A. UR is set to 1 by hardware $\overline{\text{RESET}}$. The UART reset affects only the items above; it is not a general device reset like the $\overline{\text{RESET}}$ pin.

Serial Port Halt Enable (SPH) Bit 7

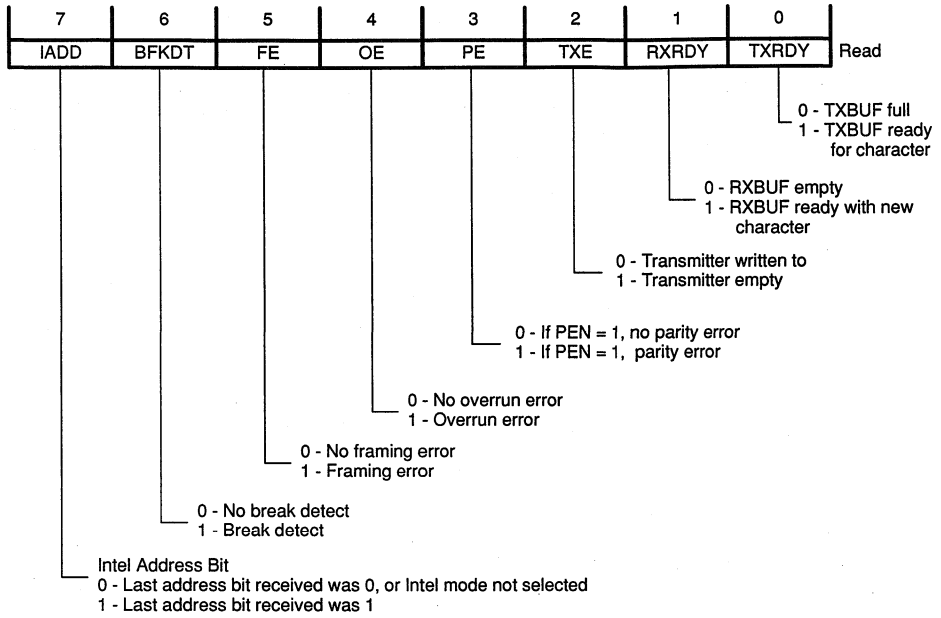
This bit determines if the serial port and Timer 3 will be active or not during an IDLE instruction.

0 — Serial port and Timer 3 will be fully active during an IDLE instruction.
1 — Serial port and Timer 3 will be halted during an IDLE instruction.

3.8.1.3 Serial Port Status Register (SSTAT)

SSTAT is a read-only register and is accessed through P22 of the peripheral file. It is used for reporting the status of the serial port. Bit 7 (IADD) test value of the last address bit received when using the INTEL multiprocessor mode. Bits 1 and 5 are cleared by $\overline{\text{RESET}}$. Bits 0 and 2 are set to 1 by reset. Bits 3, 4, 5 and 7 are not affected by $\overline{\text{RESET}}$. A write to SSTAT will have no effect on any bits in the register.

Figure 3–35. Serial Port Status Register — SSTAT



Transmitter Ready (TXRDY) Bit 0

The TXRDY bit is set by the transmitter to indicate that TXBUF is ready to receive another character. It is automatically reset when a character is loaded. If the serial port interrupt (INT4) is enabled, it is issued at the same time the TXRDY bit is set. Resetting the UART sets TXRDY to 1.

Receiver Ready (RXRDY) Bit 1

This bit is set by the receiver to indicate that RXBUF is ready with a new character. It is automatically reset when the character is read out. If the serial port interrupt (INT4) is enabled, it is set at the same time that the RXRDY bit is set. Resetting the UART sets RXRDY to 1.

Transmitter Empty (TXE) Bit 2

The TXE bit is set to 1 when the transmitter shift register (TXSHF) and TXBUF (shown in Figure 3–39, page 3-79) are empty. It is reset to 0 when the TXBUF is written to. Resetting the UART sets TXE to 1.

Parity Error (PE) Bit 3

PE is set when a character is received with a mismatch between the number of 1s and its parity bit. This bit is reset by the ER bit in SCTL0. Disabling the parity does not disable this flag, so this flag may be set even when the parity is disabled.

Overrun Error (OE) Bit 4

OE is set when a character is transferred into RXBUF (shown in Figure 3–39) before the previous character has been read out. The previous character is overwritten and lost. OE is reset by the ER bit in SCTL0.

Framing Error (FE) Bit 5

FE is set when a character is received with a 0 stop bit, meaning that synchronization with the start bit has been lost and the character is incorrectly framed. The ER bit in SCTL0 resets FE.

Break Detect (BRKDT) Bit 6

The BRKDT bit shows that a break condition has occurred. BRKDT is set if the RXD line remains continuously low for 10 bits or more, starting from the end of a frame (stop bit). When the break ends, BRKDT is set to a 0 immediately. In the serial I/O mode, BRKDT remains a 0. UR (SCTL0 bit 6) sets BRKDT to 0. A break is generated by setting port B bit 3 low. Setting B3 high again resumes TXD operation.

The TXD and RXD lines are multiplexed on I/O lines B3 and A5, respectively. This configuration allows the TXD and RXD pins to be used as I/O pins if desired. If transmission is disabled, then TXD follows B3. If reception is disabled, then no receiver interrupts occur and A5 is an input bit.

Intel Address Bit (IADD) Bit 7

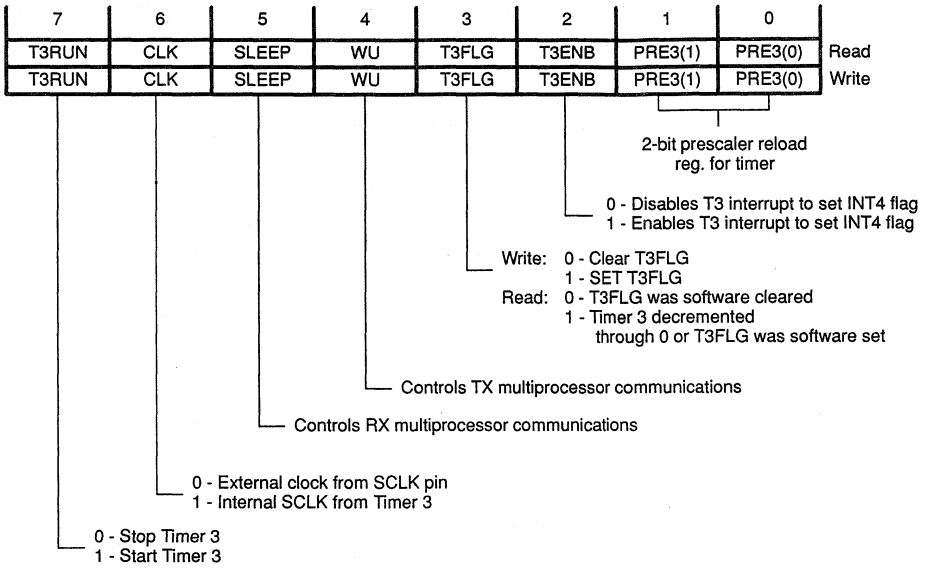
This bit shows the last data bit received when using the Intel protocol.

0 — Last address bit received was 0, or Intel mode was not selected.
1 — Last address bit received was 1.

3.8.1.4 Serial Control Register 1 (SCTL1)

The SCTL1 register is the read/write serial control register 1. It is used to control the Timer 3 start/stop function, the source of SCLK, multiprocessor communication, Timer 3 interrupt, and the Timer 3 prescaler value. SCTL1 is accessed anytime through P24 of the peripheral file; bits 2, 3, 4, 5, and 6 are cleared by RESET. Bits 0, 1, and 7 are not affected by RESET.

Figure 3–36. Serial Port Control 1 Register — SCTL1



Timer 3 Prescale Reload Register (PRE3(1), PRE3(0)) Bits 0,1

These are the prescale bits for Timer 3. The internal clock input to Timer 3 is either f_{osc}/4, /8, /16, or /32, depending on how the prescale bits are set. The Timer 3 output divided by 2 is the actual baud rate for the isosynchronous mode; divided by 8, it is the baud rate for for the asynchronous mode.

Timer 3 Interrupt Enable (T3ENB) Bit 2

When T3ENB is set to 1, Timer 3 sets INT4FLG to 1 when it sets T3FLG to 1. T3ENB is reset to 0 by a hardware reset, but not by UR (SCTL0 bit 6). This allows Timer 3 to operate independently of the serial port.

Timer 3 Interrupt Flag (T3FLG) Bit 3

The T3FLG bit is set to 1 when both the Timer 3 prescaler and Timer 3 decrement through zero together. T3FLG indicates that Timer 3 caused the serial port interrupt. T3FLG must be cleared by software in the T3 interrupt service routine, since it is not cleared when the INT4 vector is fetched by the CPU. T3FLG is reset to 0 by a hardware reset, but not by UR (SCTL0 bit 6). This allows Timer 3 to operate independently of the serial port.

Wake-Up (WU) Bit 4

The WU bit controls the TX features of the multiprocessor communication modes (see subsection 3.8.2.1 and subsection 3.8.2.2). Resetting the UART sets WU to 0; it cannot be set again until UR is cleared.

Sleep (SLEEP) Bit 5

The SLEEP bit controls the RX features of the multiprocessor modes (See subsection 3.8.2.1 and subsection 3.8.2.2). Resetting the UART sets SLEEP to 0.

Serial Clock Source (CLK) Bit 6

The CLK bit determines the SCLK source. Resetting the UART sets CLK to 0; it cannot be set again until UR is cleared.

0 — Selects an external SCLK, which is input on the high-impedance pin A4/SCLK.

1 — Selects an internal SCLK, derived from Timer 3. This signal is output on the low impedance SCLK line.

Timer 3 Start (START) Bit 7

This bit controls the starting and stopping of Timer 3.

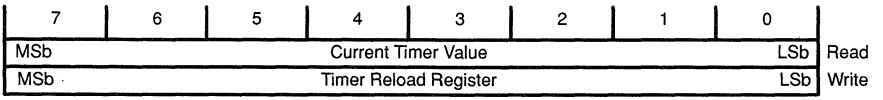
0 — Stops Timer 3.

1 — Loads Timer 3 with the Timer 3 data value and then starts the timer. Writing a 1 will have no effect if Timer 3 is already active.

3.8.1.5 Timer 3 Data Register

The Timer 3 data register, T3DATA, is a read/write register used to store the countdown value of Timer 3. T3DATA is accessed anytime through P23 of the peripheral file.

Figure 3–37. Timer 3 Data Register — T3DATA



3.8.1.6 Receiver Buffer

The receiver buffer, RXBUF, is a read-only register used to store the current RX data. Writing has no direct effect on this register. Data in the RXBUF is right justified, padded with leading 0s. RXBUF is accessed through P25 of the peripheral file.

Figure 3–38. Receive Buffer — RXBUF

7	6	5	4	3	2	1	0
MSb	Receiver Data						LSb
0	0	0	←	5 Data Bits			→
0	0	←	6 Data Bits				→
0	←	7 Data Bits					→
←	8 Data Bits						→

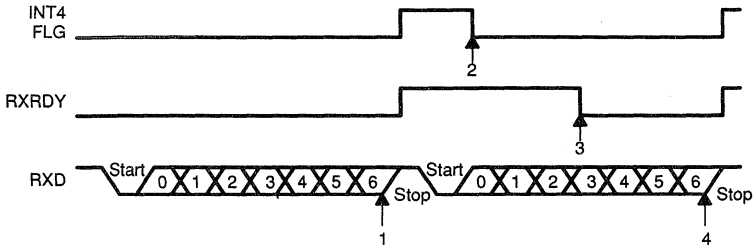
3.8.1.7 Transmitter Buffer

The transmitter buffer, TXBUF, is a write-only register used to store data bits to be transmitted by TX. Data written to TXBUF must be right justified because the left-most bits will be ignored for characters less than eight bits long. TXBUF is accessed through P26 of the peripheral file.

Figure 3–39. Transmitter Buffer — TXBUF

7	6	5	4	3	2	1	0
MSb	Transmitter Data						LSb
X	X	X	←	5 Data Bits			→
X	X	←	6 Data Bits				→
X	←	7 Data Bits					→
←	8 Data Bits						→

3.8.1.8 RX Signals in Communication Modes

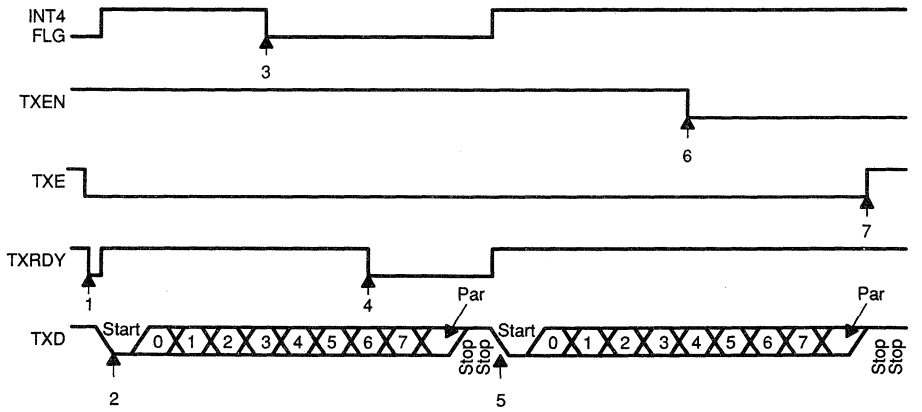


Notes: 1) Format shown is start bit + seven data bits + stop bit.
2) SCLK is continuous, external or internal.
3) If RXEN = 0, RXSHF still receives data from RXD. However, the data is not transferred to RXBUF and RXRDY and INT4FLG are not set.

Sequence of Events:

1) RXSHF data is transferred to RXBUF. Error status bits are set if an error is detected.
2) Software writes to INT4CLR to clear INT4FLG. If not, CPU clears.
3) INT4FLG on entry to level 4 interrupt routine.
4) Software reads RXBUF.

3.8.1.9 TX Signals in Communication Modes

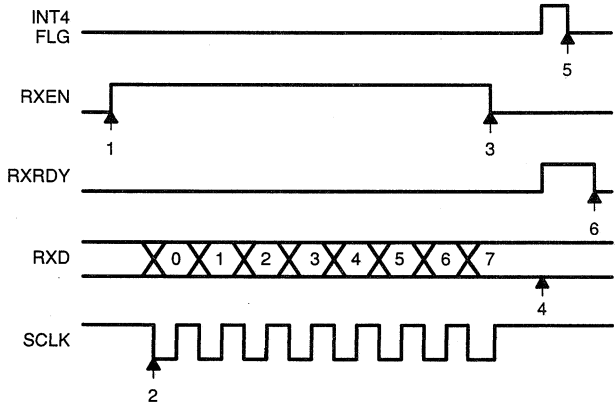


Notes: 1) Format shown is start bit + eight data bits + parity bit + two stop bits.
2) SCLK is continuous whether internal or external.

Sequence of Events:

1) Software writes to TXBUF.
2) TXBUF and WU data are transferred to TXSHF and WUT. INT4FLG and TXRDY are set.
3) Software writes to INT4CLR to clear INT4FLG or CPU clears INT4FLG on entry to level 4 interrupt routine.
4) Software writes to TXBUF.
5) Software writes to INT4CLR to clear INT4FLG or CPU clears INT4FLG on entry to level 4 interrupt routine.
6) Software resets TXEN; current frame will finish and transmission will stop whether TXBUF is full or empty.
7) TXE is set if TXBUF and TXSFT are empty.

3.8.1.10 RX Signals in Serial I/O Modes

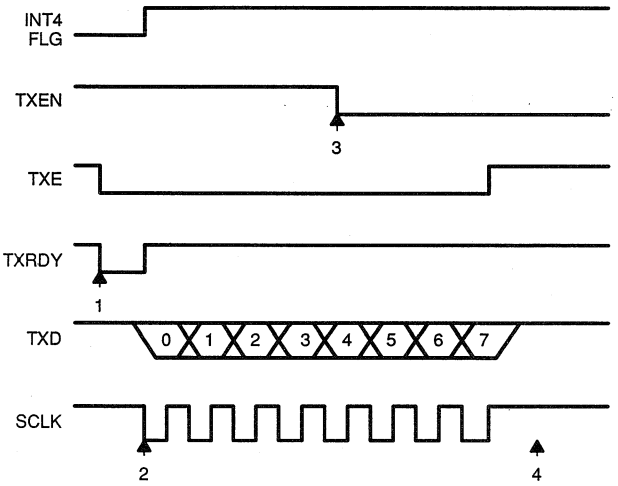


Notes: 1) RXEN has no effect on INT4FLG or RXRDY in Serial I/O mode.
2) RXD is sampled on SCLK rise; external shift registers should be clocked on SCLK fall.
3) The SCLK source should be internal as it is gated by internal circuitry.

Sequence of Events:

1) Software starts receiving by setting RXEN.
2) Gated SCLK starts and data is received.
3) RXEN is automatically cleared in last data bit.
4) RXSHF data is transferred to RXBUF, and RXRDY and INT4 are set.
5) Software writes to INT4CLR to clear INT4FLG; if not, CPU clears INT4FLG on entry to level 4 interrupt routine.
6) Software reads RXBUF.

3.8.1.11 TX Signals in Serial I/O Modes



Notes: 1) Format shown is eight data bits.
2) The SCLK source should be internal as it is gated by internal circuitry.

Sequence of Events:

1) Software writes to TXBUF.
2) TXBUF data is transferred to TXSFT; INT4FLG and TXRDY are set, and SCLK starts.
3) Software resets TXEN, current frame will finish and transmission will halt whether TXBUF is full or empty.
4) Frame ends and SCLK stops because TXEN = 0.

3.8.2 Serial Port Clock Sources

The serial port can be clocked by Timer 3 or an external baud rate generator. The source of the serial clock (SCLK) is determined by the CLK bit (SCTL1 bit 6) and the SCLKEN bit (SCTL0 bit 1).

SCLKEN	CLK	Serial Port Clock Operation
1	1	A4 is forced to output mode, independent of the data direction register (P5). Timer 3 provides the clock for the serial port which is output as SCLK on A4.
1	0	A4 is forced to input mode, independent of the data direction register (P5). An external signal applied to A4 provides the baud rate clock for the serial port.
0	1	A4 is available for general-purpose I/O. The clock for the serial port is provided by Timer 3 but is not output on any pin.
0	0	A4 is selected as general-purpose I/O with its direction register controlling the direction of A4. The serial port clock is taken from the A4 pin, so the clock can be provided by an external signal if the pin is in input mode (the same as the SCLKEN = 1, CLK = 0 option above), or by software if the pin is in output mode by writing to the A4 data register.

If SCLKEN is changed from 1 to 0, A4 will have the direction selected by the A port direction register.

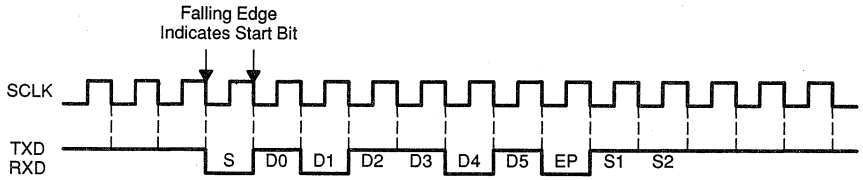
In any of these modes, reading from A4 will return the value present at the pin. SCLKEN and CLK are both set to 0 by $\overline{\text{RESET}}$. The A4 direction register is also set to 0 (input) by $\overline{\text{RESET}}$.

3.8.2.1 Asynchronous Communication Mode

In asynchronous communication mode, the frame format consists of a start bit, five to eight data bits, an even/odd/no parity bit, and one or two stop bits. The bit period is eight times the SCLK period.

Receiving a valid start bit initiates RX operation. A valid start bit consists of a negative edge followed by three samples, two of which *must* be zero. If two of the three samples are not zero, then the receiver continues to search for a Start bit. These samples occur three, four, and five SCLK periods after the negative edge. This sequence provides false start bit rejection and also locates the center of bits in the frame where the bits will be read on a majority (two out of three) basis. Figure 3–40 illustrates the asynchronous communication format, with a start bit showing how edges are found and majority vote taken.

Figure 3–40. Asynchronous Communication Format



Since RX synchronizes itself to frames, the external transmitting and receiving devices do not have to use the same SCLK; it may be generated locally. If the internal SCLK is used it is output continuously on pin A4/SCLK.

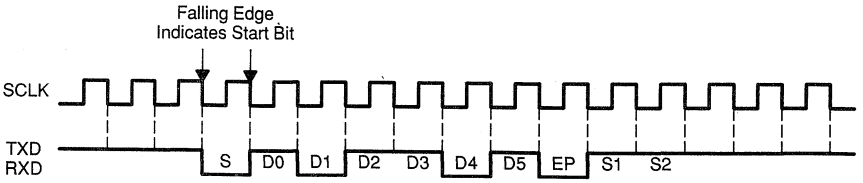
3.8.2.2 Isosynchronous Communication Mode

Isosynchronous communication mode is a hybrid protocol, combining features of the asynchronous mode and the serial I/O mode. The isosynchronous frame format is the same as the asynchronous mode frame format, consisting of a start bit, five to eight data bits, an even/odd/no parity bit, and one or two stop bits. However,it uses only one serial clock (SCLK) cycle per data bit as compared to 8 SCLKs per data bit for asynchronous mode. This allows much faster transmission rates than asynchronous mode. The bit period equals the SCLK period, as it does in serial I/O mode. Bits are read on a single value basis. Since the RX does not synchronize itself to the data bits, the transmitter and receiver must be supplied with a common SCLK. The benefit of the Isosynchronous mode is that the frame format can be configured like the asynchronous mode, yet the baud rate is that of the serial I/O mode.

Receiving a valid start bit, which consists of a negative edge, initiates RX operation. Since RX does not synchronize itself to data bits, the transmitter and receiver must be supplied with a common SCLK. An internal or external synchronizing clock must be supplied from either the internal Timer 3 or an external clock source on A4/SCLK. If the internal SCLK is used, it is output continuously on A4/SCLK.

Figure 3–41 illustrates the isosynchronous communication format, with a complete frame consisting of a start bit, six data bits, even parity, and two stop bits.

Figure 3–41. Isosynchronous Communication Format



In both the asynchronous and isosynchronous communication modes, when a frame is fully received, RXBUF is loaded from RXSHF, RXRDY, and INT4 flag

are set to 1, and the error status bits are set accordingly. RXRDY is reset to 0 when the CPU reads RXBUF.

Transmission is initiated after the CPU writes to TXBUF. This sets TXE to 0. TXSHF is loaded from TXBUF, setting TXRDY and INT4 flag to 1. After completing the transmission, TXSHF reloads if TXBUF is full; if not, TX idles and TXE is 1 until TXBUF is written to. Bit 3 of Port 3 must be set to a 1 to enable data transmission through the B3/TXD pin.

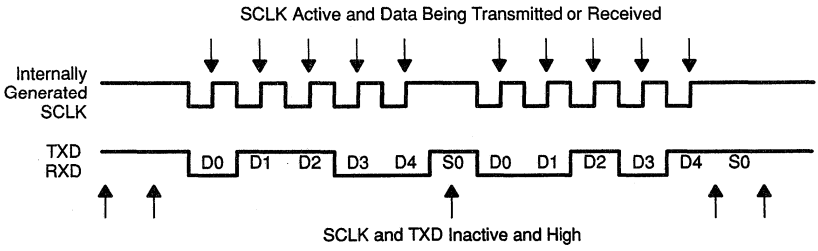
3.8.2.3 Serial I/O Mode

In serial I/O mode, the frame format is five to eight data bits and one stop bit, with no corresponding clock cycle for the stop bit. An external or internal synchronizing clock signal must be supplied from either the internal Timer 3 or an external clock. An external clock must be supplied if the external SCLK option is used. The bit period is equal to the SCLK period. TX operation is initiated by writing to TXBUF when TXRDY equals 1. RX operation is initiated by writing a 1 to the RXEN bit. When the receiver has received a full frame, the RXEN bit is automatically cleared, disabling the receiver. The transmitter starts operating when the TX enable bit (TXEN) is set to 1. Data is written to TXBUF when TXRDY equals 1. Unlike the receiver, the TXEN bit is not automatically cleared when the transmitter finishes a full frame.

To start the receiver and transmitter at the same time, first write the transmitter data to TXBUF and then set both RXEN and TXEN in one instruction. Be careful that the enable bits are not set when Timer 3 rolls over past 0. This can be done by adjusting the timer rate before the bits are enabled and then setting the timer to the correct rate after enabling.

Figure 3–42 illustrates the serial I/O format for two back-to-back frames, each containing five data bits.

Figure 3–42. Serial I/O Communication Format



If an internal SCLK is selected, it will be output on pin A4/SCLK. In serial I/O mode, SCLK is only active when data is being transmitted or received; otherwise, SCLK has a value of one.

3.8.3 Multiprocessor Communication

When the serial port is in either the asynchronous or isosynchronous communications mode, the multiprocessor communication formats are available. These formats efficiently transfer information between many microcomputers on the same serial link. Information is transferred as a block of frames from a particular source to some destination(s). The serial port has features to identify the start of a block of data, and suppress interrupts and status information from RX until a block start is identified.

In both multiprocessor modes the sequence is:

1) The serial port wakes up at the start of a block and reads the first frame (containing the destination address).
2) A software routine is entered through either an interrupt or polling routine and checks the incoming data byte against its address byte stored in memory.
3) If the block is addressed to the microcomputer the CPU reads the rest of the block; if not, the software routine puts the serial port to sleep again and therefore will not receive serial port interrupts until the next block start.

On the serial link, all processors set their SLEEP bit to 1 so that they will only be interrupted when the address bit in the data stream is a 1. When the processors receive the address of the current block, they compare it to their own addresses and those processors which are addressed set their SLEEP bit to a 0, so that they will read the rest of the block.

Although RX still operates when the SLEEP bit is 1, it will not set RXRDY, INT4 flag, or the error status bits to 1 unless the address bit in the received frame is a 1. The RX does not alter the SLEEP bit; this must be done in software.

To provide more flexibility, the serial port implements two multiprocessor protocols, one supported by Motorola and the other by Intel. The Motorola protocol is compatible with the Motorola MC6801 processor modes and the Intel protocol is compatible with the Intel protocol for the 8051. The multiprocessor mode is software selectable via the MULTI bit in the SMODE register (Figure 3–33). Both formats use the WU and SLEEP flags to control the TX and RX features of these modes.

Because the Intel multiprocessor mode contains an extra address/data bit, it is not as efficient as the Motorola mode in handling blocks containing more than 10 bytes of data. The Intel mode is more efficient in handling many small blocks of data because it does not have to wait between blocks of data as does the Motorola mode.

3.8.3.1 Motorola (MC6801) Protocol

In this protocol, blocks are separated by having a longer idle time between the blocks than between frames in the blocks. An idle time of 10 or more bits after a frame indicates the start of a new block.

The processor wakes up (serial port resets the SLEEP bit to 0) after the block start signal. The processor now recognizes the next serial port interrupt. The service routine then receives the address sent out by the transmitter and compares this address to its own. If the CPU is addressed, the service routine does not set the SLEEP bit, and receives the rest of the block. If the CPU is not addressed, the service routine sets the SLEEP bit (in software) to a 1. This lets the CPU continue to execute its main program without being interrupted by the serial port. The serial port sets the SLEEP bit to 0 whenever it detects a block start signal.

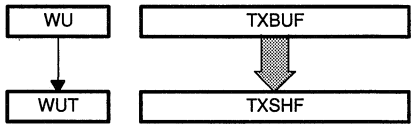
There are two ways to send a block start signal.

1) The first is to deliberately leave an idle time of 10 bits or more by delaying the time between the transmission of the last frame of data in the previous block and the address frame of the new block.

2) In the second method, the serial port implements a more efficient method of sending a block start signal. Using the Wake-Up (WU) bit, an idle time of exactly one frame (timed by the serial port) can be sent. The serial communications line is therefore not idle any longer than necessary.

Associated with the WU bit is the wake-up temporary (WUT) flag. WUT is an internal flag, double buffered with WU. When TXSHF is loaded from TXBUF, WUT is loaded from WU, and WU is reset to 0. This arrangement is shown in Figure 3–43.

Figure 3–43. Double-Buffered WUT and TXSHF



Sending out a block start signal of exactly one frame time is accomplished as follows:

1) Write a 1 to the WU bit.

2) Write a data word (don't care) to TXBUF.

3) When TXSHF is free again, TXBUF's contents are shifted to TXSHF, and the WU value is shifted to WUT.

4) If WU was set to a 1, the start, data, and parity bits are suppressed and an idle period of one frame, timed by the serial port, is transmitted.

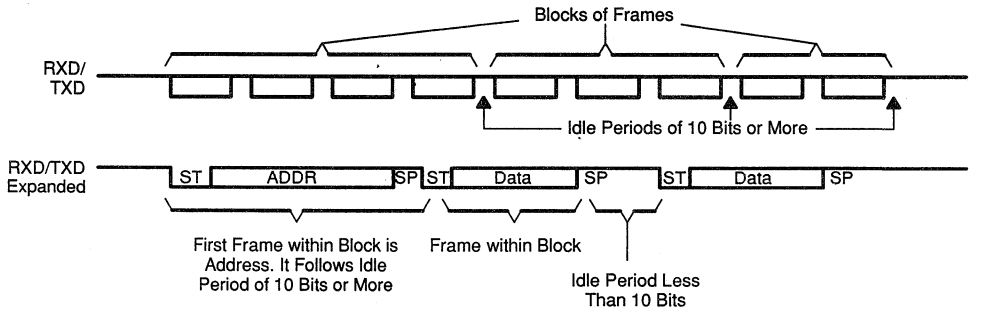
5) The next data word, shifted out of the serial port after the block start signal, is the second data word written to the TXBUF after writing a 1 to the WU bit.

6) The first data word written is suppressed while the block start signal is sent out, and ignored after that.

Writing the first don't care data word to the TXBUF is necessary so the WU bit value can be shifted to WUT. After the don't-care data word is shifted to the TXSHF, the TXBUF (and WU if necessary) may be written to again, since WUT and TXSHF are both double-buffered.

Although RX still operates when the SLEEP bit is 1, it will not set RXRDY, INT4 flag, or the error status bits to 1. The RX will set the SLEEP bit to 0 if it times an appropriate 10-bit idle time on RXD. The Motorola multiprocessor communication format is shown in Figure 3–44.

Figure 3–44. Motorola Multiprocessor Communication Format

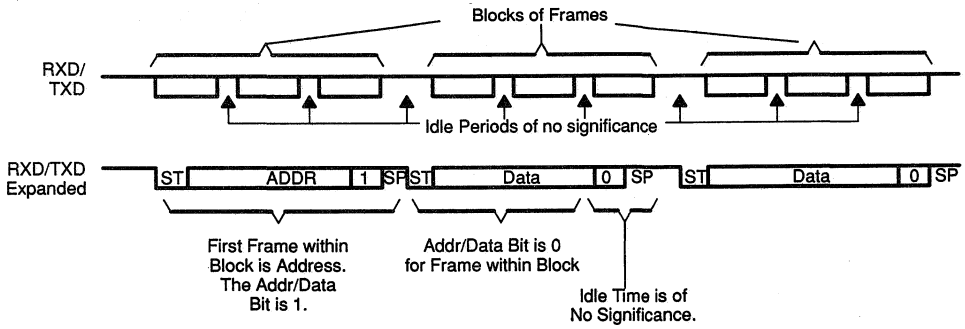


3.8.3.2 Intel (I8051) Protocol

In the Intel protocol, the frame has an extra bit called an address bit just before the parity bit. Blocks are distinguished by the first frame(s) in the block with the address bit set to 1, and all other frames with the address bit set to 0. The idle period timing is irrelevant.

The WU bit sets the address bit. In TX, when the TXBUF and WU are loaded into TXSHF and WUT, WU is reset to 0 and WUT is the value of the address bit of the current frame. Thus, to send an address, set the WU bit to a 1, and write the appropriate address value to the TXBUF. When this address value is transferred to TXSHF and shifted out, its address bit is sent as a 1, which flags the other processors on the serial link to read the address. Since TXSHF and WUT are both double-buffered, TXBUF and WU may be written to immediately after TXSHF and WUT are loaded. To transmit non-address frames in the block, the WU bit must be left at 0. *On the TMS70Cx2 devices, the received address bit is also placed in the SSTAT IADD bit.*

Figure 3–45. Intel Multiprocessor Communication Format



3.8.4 Serial Port Initialization

The serial port must be initialized before it can be used; then it may be operated by simply reading and writing to peripheral-file registers. A good programming practice is not to assume that any registers have particular values at power-up or reset. A program should write to every value or register that might affect the serial port. Initialize the serial port as follows:

❑ **TMS70Cx2, TMS77C82, and TMS70Cx8**

1) Set B3 data value to 1. This allows the TXD line to transmit.

2) Write to the SMODE register (P20). This resets the serial port (writes a 1 to the UR bit in SCTL0) and sets the character format and the type of communication mode.

3) Write to the SCTL0 register (P21). Enable the receiver or the transmitter or both. The UR bit must be set to 0.

4) Load the Timer 3 reload register value (P23).

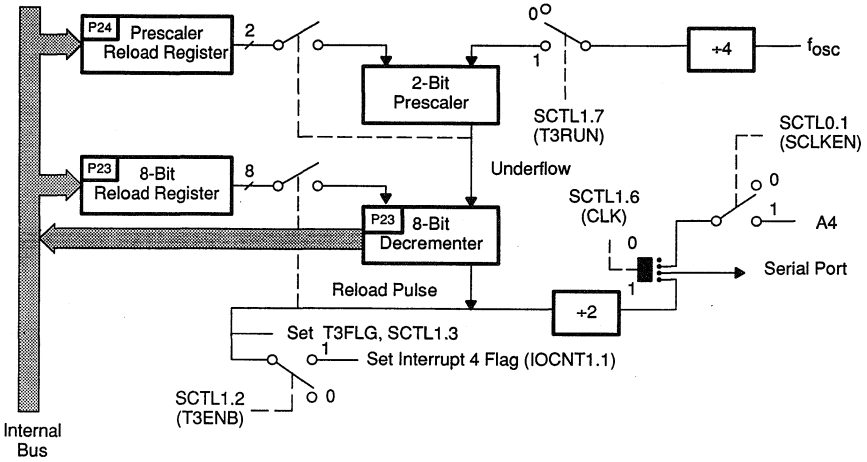
5) Write to SCTL1 register (P24) to initialize Timer 3, the clock source, and multiprocessor mode, if desired.

Once the serial port is initialized it can be operated continuously in the selected operational mode. To send data, simply write to the transmit buffers (P26), making sure that the transmitter is enabled (P21). Take input data from the receive buffer (P25) with the receiver enabled (P21). If the mode must be changed, the serial port must be reset and then re-initialized for the desired mode. The serial port can be reset in three ways: hardware reset (via the RESET pin) or software reset (via the UR bit in SCTL0), or by writing to the SMODE register.

3.8.5 Timer 3

Timer 3, illustrated in Figure 3–46, can be used as a standalone timer or as the internal baud-rate generator.

Figure 3–46. 8-Bit Timer 3 (TMS7xCx2 and TMS70Cx8)



Timer 3 is accessed through T3DATA (P20) and SCTL1 (P21) which is shared with serial port control functions. The clock source for Timer 3 is internal only, and has a period of $2 \times t_{c(C)}$. Timer 3 is a free running clock and is updated with new timer reload values when the prescaler and decrementer pass through zero together. Timer 3 is stopped and started by bit 7 in SCTL1.

Timer 3 consists of a 2-bit prescaler (SCTL1 bits 1 and 0) and an 8-bit decrementer (register T3DATA). When they decrement through zero, both the prescaler and the decrementer are reloaded from the 2-bit and 8-bit reload registers, respectively.

The Timer 3 output goes to the serial port via a ÷ 2 circuit, producing an internal equal mark-space ratio SCLK. The baud rate generated by Timer 3 is user-programmable and is determined by the value of the 2-bit prescaler and the 8-bit timer reload register. The equations for determining the baud rates for both the asynchronous and isosynchronous modes are:

Equation 3–1. Asynchronous Baud Rate, TMS7xCx2 and TMS70Cx8

$$\frac{1}{32 \times (PR + 1) \times (TR + 1) \times t_{c(C)}}$$

Equation 3–2. Isosynchronous Baud Rate, TMS7xCx2 and TMS70Cx8

$$\frac{1}{4 \times (PR + 1) \times (TR + 1) \times t_{c(C)}}$$

where:

$t_{c(C)}$ = $2/f_{osc}$
PR = Timer 3 prescale reload register value
TR = Timer 3 reload register value

For example, to program the serial port to operate at 300 baud in asynchronous mode (with f_{osc} = 8 MHz), the prescaler value is set to 3 and the reload register value is set to 103 decimal, or >67. Other prescaler and timer values for common baud rates are shown in Table 3–18.

Table 3–18. Timer Values for Commonly Used Baud Rates Using Asynchronous Modes — TMS7xCx2 and TMS70Cx8

Baud	3.579454 MHz		4.9152 MHz		7.158908 MHz		8 MHz	
Rate	PS, T	Error	PS, T	Error	PS, T	Error	PS, T	Error
75	3, 186	0.2%	3, 255	.0%	–	–	–	–
110	1, 253	0.1%	3, 174	0.3%	3, 253	0.1%	–	–
300	0, 185	0.2%	0, 255	.0%	2, 123	.0%	3, 103	0.2%
600	0, 92	0.2%	0, 127	.0%	0, 185	0.2%	3, 51	0.2%
1200	0, 46	0.8%	0, 63	.0%	0, 92	0.2%	3, 25	0.2%
2400	0, 22	1.3%	0, 31	.0%	0, 46	0.8%	3, 12	0.2%
4800	0, 11	3.0%	0, 15	.0%	0, 22	1.3%	1, 12	0.2%
9600	0, 5	3.0%	0, 7	.0%	0, 11	3.0%	0, 12	0.2%
19200	0, 2	3.0%	0, 3	.0%	0, 5	3.0%	–	–
38400	–	–	0, 1	.0%	0, 2	3.0%	0, 2	.0%
125000	–	–	–	–	–	–	0, 0	.0%

Note: PS = prescaler; T = timer

The Timer 3 output always sets T3FLG to 1, and sets INT4 flag to 1 if T3ENB is a 1 when the timer and prescaler decrement through 0. This allows Timer 3 to be used as a utility timer if it is not used by the serial port. Timer 3 and its flags are not affected by the serial port software reset, UR, allowing Timer 3 to be used independently of the serial port.

3.8.6 Initialization Examples

This section contains four examples that initialize the serial port. In each case the data is moved to and from the buffers in the interrupt routines.

- The first example shows a typical RS-232 application that connects to a terminal.
- The second demonstrates a system using the serial I/O mode to connect to a shift register.

- The third example uses the baud-rate timer as an additional third timer when the serial port is not used.
- The last example illustrates use of the Intel mode in a multiprocessor application.

In all examples, assume the register mnemonics have been equated (EQU) with the corresponding peripheral-file location.

3.8.6.1 RS-232-C Example

This example transmits and receives data from a standard RS-232-C-type terminal at 9600 baud with a data format of 7 data bits, 2 stop bits and no parity.

```
RS232   DINT                          Precaution
        ORP    %?00001000,PORTB       Enable TX pin
        MOVP   %?00001011,IOCNT1      Enable INT4
        MOVP   %0,P17                 Point to SCTL0
        MOVP   %?00010000,SCTL0       Reset the UART
        MOVP   %?11001010,SMODE       Two stop, 7 data bits, no
*                                     parity, no extra Intel mode bit,
*                                     communications mode
        MOVP   %?00010101,SCTL0       Clear RESET, clear error flags,
*                                     enable TX and RX
        MOVP   %7,T3DATA              Set baud rate to 9600
*                                     (4.9152 MHz crystal)
        MOVP   %?01000000,SCTL1       Internal clock, prescale=0, no
*                                     multiprocessing, disable Timer 3
*                                     interrupt, start Timer 3
        EINT
```

3.8.6.2 Serial I/O Example

This routine sends and receives data from a shift register device at 1200 baud with 8 data bits and no parity.

```
SERIAL  DINT                          Precaution
        ORP    %?00001000,PORTB       Enable TX pin
        MOVP   %?00001011,IOCNT1      Enable INT4
        MOVP   %0,P17                 Point to SCTL0
        MOVP   %?00010000,SCTL0       Reset the UART
        MOVP   %?00001100,SMODE       One stop, 8 data bits,
*                                     no parity, no extra Intel
*                                     mode bit, Serial I/O mode
        MOVP   %?00010101,SCTL0       Clear RESET, clear error
*                                     flags, enable TX and RX
        MOVP   %64,T3DATA             Set baud rate to
*                                     1200 (5MHz crystal)
        MOVP   %?11000000,SCTL1       Internal clock, prescale=0,
*                                     no multiprocessing, disable
*                                     Timer 3 interrupt,
*                                     start Timer 3
        EINT
```

3.8.6.3 Extra Timer with No Serial Port

Timer 3 can be used as an additional timer when the serial port is not needed. INT4 occurs whenever the timer passes 0. The timer period is determined by the value TIME and the prescale bit in SCTL1. Disable the transmitter and receiver to assure no interrupts come from that source. This timer works best as a periodic interrupt, allowing a task to be performed at a fixed interval.

```
TIMER3  DINT                        Precaution
        MOVP  %?00001011,IOCNT1     Enable INT4
        MOVP  %0,P17                Point to SCTL0
        MOVP  %?00010000,SCTL0      Reset the UART
        MOVP  %?01000010,SMODE      Asynchronous communication mode
        MOVP  %?00010000,SCTL0      Clear RESET, clear error
*                                   flags, disable TX and RX
        MOVP  %TIME,T3DATA          Set timer to selected rate
        MOVP  %?110001XX,SCTL1      Internal clock, no
*                                   multiprocessing selected
*                                   prescale, enable Timer 3
*                                   interrupts, start Timer 3
        EINT
```

3.8.6.4 Intel Multiprocessor Example

This example illustrates basic concepts of sending and receiving data in a multiprocessor system. The processors are usually close to each other so they can send at maximum speed without problems. The data is sent and received during the interrupt routines.

```
MULTI   DINT                        Precaution
        ORP   %?00001000,PORTB      Enable TX pin (?=binary)
        MOVP  %?00001011,IOCNT1     Enable INT4
        MOVP  %0,P17                Point to SCTL0
        MOVP  %?00010000,SCTL0      Reset the UART
        MOVP  %?01111111,SMODE      One stop, 8 data bits,
*                                   odd parity, Intel mode bit,
*                                   communications mode
        MOVP  %?00010101,SCTL0      Clear RESET, clear error
                                    flags, enable TX and RX
        MOVP  %0,T3DATA             Set baud rate to full
*                                   speed (5MHz crystal)
        MOVP  %?11100000,SCTL1      Internal clock, prescale=0,
*                                   no multiprocessing, disable
*                                   Timer 3 interrupts, put
*                                   receiver to sleep,
*                                   start Timer 3
        EINT
*
*   Meanwhile, back at the interrupt routines
*
SENDIT  ORP   %BIT4,SCTL1           Send Wake-Up bit
*                                   (Bit4=00010000)
        MOVP  %ADDRS,TXBUF          Send address byte
*                                   wait for the transmit
*                                   complete interrupt . . . . .
        ANDP  %#BIT4,SCTL1          Clear Wake-Up bit
*                                   (# = logical NOT)
        MOVP  %DATA,TXBUF           start sending data bytes
*
*
GETIT   MOVP  RXBUF,A               Get address byte
*                                   (it only interrupts on an
*                                   address byte when sleeping)
        CMP   %ADDRS,A              Is it this processor's address?
        JNE   NOTIT                 If this is not the correct
*                                   address ignore the rest
*                                   of the following data bytes
        ANDP  %#BIT5,SCTL1          Clear Sleep bit and wait for
*                                   additional data bytes
*                                   Some method should determine
*                                   End of Data so that the pro-
*                                   cessor can go back to sleep
*                                   Byte count in first data byte
*                                   or special end of data byte
*                                   are two methods
```

3.8.7 Serial Port Interrupts

INT4 is dedicated to the serial port. Three sources can generate an interrupt through INT4:

1) The transmitter (TX),
2) The receiver (RX), and
3) Timer 3 (T3).

Setting TXEN to 1 allows data loaded into the TXBUF to be shifted into TXSHF. The TX sets TXRDY and INT4 flag to 1 when TXSHF is loaded from TXBUF.

In the communication modes, if RXEN is set to 1, RX sets RXRDY and INT4 flag to a 1 when RXBUF is loaded from RXSHF. If RXEN is 0, RXSHF still receives frames and shifts them into RXBUF, but RXRDY and INT4 flag are held to 0. If a character is in RXBUF, and RXEN is then set to a 1, RXRDY and INT4 flag will be set to 1.

In serial I/O mode, RXEN is set to initiate the reception of a frame. When the last bit of the frame is received RXEN is reset to 0; however, RXRDY and INT4 flag are still set to 1 when the character is shifted from RXSHF to RXBUF. RXRDY and INT4 flag bits are not masked by RXEN.

Timer 3 sets T3FLG and INT4 flag (if T3ENB is 1) when its prescaler and timer decrement through 0 together.

When the CPU acknowledges INT4, RXRDY, TXRDY, and T3FLG are the flags that indicate its source. The INT4 service routine must determine which of these sources caused INT4 in the specific application. For example, if all three are likely sources, the INT4 service routine must check for the following possible situations:

1) RXRDY only
2) TXRDY only
3) T3 only
4) RXRDY, TXRDY, T3
5) RXRDY, TXRDY
6) RXRDY, T3
7) TXRDY, T3
8) None

The last check is necessary because RXRDY, TXRDY, or T3FLG can set INT4 flag. It is possible that one or more interrupts may occur between CPU acknowledgement of INT4 and INT4 service routine testing of RXRDY, TXRDY, and T3FLG. The CPU clears the INT4 flag bit when it acknowledges INT4. If a second INT4 source is set in the time between this clearing and the software testing, the second or third interrupts will be serviced by the current INT4 service routine. Thus, when INT4 is again acknowledged (INT4 flag was set again by the second interrupt) RXRDY, TXRDY, and T3FLG will all be set to 0.

Chapter 4

Electrical Specifications

This chapter contains electrical and timing information for each category of TMS7000 family devices. All TMS7000 CMOS devices with the exception of the TMS70CTx0 devices can operate at wide voltage and frequency ranges; therefore, the CMOS specifications are presented using two separate test voltage ranges.

CMOS Devices:

Section		Page
4.1	TMS70C00, TMS70C20, and TMS70C40 Specifications (Wide Voltage)	4-2
4.2	TMS70C00, TMS70C20, and TMS70C40, Specifications (5V ±10%)	4-10
4.3	TMS70CT20 and TMS70CT40 Specifications (5V ±10%)	4-17
4.4	TMS70C02, TMS70C42, and TMS70C82 Specifications (Wide Voltage)	4-21
4.5	TMS70C02, TMS70C42, and TMS70C82 Specifications (5V ±10%)	4-31
4.6	TMS70C08 and TMS70C48 Specifications	4-40
4.7	SE70CP160A Specifications	4-47
4.8	TMS77C82 Specifications	4-51
4.9	SE70CP168 Specifications	4-59

4.1 TMS70C00, TMS70C20, and TMS70C40 Specifications (Wide Voltage)

Table 4–1. Absolute Maximum Rating over Operating Free-Air Temperature Range (Unless Otherwise Noted)

Supply voltage, V_{CC}† . . . . . – 0.3 V to 7 V
All input voltages . . . . . – 0.3 V to V_{CC} + 0.3 V
All output voltages . . . . . – 0.3 V to V_{CC} + 0.3 V
Maximum I/O buffer current (per pin) . . . . . ±10 mA
Storage temperature range . . . . . – 55°C to 150°C
I_{CC}, I_{SS} current (maximum into pins 25 and 40) . . . . . ±60 mA
Continuous power dissipation . . . . . 0.5 W
† Unless otherwise noted, all voltages are with respect to V_{SS}.

Stresses beyond those listed under "Absolute Maximum Ratings" 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 beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

Table 4–2. Recommended Operating Conditions

			Min	Nom	Max	Unit
V_{CC}	Supply voltage		2.5		6.0	V
		INT1, INT3, RESET, and XTAL Pins				
V_{IH}	High-level input voltage	5.5 V ≤ V_{CC} ≤ 6 V	V_{CC} – 1.0		V_{CC}	V
		4.5 V ≤ V_{CC} ≤ 5.5 V	V_{CC} – 0.7		V_{CC}	V
		3.5 V ≤ V_{CC} ≤ 4.5 V	V_{CC} – 0.5		V_{CC}	V
		2.5 V ≤ V_{CC} ≤ 3.5 V	V_{CC} – 0.35		V_{CC}	V
		MC Pin				
V_{IH}	High-level input voltage	5 V ≤ V_{CC} ≤ 6 V	V_{CC} – 0.5		V_{CC}	V
		4 V ≤ V_{CC} < 5 V	V_{CC} – 0.4		V_{CC}	V
		3 V ≤ V_{CC} < 4 V	V_{CC} – 0.3		V_{CC}	V
		2.5 V ≤ V_{CC} < 3 V	V_{CC} – 0.2		V_{CC}	V
		Port (All Other Pins)				
V_{IH}	High-level input voltage	5 V ≤ V_{CC} ≤ 6 V	V_{CC} – 1.3		V_{CC}	V
		4 V ≤ V_{CC} < 5 V	V_{CC} – 1.0		V_{CC}	V
		3 V ≤ V_{CC} < 4 V	V_{CC} – 0.7		V_{CC}	V
		2.5 V ≤ V_{CC} < 3 V	V_{CC} – 0.4		V_{CC}	V

Table 4–2. Recommended Operating Conditions (Continued)

			Min	Nom	Max	Unit
INT1, INT3, RESET, and XTAL Pins						
V_{IL}	Low-level input voltage	$5.5\ V \le V_{CC} \le 6\ V$	0		1.00	V
		$4.5\ V \le V_{CC} \le 5.5\ V$	0		0.70	V
		$3.5\ V \le V_{CC} \le 4.5\ V$	0		0.50	V
		$2.5\ V \le V_{CC} \le 3.5\ V$	0		0.35	V
MC Pin						
V_{IL}	Low-level input voltage	$5\ V \le V_{CC} \le 6\ V$	0		0.5	V
		$4\ V \le V_{CC} < 5\ V$	0		0.4	V
		$3\ V \le V_{CC} < 4\ V$	0		0.3	V
		$2.5\ V \le V_{CC} < 3\ V$	0		0.2	V
Port (All Other Pins)						
V_{IL}	Low-level input voltage	$5\ V \le V_{CC} \le 6\ V$	0		1.5	V
		$4\ V \le V_{CC} < 5\ V$	0		1.1	V
		$3\ V \le V_{CC} < 4\ V$	0		0.7	V
		$2.5\ V \le V_{CC} < 3\ V$	0		0.3	V
T_A	Operating temperature range	Commercial (TMS70Cx0N)	0		70	°C
		Industrial (TMS70Cx0NA)	– 40		85	°C

Table 4–3. Electrical Characteristics over Full Range of Operating Conditions

Parameter		Test Conditions	Min	Typ†	Max	Unit
I_I	Input leakage current	$V_{IN} = V_{SS}$ to V_{CC}		±0.10	±5	µA
C_I	Input capacitance			5		pF
V_{OH}	High-level output voltage ‡	V_{CC} = 5.0 V, I_{OH} = – 1 mA	2.5	4.5		V
		V_{CC} = 5.0 V, I_{OH} = – 0.3 mA	4.5	4.8		V
V_{OL}	Low-level output voltage ‡	V_{CC} = 5.0 V, I_{OL} = 1.7 mA		0.3	0.4	V
I_{OH}	Output source current	V_{CC} = 5.0 V V_{OH} = 4.5 V	– 0.3	– 1.4		mA
		V_{CC} = 4.0 V V_{OH} = 3.5 V	– 0.2	– 1.0		mA
		V_{CC} = 3.0 V V_{OH} = 2.5 V	– 0.1	– 0.7		mA
		V_{CC} = 2.5 V V_{OH} = 2.0 V	– 0.05	– 0.3		mA
		V_{CC} = 5.0 V V_{OH} = 2.5 V	– 1.0	– 5.0		mA
I_{OL}	Output sink current	V_{CC} = 5.0 V V_{OL} = 0.4 V	1.7	2.8		mA
		V_{CC} = 4.0 V V_{OL} = 0.4 V	1.2	2.4		mA
		V_{CC} = 3.0 V V_{OL} = 0.4 V	0.7	2.0		mA
		V_{CC} = 2.5 V V_{OL} = 0.4 V	0.2	1.3		mA

† V_{CC} = 5 V, T_A = 25°C

‡ Output levels ensure 400 mV of noise margin over specified input levels.

Table 4–4. Supply Current Requirements

Parameter		Test Conditions	Min	Typ	Max	Unit
I_{CC}	Operating mode	f_{osc} = 6.0 MHz, V_{CC} = 5.0 V		9.0	14.4	mA
		f_{osc} = 3.0 MHz, V_{CC} = 5.0 V		4.5	7.2	mA
		f_{osc} = 0.5 MHz, V_{CC} = 5.0 V		0.8	1.2	mA
		f_{osc} = Z MHz, V_{CC} = 5.0 V		1.5	2.4	mA/ MHz
		f_{osc} = 0.5 MHz, V_{CC} = 2.5 V		370	800	µA
I_{CC}	Wake-up mode (timer active)	f_{osc} = 6.0 MHz, V_{CC} = 5.0 V		960	1920	µA
		f_{osc} = 3.0 MHz, V_{CC} = 5.0 V		480	960	µA
		f_{osc} = 0.5 MHz, V_{CC} = 5.0 V		80	160	µA
		f_{osc} = Z MHz, V_{CC} = 5.0 V		160	320	µA/ MHz
		f_{osc} = 0.5 MHz, V_{CC} = 2.5 V		40	80	µA
I_{CC}	Halt OSC-On	f_{osc} = 6.0 MHz, V_{CC} = 5.0 V		480	980	µA
		f_{osc} = 3.0 MHz, V_{CC} = 5.0 V		240	500	µA
		f_{osc} = 0.5 MHz, V_{CC} = 5.0 V		45	100	µA
		f_{osc} = Z MHz, V_{CC} = 5.0 V			See Note 2	µA
		f_{osc} = 0.5 MHz, V_{CC} = 2.5 V		25	60	µA
I_{CC}	Halt OSC-Off	V_{CC} = 2.5 to 6 V		1	10	µA

Notes: 1) All inputs = V_{CC} or V_{SS} (except XTAL2). All I/O and output pins are open.
2) Maximum current = 160(Z) + 20 µA.

Figure 4–1. Clock Timing

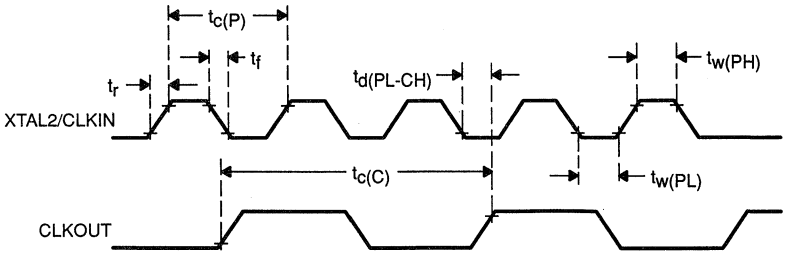


Table 4–5. Recommended Crystal/Clockin Operating Conditions over Full Operating Range

Parameter		Test Conditions	Min	Typ†	Max	Unit
f_{osc}	Crystal frequency	V_{CC} = 2.5 V	0.5		0.8	MHz
		V_{CC} = 4.0 V	0.5		4.0	MHz
		V_{CC} = 5.0 V	0.5		6.0	MHz
		V_{CC} = 6.0 V	0.5		6.5	MHz
	CLKIN duty cycle		45		55	%
$t_{c(P)}$	Crystal cycle time ‡	V_{CC} = 2.5 V	1250		2000	ns
		V_{CC} = 4.0 V	250		2000	ns
		V_{CC} = 5.0 V	166		2000	ns
		V_{CC} = 6.0 V	153		2000	ns
$t_{c(C)}$	Internal state cycle time	V_{CC} = 2.5 V	2500		4000	ns
		V_{CC} = 4.0 V	500		4000	ns
		V_{CC} = 5.0 V	333		4000	ns
		V_{CC} = 6.0 V	306		4000	ns
$t_{w(PH)}$	CLKIN pulse duration high		70			ns
$t_{w(PL)}$	CLKIN pulse duration low		70			ns
t_r	CLKIN rise time				30	ns
t_f	CLKIN fall time				30	ns
$t_{d(PL-CH)}$	CLKIN fall to CLKOUT rise delay			110	250	ns

† V_{CC} = 5 V, T_A = 25°C
‡ See Section 3.4 for Recommended Clock Connections.

Figure 4–2. Operating Frequency Range

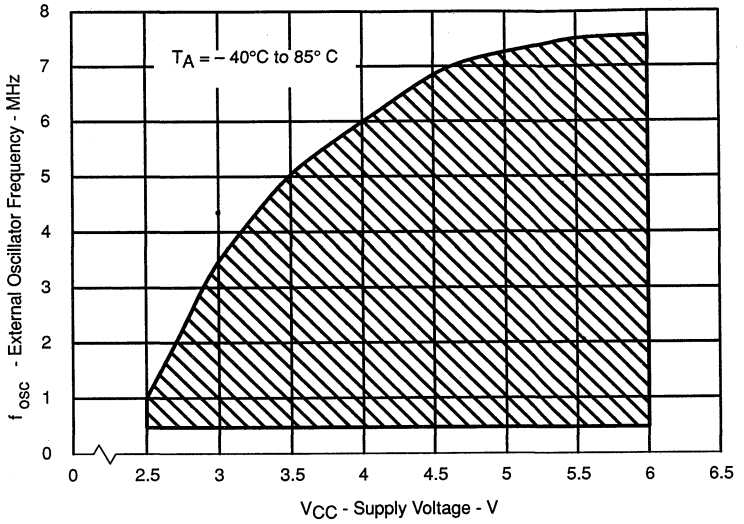


Figure 4–3. Typical Operating Current vs. Supply Voltage

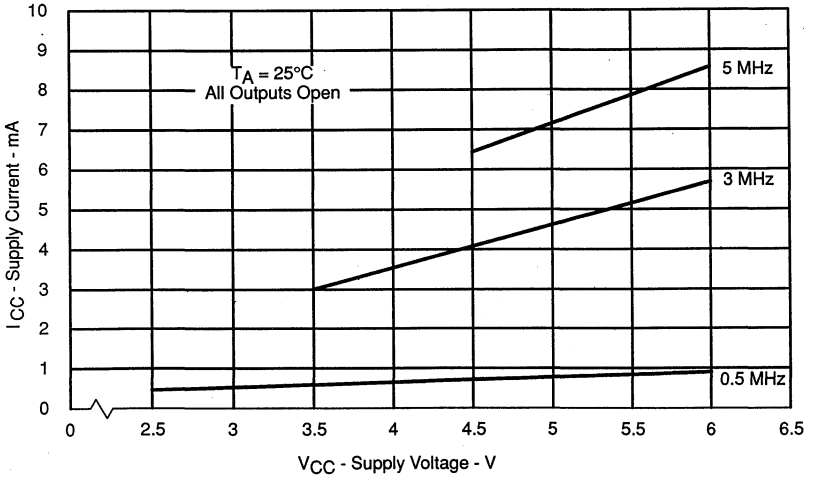


Figure 4–4. Typical Power-Down Current vs. Oscillator Frequency

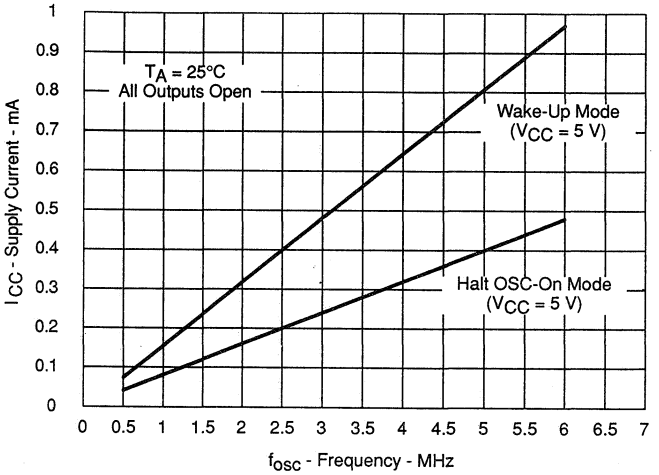


Figure 4–5. Typical Operating I_{CC} vs. Oscillator Frequency

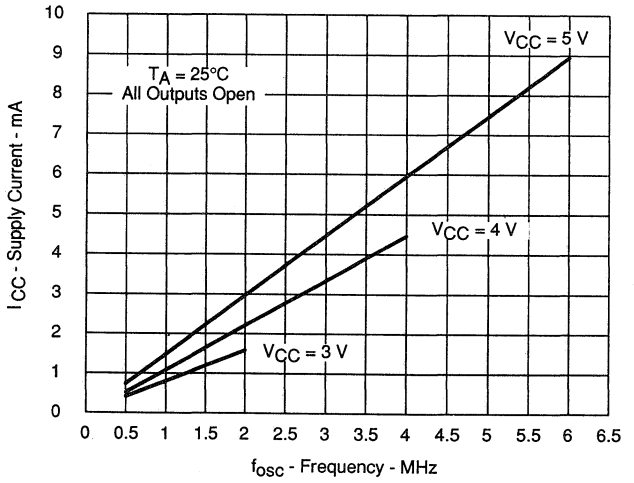


Figure 4–6. Typical Output Source Characteristics

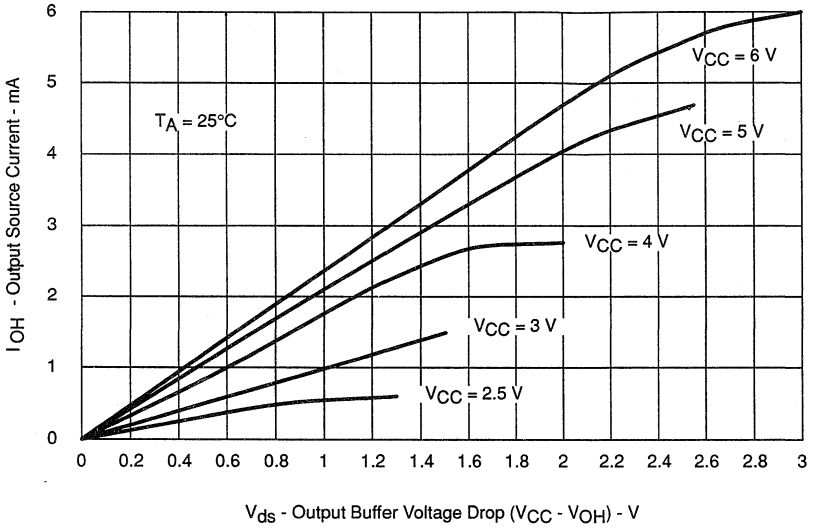
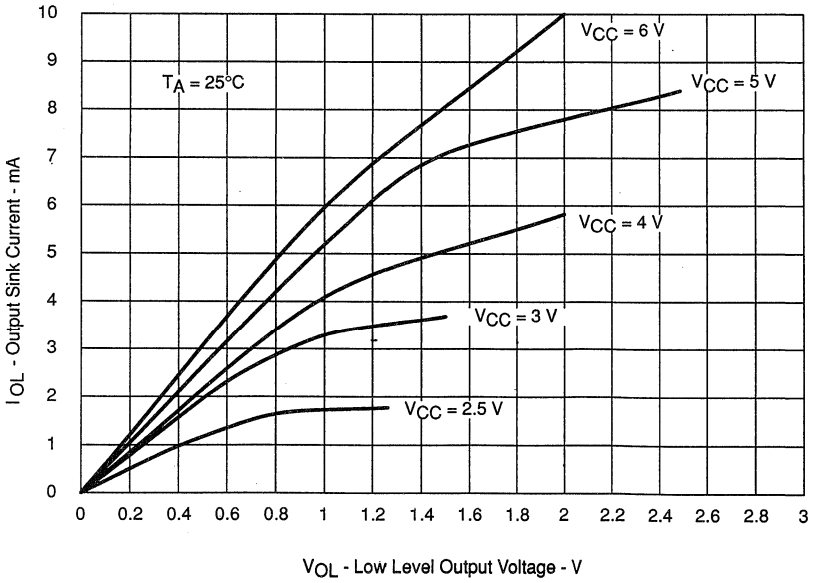


Figure 4–7. Typical Output Sink Characteristics



4.2 TMS70C00, TMS70C20, and TMS70C40 Specifications (5V ±10%)

Table 4–6. Absolute Maximum Rating over Operating Free-Air Temperature Range (Unless Otherwise Noted)

Supply voltage, V_{CC}†	– 0.3 V to 7 V
All input voltages	– 0.3 V to V_{CC} + 0.3 V
All output voltages	– 0.3 V to V_{CC} + 0.3 V
Maximum I/O buffer current (per pin)	±10 mA
Storage temperature range	– 55°C to 150°C
I_{CC}, I_{SS} current (maximum into pins 25 and 40)	±60 mA
Continuous power dissipation	0.5 W

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Stresses beyond those listed under "Absolute Maximum Ratings" 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 beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

Table 4–7. Recommended Operating Conditions

Parameter			Min	Nom	Max	Unit
V_{CC}	Supply voltage		4.5		5.5	V
INT1, INT3, RESET, and XTAL Pins						
V_{IH}	High-level input voltage	4.5 V ≤ V_{CC} ≤ 5.5 V	V_{CC} – 0.7		V_{CC}	V
MC Pin						
V_{IH}	High-level input voltage	5 V ≤ V_{CC} < 5.5 V	V_{CC} – 0.5		V_{CC}	V
		4.5 V ≤ V_{CC} < 5V	V_{CC} – 0.4		V_{CC}	V
Port (All Other Pins)						
V_{IH}	High-level input voltage	5 V ≤ V_{CC} < 5.5 V	V_{CC} – 1.3		V_{CC}	V
		4.5 V ≤ V_{CC} < 5 V	V_{CC} – 1.0		V_{CC}	V
INT1, INT3, RESET, and XTAL Pins						
V_{IL}	Low-level input voltage	4.5 V ≤ V_{CC} ≤ 5.5 V	0		0.70	V
MC Pin						
V_{IL}	Low-level input voltage	5 V ≤ V_{CC} < 5.5 V	0		0.5	V
		4.5 V ≤ V_{CC} < 5 V	0		0.4	V
Port (All Other Pins)						
V_{IL}	Low-level input voltage	5 V ≤ V_{CC} < 5.5 V	0		1.5	V
		4.5 V ≤ V_{CC} < 5 V	0		1.1	V
T_A	Operating temperature range	Commercial (TMS70Cx0N)	0		70	°C
		Industrial (TMS70Cx0NA)	– 40		85	°C

Table 4–8. Electrical Characteristics over Full Range of Operating Conditions

Parameter		Test Conditions	Min	Typ†	Max	Unit
I_I	Input leakage current	$V_{IN} = V_{SS}$ to V_{CC}		±0.1	±5	µA
C_I	Input capacitance			5		pF
V_{OH}	High-level output voltage	$I_{OH} = -0.3$ mA	$V_{CC} - 0.5$	4.7		V
V_{OL}	Low-level output voltage	$I_{OL} = 1.4$ mA		0.2	0.4	V
I_{OH}	High-level output source current	$V_{OH} = V_{CC} - 0.5$ V	–0.3	–1.2		mA
		$V_{OH} = 2.5$ V min	–1.0	–3.0		mA
I_{OL}	Output sink current	$V_{OL} = 0.4$ V	1.4	2.0		mA

† $V_{CC} = 5$ V, $T_A = 25°C$

Figure 4–8. Output Loading Circuit for Test

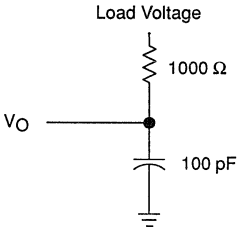


Figure 4–9. Measurement Points for Switching Characteristics

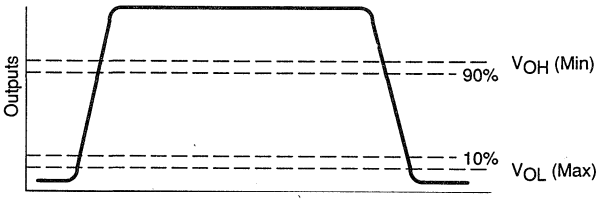


Table 4–9. AC Characteristics for I/O Ports

Parameter		Test Conditions	Min	Typ	Max	Unit
t_r	I/O port output rise time	$C_{load} = 15$ pF, $V_{CC} = 5$ V		35	60	ns
t_f	I/O port output fall time	$C_{load} = 15$ pF, $V_{CC} = 5$ V		20	50	ns

Note: Rise and fall times are measured between the maximum low level and the minimum high level using the 10% and 90% points.

Table 4–10. Supply Current Requirements

Parameter	Test Conditions	Min	Typ	Max	Unit
I_{CC} Operating mode	f_{osc} = 5.0 MHz		7.5	13.5	mA
	f_{osc} = 3.0 MHz		4.5	8.1	mA
	f_{osc} = 1.0 MHz		1.5	2.7	mA
	f_{osc} = Z MHz		1.5	2.7	mA/ MHz
I_{CC} Wake-up mode (timer active)	f_{osc} = 5.0 MHz		800	1750	µA
	f_{osc} = 3.0 MHz		480	1050	µA
	f_{osc} = 1.0 MHz		160	350	µA
	f_{osc} = Z MHz		160	350	µA/ MHz
I_{CC} Halt OSC-On	f_{osc} = 5.0 MHz		480	920	µA
	f_{osc} = 3.0 MHz		240	560	µA
	f_{osc} = 1.0 MHz		80	200	µA
	f_{osc} = Z MHz		See Note 2		µA
I_{CC} Halt OSC-Off			1	10	µA

Notes: 1) All inputs = V_{CC} or V_{SS} (except XTAL2). All I/O and output pins are open.
2) Maximum current = 180(Z) + 20 µA.

Table 4–11. Recommended Crystal/Clockin Operating Conditions over Full Operating Range

Parameter		Min	Typ†	Max	Unit
f_{osc}	Crystal frequency	0.5		5.0	MHz
	CLKIN duty cycle	45		55	%
$t_{c(P)}$	Crystal cycle time ‡	200		2000	ns
$t_{c(C)}$	Internal state cycle time	400		4000	ns
$t_{w(PH)}$	CLKIN pulse duration high	90			ns
$t_{w(PL)}$	CLKIN pulse duration low	90			ns
t_r	CLKIN rise time			30	ns
t_f	CLKIN fall time			30	ns
$t_{d(PL-CH)}$	CLKIN fall to CLKOUT rise delay		140	250	ns

† V_{CC} = 5 V, T_A = 25°C.
‡ See Section 3.4 for Recommended Clock Connections.

Figure 4–10. Clock Timing

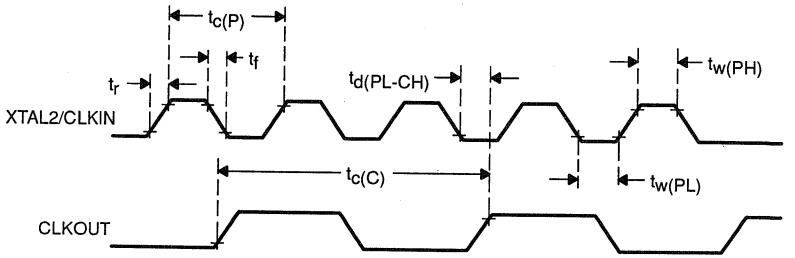


Table 4–12. Memory Interface Timings †

Parameter		Min	Typ	Max	Unit
$t_{c(C)}$	Clock cycle time		t_c		ns
$t_{w(CH)}$	Clockout high	$0.5t_c-100$	$0.5t_c$	$0.5t_c+100$	ns
$t_{w(CL)}$	Clockout low	$0.5t_c-100$	$0.5t_c$	$0.5t_c+100$	ns
$t_{c(R)}$	Read cycle time		$2\,t_c$		ns
$t_{c(W)}$	Write cycle time		$2\,t_c$		ns
$t_{d(CH-JL)}$	Clock high to ALATCH low	$0.5t_c-50$	$0.5t_c$		ns
$t_{d(CH-EL)}$	Clock high to $\overline{\text{ENABLE}}$ low	–10	30		ns
$t_{a(EL-D)}$	$\overline{\text{ENABLE}}$ active to data in	$0.75t_c-160$		$0.75t_c$	ns
$t_{h(EH-D)}$	Data in hold after $\overline{\text{ENABLE}}$ inactive	0			ns
$t_{a(A-D)}$	Data in from address valid	$1.5t_c-200$		$1.5t_c$	ns
$t_{w(JH)}$	ALATCH active	$0.25t_c-50$		$0.25t_c$	ns
$t_{su(AH-JL)}$	High address to ALATCH fall	$0.25t_c-50$		$0.25t_c$	ns
$t_{su(AL-JL)}$	Low address to ALATCH fall	$0.25t_c-55$		$0.25t_c$	ns
$t_{h(JL-AL)R}$	Low address hold from ALATCH fall (Read)	$0.5t_c-150$		$0.5t_c$	ns
$t_{d(LA-EL)}$	Low address high–Z to $\overline{\text{ENABLE}}$ active	0			ns
$t_{h(EH-RW)}$	$\overline{\text{ENABLE}}$ inactive to R/$\overline{\text{W}}$	$0.5t_c-100$		$0.5t_c$	ns
$t_{d(JL-EL)}$	ALATCH fall to $\overline{\text{ENABLE}}$ active	$0.5t_c-80$		$0.5t_c$	ns
$t_{d(EH-AL)}$	$\overline{\text{ENABLE}}$ inactive to low address drive	$0.5t_c-100$		$0.5t_c$	ns
$t_{h(EH-AH)}$	High address hold from $\overline{\text{ENABLE}}$ inactive	$0.5t_c-100$		$0.5t_c$	ns
$t_{su(Q-EH)}$	Data out to $\overline{\text{ENABLE}}$ inactive	$0.5t_c-50$		$0.5t_c$	ns
$t_{h(EH-Q)}$	Data out hold from $\overline{\text{ENABLE}}$ inactive	$0.5t_c-60$		$0.5t_c$	ns
$t_{d(A-Q)}$	Access time from address	t_c-100		t_c	ns
$t_{d(A-EH)}$	Low address to $\overline{\text{ENABLE}}$ high	$1.5t_c-100$		$1.5t_c$	ns
$t_{d(EH-A)}$	$\overline{\text{ENABLE}}$ rise to next address	$0.5t_c-60$		$0.5t_c$	ns
$t_{su(RW-JL)}$	R/$\overline{\text{W}}$ to ALATCH fall	$0.25t_c-60$		$0.25t_c$	ns
$t_{h(JL-D)}$	After ALATCH fall to data in	$1.25t_c-200$		$1.25t_c$	ns
$t_{d(EH-JH)}$	$\overline{\text{ENABLE}}$ rise to ALATCH rise	$0.5t_c-60$		$0.5t_c$	ns
$t_{w(E)}$	$\overline{\text{ENABLE}}$ pulse width	$0.75t_c-80$		$0.75t_c$	ns

† V_{CC} = 5 V ±10%, t_c = 2/freq

Figure 4–11. Output Loading Circuit for Test

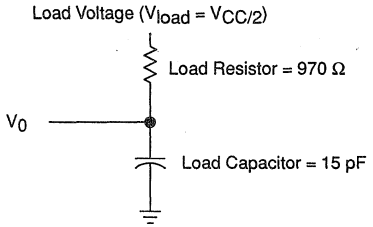
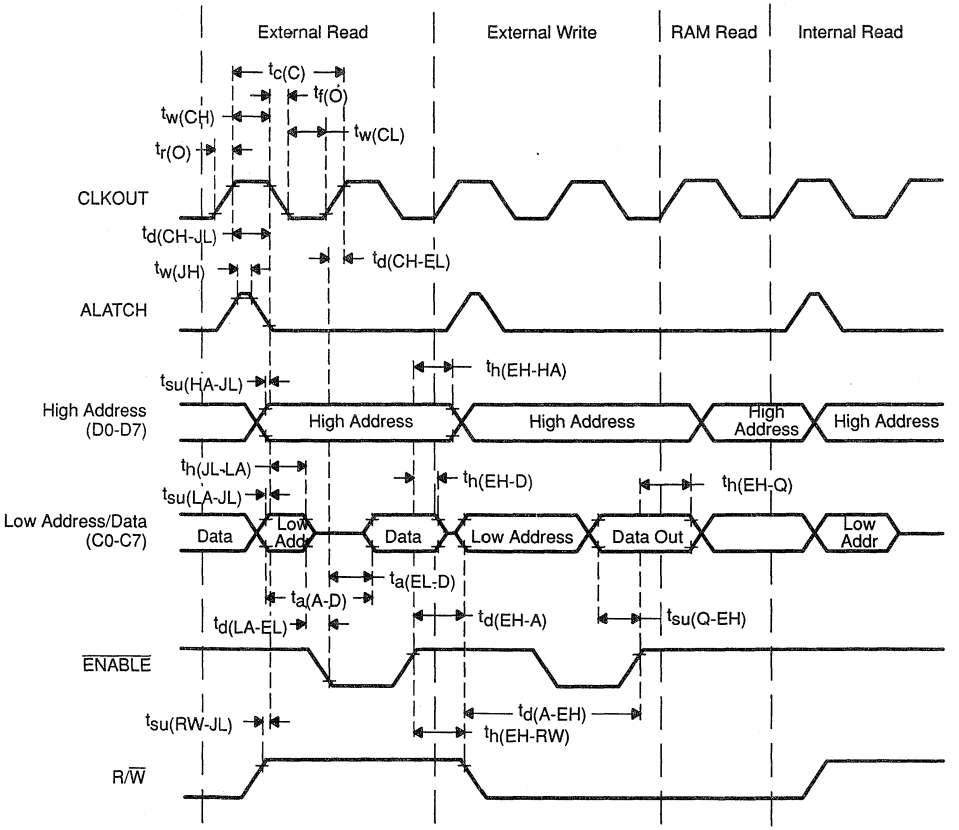


Figure 4–12. Read and Write Cycle Timing



4.3 TMS70CT20 and TMS70CT40 Specifications (5 V ± 10%)

Table 4–13. Absolute Maximum Ratings over Operating Free-Air Temperature Range (Unless Otherwise Noted)

Supply voltage range, V_{CC}† . . . . . – 0.3 V to 7 V
Input voltage range . . . . . – 0.3 V to V_{CC}+0.3 V
Output voltage range . . . . . – 0.3 V to V_{CC}+0.3 V
Maximum I/O buffer current (per pin) . . . . . ±10 mA
Storage temperature range . . . . . – 55°C to 150°C
I_{CC}, I_{SS} (maximum into pins 17 and 1) . . . . . ±60 mA
Continuous power dissipation . . . . . 0.5 W

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Stresses beyond those listed under "Absolute Maximum Ratings" 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 beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute maximum rated conditions for extended periods may affect device reliability.

Table 4–14. Recommended Operating Conditions

Parameter			Min	Nom	Max	Unit
V_{CC}	Supply voltage		4.5		5.5	V
INT1, INT3, RESET, and XTAL Pins						
V_{IH}	High-level input voltage	4.5 V ≤ V_{CC} ≤ 5.5 V	V_{CC} – 0.7		V_{CC}	V
MC Pin						
V_{IH}	High-level input voltage	5 V ≤ V_{CC} < 5.5 V	V_{CC} – 0.5		V_{CC}	V
		4.5 V ≤ V_{CC} < 5 V	V_{CC} – 0.4		V_{CC}	V
Port (All Other Pins)						
V_{IH}	High-level input voltage	5 V ≤ V_{CC} < 5.5 V	V_{CC} – 1.3		V_{CC}	V
		4.5 V ≤ V_{CC} < 5 V	V_{CC} – 1.0		V_{CC}	V
INT1, INT3, RESET, and XTAL Pins						
V_{IL}	Low-level input voltage	4.5 V ≤ V_{CC} ≤ 5.5 V	0		0.70	V
MC Pin						
V_{IL}	Low-level input voltage	5 V ≤ V_{CC} < 5.5 V	0		0.5	V
		4.5 V ≤ V_{CC} < 5 V	0		0.4	V
Port (All Other Pins)						
V_{IL}	Low-level input voltage	5 V ≤ V_{CC} < 5.5 V	0		1.5	V
		4.5 V ≤ V_{CC} < 5 V	0		1.1	V
T_A	Operating temperature range		0		70	°C

Table 4–15. Electrical Characteristics over Full Range of Operating Conditions

Parameter		Test Conditions	Min	Typ†	Max	Unit
I_I	Input leakage current	$V_{IN} = V_{SS}$ to V_{CC}		±1	±5	µA
C_I	Input capacitance			5		pF
V_{OH}	High-level output voltage	$I_{OH} = -0.3$ mA	$V_{CC} - 0.54.7$			V
V_{OL}	Low-level output voltage	$I_{OL} = 1.4$ mA		0.2	0.4	V
I_{OH}	High-level output source current	$V_{OH} = V_{CC} - 0.5$ V	– 0.3	– 1.2		mA
		$V_{OH} = 2.5$ V	– 1.0	– 3.0		mA
I_{OL}	Output sink current	$V_{OL} = 0.4$ V	1.4	2.0		mA

† $V_{CC} = 5$ V, $T_A = 25°C$

Figure 4–13. Output Loading Circuit for Test

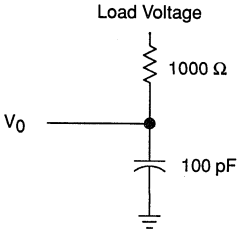


Figure 4–14. Measurement Points for Switching Characteristics

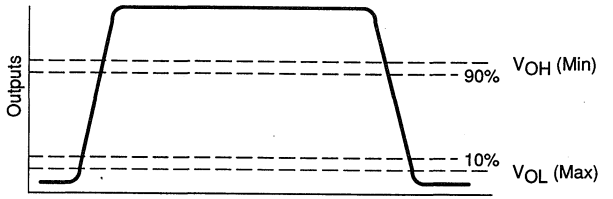


Table 4–16. AC Characteristics for I/O Port

Parameter		Test Conditions	Min	Typ	Max	Unit
t_r	I/O port output rise time	$C_{load} = 15$ pF, $V_{CC} = 5$ V		35	60	ns
t_f	I/O port output fall time	$C_{load} = 15$ pF, $V_{CC} = 5$ V		20	50	ns

Note: Rise and fall times are measured between the maximum low level and the minimum high level using the 10% and 90% points.

Table 4–17. Supply Current Requirements

Parameter	Test Conditions	Min	Typ	Max	Unit
I_{CC} Operating mode	f_{osc} = 5.0 MHz		7.5	13.5	mA
	f_{osc} = 3.0 MHz		4.5	8.1	mA
	f_{osc} = 1.0 MHz		1.5	2.7	mA
	f_{osc} = Z MHz		1.5	2.7	mA/ MHz
I_{CC} Wake-up mode (timer active)	f_{osc} = 5.0 MHz		800	1750	µA
	f_{osc} = 3.0 MHz		480	1050	µA
	f_{osc} = 1.0 MHz		160	350	µA
	f_{osc} = Z MHz		160	350	µA/ MHz
I_{CC} Halt OSC-On	f_{osc} = 5.0 MHz		480	920	µA
	f_{osc} = 3.0 MHz		240	560	µA
	f_{osc} = 1.0 MHz		80	200	µA
	f_{osc} = Z MHz	See Note 2			µA
I_{CC} Halt OSC-Off			1	10	µA

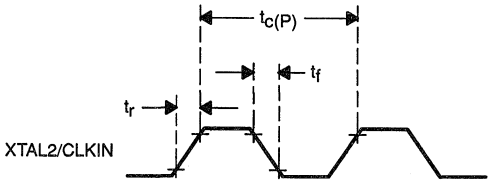
Notes: 1) All inputs = V_{CC} or V_{SS} (except XTAL2). All I/O and output pins are open.
2) Maximum current = 180(Z) + 20 µA.

Table 4–18. Recommended Crystal/Clockin Operating Conditions over Full Operating Range

Parameter		Min	Typ†	Max	Unit
f_{osc}	Crystal frequency	0.5		5.0	MHz
	CLKIN duty cycle	45		55	%
$t_{c(P)}$	Crystal cycle time ‡	200		2000	ns
$t_{c(C)}$	Internal state cycle time	400		4000	ns
$t_{w(PH)}$	CLKIN pulse duration high	90			ns
$t_{w(PL)}$	CLKIN pulse duration low	90			ns
t_r	CLKIN rise time			30	ns
t_f	CLKIN fall time			30	ns

† V_{CC} = 5 V, T_A = 25°C
‡ See Section 3.4 for Recommended Clock Connections.

Figure 4–15. Clock Timing



4.4 TMS70C02, TMS70C42, and TMS70C82 Specifications (Wide Voltage)

Table 4–19. Absolute Maximum Ratings over Operating Free-Air Temperature Range (Unless Otherwise Noted)

Supply voltage range, V_{CC}†	– 0.3V to 7 V
Input voltage range	– 0.3 V to V_{CC}+0.3 V
Output voltage range	– 0.3V to V_{CC}+0.3 V
Maximum I/O buffer current (per pin)	±10 mA
Storage temperature range	– 55°C to 150°C
I_{CC}, I_{SS} (maximum into pin 25 or 40)	±60 mA
Continuous power dissipation	0.5 W

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Stresses beyond those listed under "Absolute Maximum Ratings" 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 beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute maximum rated conditions for extended periods may affect device reliability.

Table 4–20. Recommended Operating Conditions

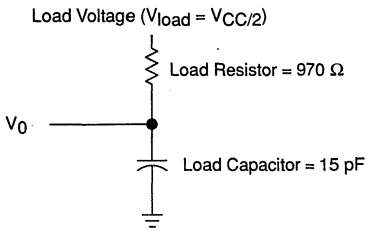
			Min	Nom	Max	Unit
V_{CC}	Supply voltage		2.5		6.0	V
INT1, INT3, RESET, and XTAL Pins						
V_{IH}	High-level input voltage	5.5 V ≤ V_{CC} ≤ 6 V	V_{CC} – 1.0		V_{CC}	V
		4.5 V ≤ V_{CC} ≤ 5.5 V	V_{CC} – 0.7		V_{CC}	V
		3.5 V ≤ V_{CC} ≤ 4.5 V	V_{CC} – 0.5		V_{CC}	V
		2.5 V ≤ V_{CC} ≤ 3.5 V	V_{CC} – 0.35		V_{CC}	V
MC Pin						
V_{IH}	High-level input voltage	5 V ≤ V_{CC} ≤ 6 V	V_{CC} – 0.5		V_{CC}	V
		4 V ≤ V_{CC} < 5 V	V_{CC} – 0.4		V_{CC}	V
		3 V ≤ V_{CC} < 4 V	V_{CC} – 0.3		V_{CC}	V
		2.5 V ≤ V_{CC} < 3 V	V_{CC} – 0.2		V_{CC}	V
Port (All Other Pins)						
V_{IH}	High-level input voltage	5 V ≤ V_{CC} ≤ 6 V	V_{CC} – 1.3		V_{CC}	V
		4 V ≤ V_{CC} < 5 V	V_{CC} – 1.0		V_{CC}	V
		3 V ≤ V_{CC} < 4 V	V_{CC} – 0.7		V_{CC}	V
		2.5 V ≤ V_{CC} < 3 V	V_{CC} – 0.4		V_{CC}	V
INT1, INT3, RESET, and XTAL Pins						
V_{IL}	Low-level input voltage	5.5 V ≤ V_{CC} ≤ 6 V	0		1.00	V
		4.5 V ≤ V_{CC} ≤ 5.5 V	0		0.70	V
		3.5 V ≤ V_{CC} ≤ 4.5 V	0		0.50	V
		2.5 V ≤ V_{CC} ≤ 3.5 V	0		0.35	V
MC Pin						
V_{IL}	Low-level input voltage	5 V ≤ V_{CC} ≤ 6 V	0		0.5	V
		4 V ≤ V_{CC} < 5 V	0		0.4	V
		3 V ≤ V_{CC} < 4 V	0		0.3	V
		2.5 V ≤ V_{CC} < 3 V	0		0.2	V
Port (All Other Pins)						
V_{IL}	Low-level input voltage	5 V ≤ V_{CC} ≤ 6 V	0		1.5	V
		4 V ≤ V_{CC} < 5 V	0		1.1	V
		3 V ≤ V_{CC} < 4 V	0		0.7	V
		2.5 V ≤ V_{CC} < 3 V	0		0.3	V
T_A	Operating free-air temperature	Commercial (TMS70Cx2N)	0		70	°C
		Industrial (TMS70Cx2NA)	– 40		85	°C

Table 4–21. Electrical Characteristics over Full Range of Operating Conditions

Parameter		Test Conditions	Min	Typ†	Max	Unit
I_I	Input current	MC pin, $V_{IN} = V_{SS}$ or V_{CC} All others, $V_{IN} = V_{SS}$ to V_{CC}		±0.10	±5	µA
C_I	Input capacitance			5		pF
V_{OH}	High-level output voltage	V_{CC} = 5.0 V, I_{OH} = – 1 mA	2.5	4.5		V
		V_{CC} = 5.0 V, I_{OH} = – 0.3 mA	4.5	4.8		V
V_{OL}	Low-level output voltage	V_{CC} = 5.0 V, I_{OL} = 1.7 mA		0.3	0.4	V
I_{OH}	Output source current	V_{CC} = 5.0 V, V_{OH} = 4.5 V	– 0.3	– 1.4		µA
		V_{CC} = 4.0 V, V_{OH} = 3.5 V	– 0.2	– 1.0		mA
		V_{CC} = 3.0 V, V_{OH} = 2.5 V	– 0.1	– 0.7		mA
		V_{CC} = 2.5 V, V_{OH} = 2.0 V	– 0.05	– 0.3		mA
		V_{CC} = 5.0 V, V_{OH} = 2.5 V	– 1.0	– 5.0		mA
I_{OL}	Output sink current	V_{CC} = 5.0 V, V_{OL} = 0.4 V	1.7	2.8		mA
		V_{CC} = 4.0 V, V_{OL} = 0.4 V	1.2	2.4		mA
		V_{CC} = 3.0 V, V_{OL} = 0.4 V	0.7	2.0		mA
		V_{CC} = 2.5 V, V_{OL} = 0.4 V	0.2	1.3		mA

† V_{CC} = 5 V, T_A = 25°C
‡ Output levels ensure 400 mV of noise margin over specified input levels.

Figure 4–16. Output Loading Circuit for Test



Note: Rise and fall times are measured between the maximum low level and the minimum high level using the 10% and 90% points.

Table 4–22. Supply Current Requirements

Parameter		Test Conditions		Min	Typ	Max	Unit
I_{CC}	Operating mode	f_{osc} = 7.0 MHz,	V_{CC} = 5.0 V		17	24.5	mA
		f_{osc} = 3.0 MHz,	V_{CC} = 5.0 V		7.2	10.5	mA
		f_{osc} = 0.5 MHz,	V_{CC} = 5.0 V		1.2	1.8	mA
		f_{osc} = Z MHz,	V_{CC} = 5.0 V		2.4	3.5	mA/ MHz
		f_{osc} = 0.5 MHz,	V_{CC} = 2.5 V		0.4	1.2	mA
		f_{osc} = 8 MHz,	V_{CC} = 5.0 V (see Note 3)		20.0	36.0	mA
I_{CC}	Wake-up mode 1 (one timer and UART active)	f_{osc} = 7.0 MHz,	V_{CC} = 5.0 V		2400	5600	µA
		f_{osc} = 3.0 MHz,	V_{CC} = 5.0 V		1200	3300	µA
		f_{osc} = 0.5 MHz,	V_{CC} = 5.0 V		250	800	µA
		f_{osc} = 8 MHz,	V_{CC} = 5.0 V (see Note 3)		2600	5800	µA
I_{CC}	Wake-up mode 2 (one timer active and UART inactive)	f_{osc} = 7.0 MHz,	V_{CC} = 5.0 V		960	3400	µA
		f_{osc} = 3.0 MHz,	V_{CC} = 5.0 V		480	2000	µA
		f_{osc} = 0.5 MHz,	V_{CC} = 5.0 V		140	550	µA
		f_{osc} = 8 MHz,	V_{CC} = 5.0 V (see Note 3)		1100	3700	µA
I_{CC}	Wake-up mode 3 (UART active only)	f_{osc} = 7.0 MHz,	V_{CC} = 5.0 V		1500	2400	µA
		f_{osc} = 3.0 MHz,	V_{CC} = 5.0 V		800	1500	µA
		f_{osc} = 0.5 MHz,	V_{CC} = 5.0 V		180	600	µA
		f_{osc} = 8 MHz,	V_{CC} = 5.0 V (see Note 3)		1600	3000	µA
I_{CC}	Halt OSC-On	f_{osc} = 7.0 MHz,	V_{CC} = 5.0 V		560	1280	µA
		f_{osc} = 3.0 MHz,	V_{CC} = 5.0 V		240	560	µA
		f_{osc} = 1.0 MHz,	V_{CC} = 5.0 V		80	200	µA
		f_{osc} = Z MHz				See Note 2	µA
		f_{osc} = 8 MHz,	V_{CC} = 5.0 V (see Note 3)		800	1460	µA
I_{CC}	Halt OSC-Off				5	10	µA

Notes: 1) All inputs = V_{CC} or V_{SS} (except XTAL2). All I/O and output pins are open.
2) Maximum current = 180(Z) + 20 µA.
3) 8 MHz capability is defined at test level. Such devices should be ordered under SPEC2 special request.

Table 4–23. Recommended Crystal/Clockin Operating Conditions over Full Operating Range

Parameter		Test Conditions	Min	Typ†	Max	Unit
f_{osc}	Crystal frequency	V_{CC} = 2.5 V	0.5		3.5	MHz
		V_{CC} = 4.0 V	0.5			MHz
		V_{CC} = 5.0 V	0.5		0	MHz
		V_{CC} = 6.0 V	0.5		7.5	MHz
	for SPEC2 option	V_{CC} = 5.0 V ± 5%	3.0		8.0	MHz
	CLKIN duty cycle		47		53	%
$t_{c(P)}$	CLKIN cycle time	V_{CC} = 2.5 V	333		2000	ns
		V_{CC} = 4.0 V	167		2000	ns
		V_{CC} = 5.0 V	143		2000	ns
		V_{CC} = 6.0 V	133		2000	ns
	for SPEC2 option	V_{CC} = 5.0 V ± 5%	125		2000	ns
$t_{c(C)}$	Internal state cycle time	V_{CC} = 2.5 V	666		4000	ns
		V_{CC} = 4.0 V	333		4000	ns
		V_{CC} = 5.0 V	286		4000	ns
		V_{CC} = 6.0 V	267		4000	ns
	for SPEC2 option	V_{CC} = 5.0 V ± 5%	250		4000	ns
$t_{w(PH)}$	CLKIN pulse duration high		50			ns
$t_{w(PL)}$	CLKIN pulse duration low		50			ns
t_r	CLKIN rise time				20	ns
t_f	CLKIN fall time				10	ns
$t_{d(PL-CH)}$	CLKIN fall to CLKOUT rise			110	250	ns

Figure 4–17. Clock Timing

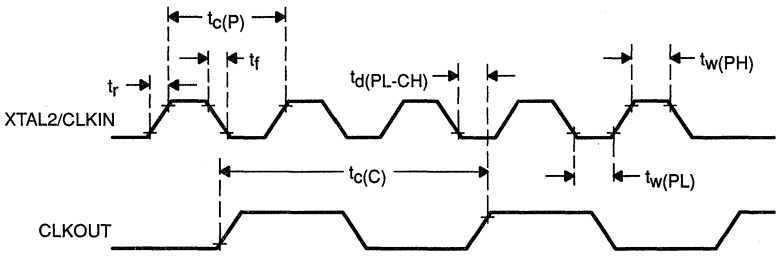


Figure 4–18. Operating Frequency Range

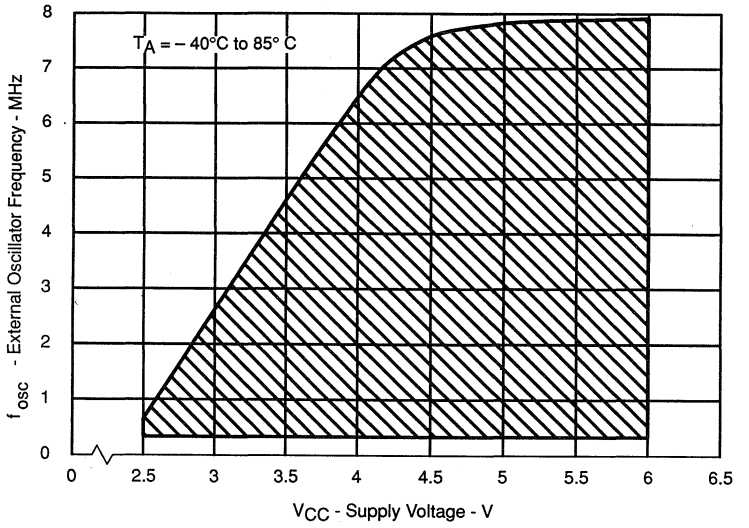


Figure 4–19. Typical Operating Current vs. Supply Voltage

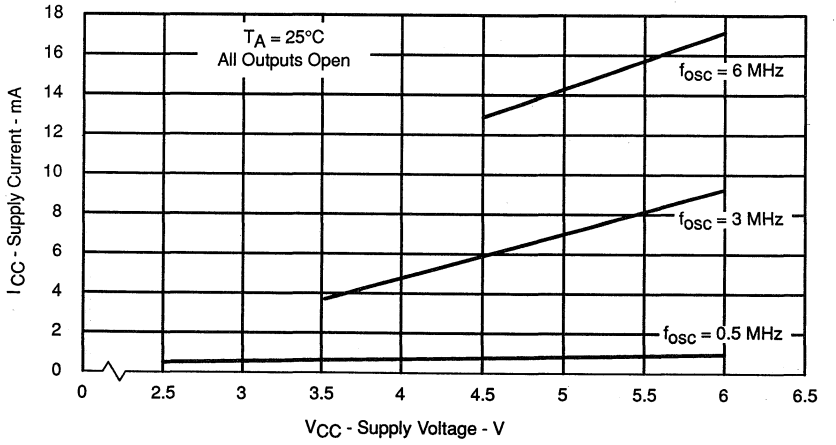


Figure 4–20. Typical Operating I_{CC} vs. Oscillator Frequency

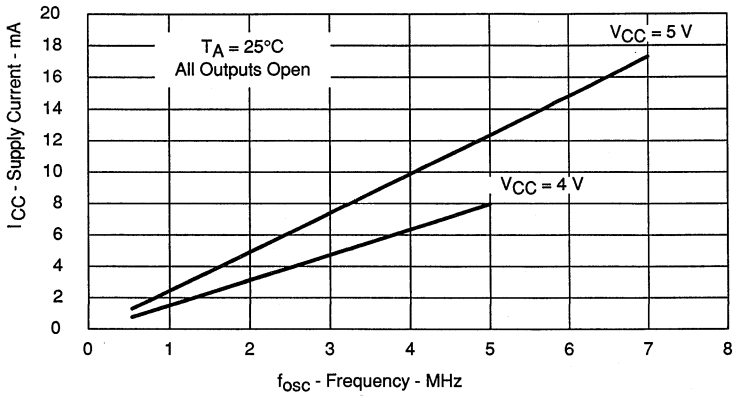


Figure 4–21. Typical Operating Current vs. Supply Voltage

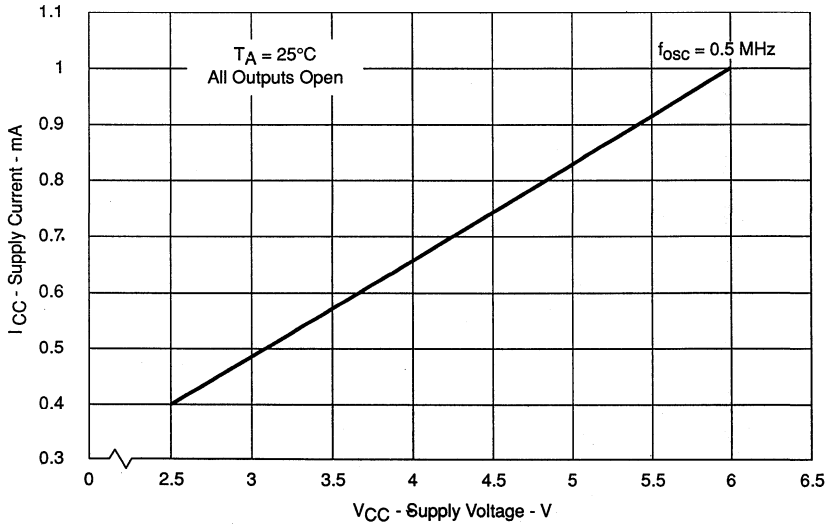


Figure 4–22. Typical Output Source Characteristics

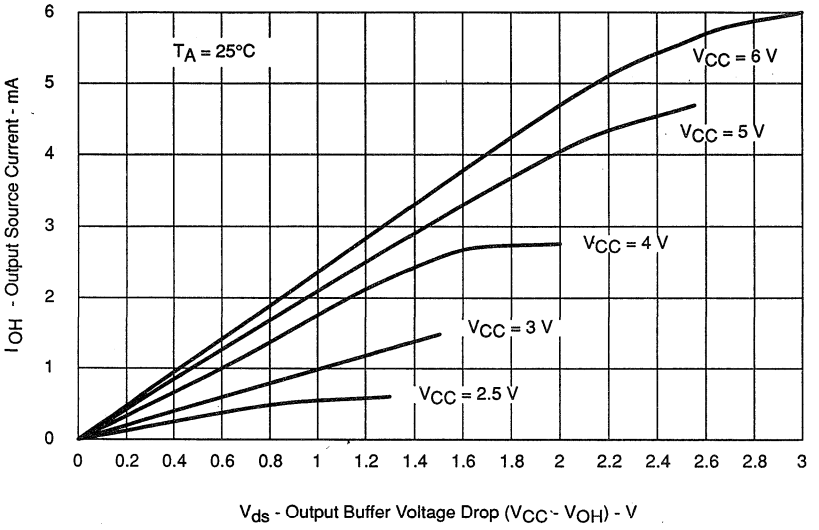
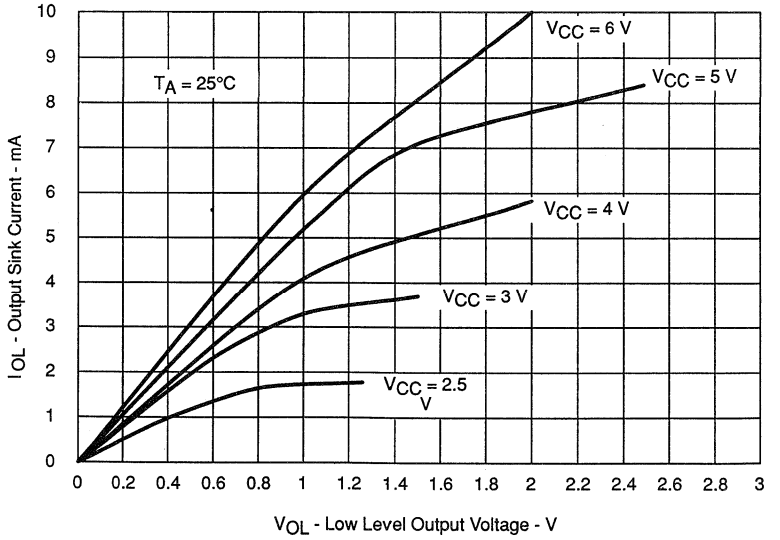
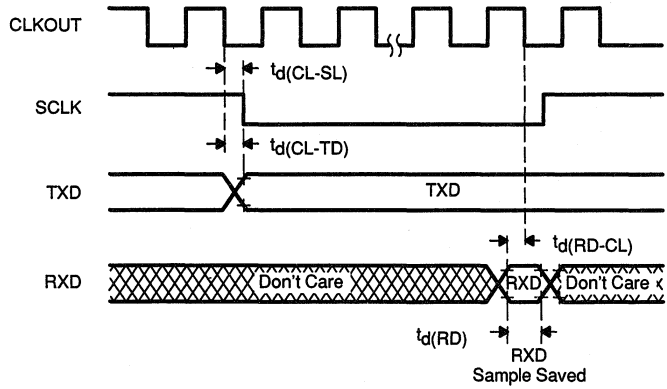


Figure 4–23. Typical Output Sink Characteristics



4.4.1 Serial Port Timing

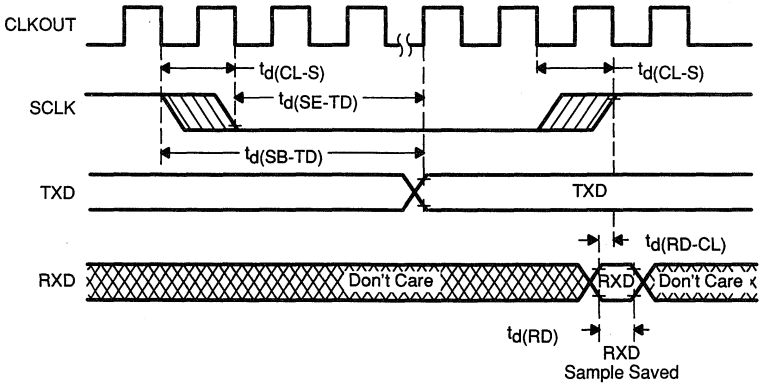
4.4.1.1 Internal Serial Clock



Notes: 1) The CLKOUT signal is not available in single-chip mode.
2) CLKOUT = $t_{c(C)}$.

Parameter		Typ	Unit
$t_{d(CL-SL)}$	CLKOUT low to SCLK low	1/4 $t_{c(C)}$	ns
$t_{d(CL-TD)}$	CLKOUT low to new TXD data	1/4 $t_{c(C)}$	ns
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns
$t_{c(S)}$	SCLK cycle time	4 $t_{c(C)}$	ns
$t_{w(SH)}$	SCLK out high	2 $t_{c(C)}$	ns
$t_{w(SL)}$	SCLK out low	2 $t_{c(C)}$	ns

4.4.1.2 External Serial Clock



Notes: 1) The CLKOUT signal is not available in single-chip mode.
2) CLKOUT = $t_{c(C)}$.
3) SCLK sampled; if SCLK = 1 then 0, fall transition found.
4) SCLK sampled; if SCLK = 0 then 1, rise transition found.

Parameter		Typ	Unit
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns
$t_{d(SB-TD)}$	Start of SCLK sample to new TXD data	3 1/4 $t_{c(C)}$	ns
$t_{d(SE-TD)}$	End of SCLK sample to new TXD data	2 1/4 $t_{c(C)}$	ns
$t_{d(CL-S)}$	Clockout low to SCLK transition	$t_{c(C)}$	ns
$t_{c(S)}$	SCLK cycle time	6 $t_{c(C)}$	ns
$t_{w(SH)}$	SCLK high	2 $t_{c(C)}$	ns
$t_{w(SL)}$	SCLK low	4 $t_{c(C)}$	ns

4.5 TMS70C02, TMS70C42, and TMS70C82 Specifications (5V ±10%)

Table 4–24. Absolute Maximum Ratings over Operating Free-Air Temperature Range (Unless Otherwise Noted)

Supply voltage range, V_{CC}†	– 0.3 V to 7 V
Input voltage range	– 0.3 V to V_{CC}+0.3 V
Output voltage range	– 0.3 V to V_{CC}+0.3 V
Maximum I/O buffer current (per pin)	±10 mA
Storage temperature range	– 55°C to 150°C
I_{CC}, I_{SS} (maximum into pin 25 or 40	±60 mA
Continuous power dissipation	0.5 W

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Stresses beyond those listed under "Absolute Maximum Ratings" 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 beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute maximum rated conditions for extended periods may affect device reliability.

Table 4–25. Recommended Operating Conditions

			Min	Nom	Max	Unit
V_{CC}	Supply voltage		4.5		5.5	V
INT1, INT3, RESET, and XTAL Pins						
V_{IH}	High-level input voltage	4.5 V ≤ V_{CC} ≤ 5.5 V	V_{CC} – 0.7		V_{CC}	V
MC Pin						
V_{IH}	High-level input voltage	5 V ≤ V_{CC} < 5.5 V	V_{CC} – 0.5		V_{CC}	V
		4.5 V ≤ V_{CC} < 5 V	V_{CC} – 0.4		V_{CC}	V
Port (All Other Pins)						
V_{IH}	High-level input voltage	5 V ≤ V_{CC} < 5.5 V	V_{CC} – 1.3		V_{CC}	V
		4.5 V ≤ V_{CC} < 5 V	V_{CC} – 1.0		V_{CC}	V
INT1, INT3, RESET, and XTAL Pins						
V_{IL}	Low-level input voltage	4.5 V ≤ V_{CC} ≤ 5.5 V	0		0.70	V
MC Pin						
V_{IL}	Low-level input voltage	5 V ≤ V_{CC} < 5.5 V	0		0.5	V
		4.5 V ≤ V_{CC} < 5 V	0		0.4	V
Port (All Other Pins)						
V_{IL}	Low-level input voltage	5 V ≤ V_{CC} < 5.5 V	0		1.5	V
		4.5 V ≤ V_{CC} < 5 V	0		1.1	V
T_A	Operating temperature	Commercial (TMS70Cx2N)	0		70	°C
		Industrial (TMS70Cx2NA)	– 40		85	°C

Table 4–26. Electrical Characteristics over Full Range of Operating Conditions

Parameter		Test Conditions	Min	Typ†	Max	Unit
I_I	Input leakage current	MC pin, $V_{IN} = V_{SS}$ or V_{CC} All others, $V_{IN} = V_{SS}$ to V_{CC}		±0.1	±5	µA
C_I	Input capacitance			5		pF
V_{OH}	High-level output voltage	V_{CC} = 5.0 V, I_{OH} = – 1 mA	2.5	4.5		V
		V_{CC} = 5.0 V, I_{OH} = – 0.3 mA	4.5	4.8		V
V_{OL}	Low-level output voltage	V_{CC} = 5.0 V, I_{OL} = 1.7 mA		0.3	0.4	V
I_{OH}	High-level output source current	$V_{OH} = V_{CC}$ – 0.5 V	– 0.3	– 1.2		mA
		V_{OH} = 2.5 V min	– 1.0	– 3.0		mA
I_{OL}	Output sink current	V_{OL} = 0.4 V	1.5	2.6		mA

Table 4–27. AC Characteristics for Input/Output Ports†

Parameter		Test Conditions	Min	Typ†	Max	Unit
$t_{r(IO)}$	I/O port output rise time	C_{load} = 15 pF, V_{CC} = 5 V		35	60	ns
$t_{f(IO)}$	I/O port output fall time	C_{load} = 15 pF, V_{CC} = 5 V		20	50	ns

† Rise and fall times are measured between the maximum low level and the minimum high level using the 10% and 90% points.

Figure 4–24. Output Loading Circuit for Test

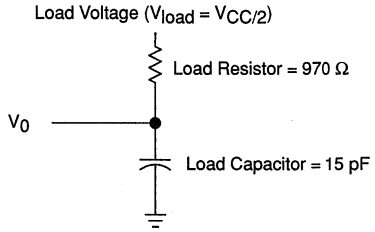


Figure 4–25. Measurement Points for Switching Characteristics

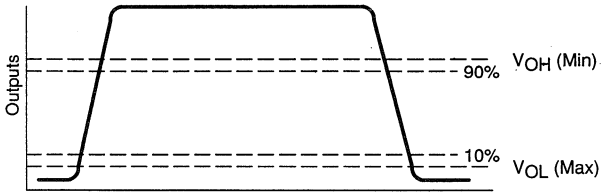


Table 4–28. Supply Current Requirements

Parameter		Test Conditions	Min	Typ	Max	Unit
I_{CC}	Supply current	f_{osc} = 6.0 MHz		15	24	mA
		f_{osc} = 3.0 MHz		7.2	12	mA
		f_{osc} = 1.0 MHz		2.4	4.0	mA
		f_{osc} = Z MHz		2.4	4.0	mA/ MHz
I_{CC}	Wake-up mode 1 (one timer and UART active)	f_{osc} = 6.0 MHz		2400	5400	µA
		f_{osc} = 3.0 MHz		1200	2900	µA
		f_{osc} = 1.0 MHz		650	1500	µA
I_{CC}	Wake-up mode 2 (one timer active, and UART inactive)	f_{osc} = 6.0 MHz		960	3200	µA
		f_{osc} = 3.0 MHz		480	1800	µA
		f_{osc} = 1.0 MHz		350	1000	µA
I_{CC}	Wake-up mode 3 (UART active only)	f_{osc} = 6.0 MHz		1500	2200	µA
		f_{osc} = 3.0 MHz		800	1300	µA
		f_{osc} = 1.0 MHz		400	1100	µA
I_{CC}	Halt OSC-On	f_{osc} = 6.0 MHz		480	1120	µA
		f_{osc} = 3.0 MHz		240	560	µA
		f_{osc} = 1.0 MHz		80	200	µA
		f_{osc} = Z MHz			See Note 2	µA
I_{CC}	Halt OSC-Off			5	10	µA

Notes: 1) All inputs = V_{CC} or V_{SS} (except XTAL2). All output pins are open.
2) Maximum current = 180(Z) + 20 µA.

Table 4–29. Recommended Crystal/Clockin Operating Conditions over Full Operating Range

Parameter		Min	Typ	Max	Unit
f_{osc}	CLKIN frequency	0.5		6.0	MHz
	CLKIN duty cycle	47		53	%
$t_{c(P)}$	CLKIN cycle time	167		2000	ns
$t_{c(C)}$	Internal state cycle time	333		4000	ns
$t_{w(PH)}$	CLKIN pulse duration high	50			ns
$t_{w(PL)}$	CLKIN pulse duration low	50			ns
t_r	CLKIN rise time			20	ns
t_f	CLKIN fall time			10	ns
$t_{d(PL-CH)}$	CLKIN fall to CLKOUT rise delay		110	250	ns

† V_{CC} = 5 V, T_A = 25°C

Figure 4–26. Clock Timing

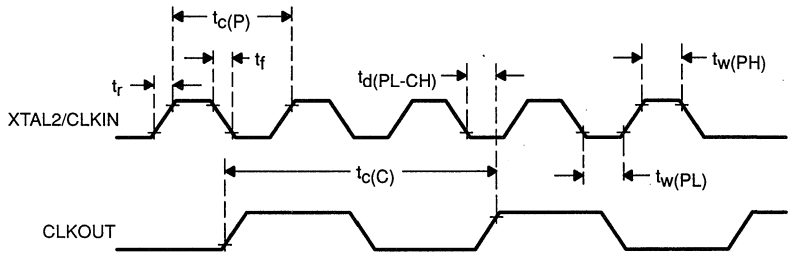
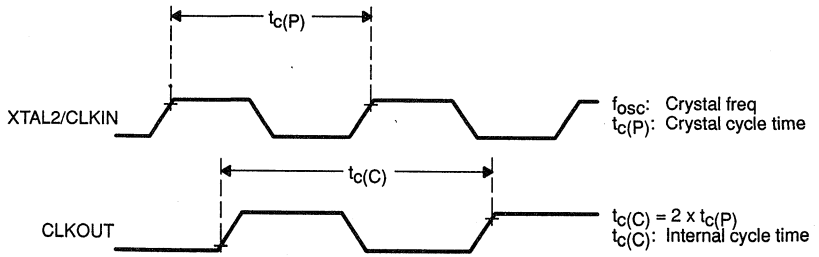


Table 4–30. Memory Interface Timings†

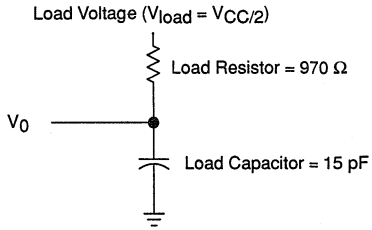
Parameter		Min	Typ†	Max	Unit
$t_{c(C)}$	CLKOUT cycle time		t_c		ns
$t_{w(CH)}$	CLKOUT high pulse duration	$0.5t_c-100$	$0.5t_c$	$0.5t_c+100$	ns
$t_{w(CL)}$	CLKOUT low pulse duration	$0.5t_c-100$	$0.5t_c$	$0.5t_c+100$	ns
$t_{c(R)}$	Read cycle time		$2t_c$		ns
$t_{c(W)}$	Write cycle time		$2t_c$		ns
$t_{d(CH-JH)}$	Clockout high to ALATCH high	$0.25t_c-40$	$0.25t_c$		ns
$t_{d(CH-JL)}$	Clockout high to ALATCH low	$0.5t_c-50$	$0.5t_c$		ns
$t_{d(CH-HA)}$	Clockout high to high address valid	$0.25t_c-40$	$0.25t_c$		ns
$t_{d(CH-EL)}$	Clockout high to $\overline{\text{ENABLE}}$ low	–10	30		ns
$t_{a(EL-D)}$	$\overline{\text{ENABLE}}$ active to data in	$0.75t_c-160$	$0.75t_c$		ns
$t_{h(EH-D)}$	Data in hold after $\overline{\text{ENABLE}}$ inactive	0			ns
$t_{a(A-D)}$	Data in from address valid	$1.5t_c-200$	$1.5t_c-100$		ns
$t_{W(JH)}$	ALATCH active	$0.25t_c-50$	$0.25t_c$		ns
$t_{su(HA-JL)}$	High address to ALATCH fall	$0.25t_c-50$	$0.25t_c$		ns
$t_{su(LA-JL)}$	Low address to ALATCH fall	$0.25t_c-55$	$0.25t_c$		ns
$t_{h(JL-LA)R}$	Low address hold from ALATCH fall (RD)	$0.25t_c-50$	$0.25t_c$		ns
$t_{h(EH-RW)}$	$\overline{\text{ENABLE}}$ inactive to R/$\overline{\text{W}}$	$0.5t_c-100$	$0.5t_c$		ns
$t_{d(EH-JH)}$	$\overline{\text{ENABLE}}$ inactive to ALATCH high	$0.5t_c-60$	$0.5t_c$		ns
$t_{d(EH-A)}$	$\overline{\text{ENABLE}}$ inactive to low address drive	$0.5t_c-100$	$0.5t_c$		ns
$t_{h(EH-HA)}$	High address hold from $\overline{\text{ENABLE}}$ inactive	$0.5t_c-100$	$0.5t_c$		ns
$t_{su(Q-EH)}$	Data out to $\overline{\text{ENABLE}}$ inactive	$0.5t_c-50$	$0.5t_c$		ns
$t_{h(EH-Q)}$	Data out hold from $\overline{\text{ENABLE}}$ inactive	$0.5t_c-60$	$0.5t_c$		ns
$t_{d(LA-EL)}$	Low address high-Z to $\overline{\text{ENABLE}}$ active	0	$0.25t_c$		ns
$t_{d(A-EH)}$	Low address to $\overline{\text{ENABLE}}$ high	$1.5t_c-100$	$1.5t_c$		ns
$t_{h(JL-LA)w}$	Low address hold from ALATCH fall (WR)	$0.75t_c-100$	$0.75t_c$		ns
$t_{su(RW-JL)}$	R/$\overline{\text{W}}$ valid before ALATCH fall	$0.25t_c-60$	$0.25t_c$		ns
$t_{w(E)}$	$\overline{\text{ENABLE}}$ pulse width	$0.75t_c-80$	$0.75t_c$		ns

† V_{CC} = 5 V ±10%, t_c = 2/freq



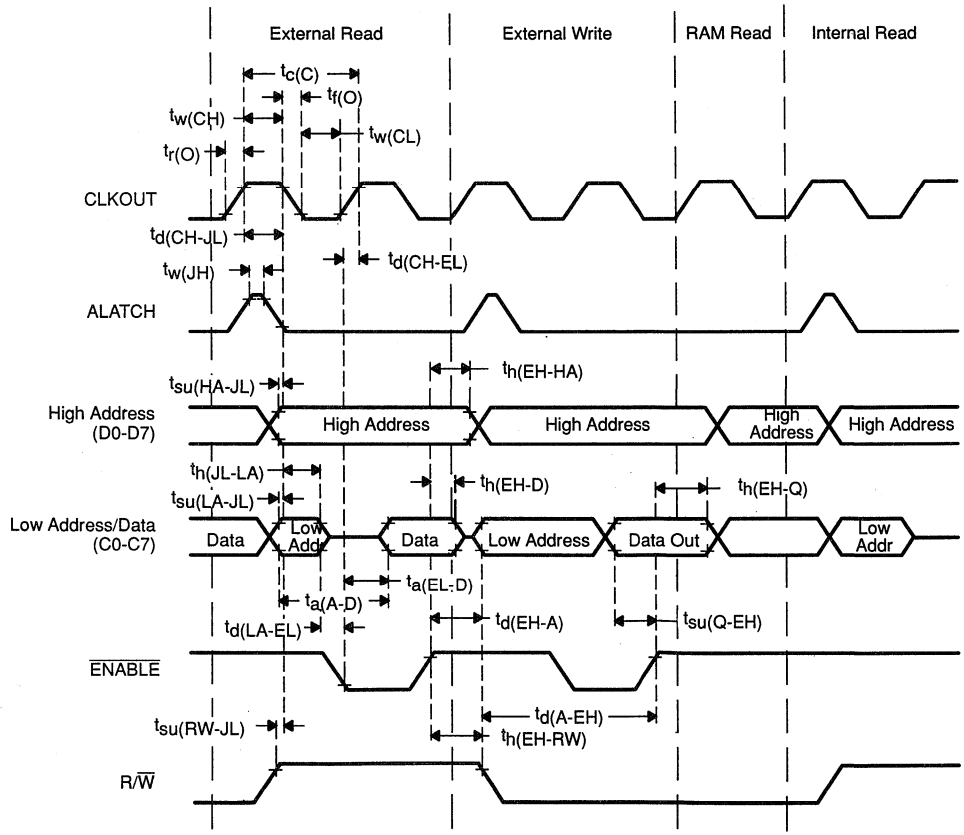
Note: Period of internal clock $t_{c(C)} = 2 \times t_{c(P)} = 2 / f_{osc}$. Timings are given in $t_{c(C)}$.

Figure 4–27. Output Loading Circuit for Test



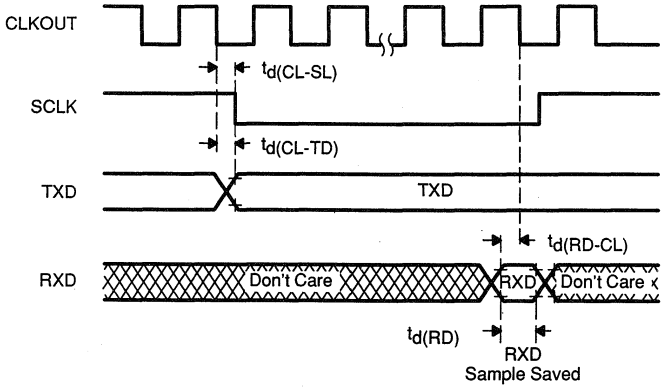
Note: Rise and fall times are measured between the maximum low level and the minimum high level using the 10% and 90% points.

Figure 4–28. Read and Write Cycle Timing



4.5.1 Serial Port Timing

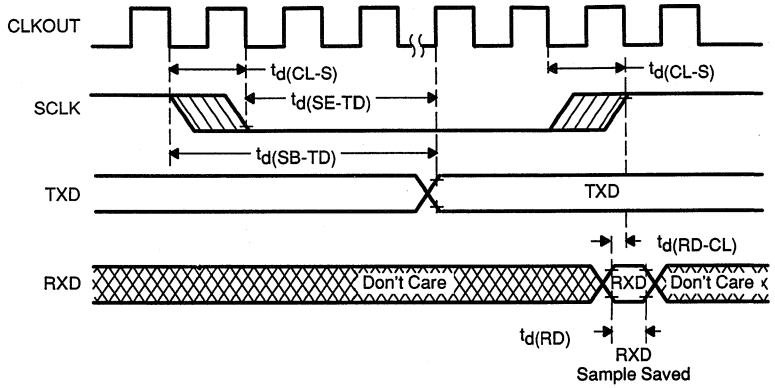
4.5.1.1 Internal Serial Clock



Notes: 1) The CLKOUT signal is not available in single-chip mode.
2) CLKOUT = $t_{c(C)}$.

Parameter		Typ	Unit
$t_{d(CL-SL)}$	CLKOUT low to SCLK low	1/4 $t_{c(C)}$	ns
$t_{d(CL-TD)}$	CLKOUT low to new TXD data	1/4 $t_{c(C)}$	ns
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns
$t_{c(S)}$	SCLK cycle time	4 $t_{c(C)}$	ns
$t_{w(SH)}$	SCLK out high	2 $t_{c(C)}$	ns
$t_{w(SL)}$	SCLK out low	2 $t_{c(C)}$	ns

4.5.1.2 External Serial Clock



Notes: 1) The CLKOUT signal is not available in single-chip mode.
2) CLKOUT = $t_{c(C)}$.
3) SCLK sampled; if SCLK = 1 then 0, fall transition found.
4) SCLK sampled; if SCLK = 0 then 1, rise transition found.

Parameter		Typ	Unit
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns
$t_{d(SB-TD)}$	Start of SCLK sample to new TXD data	3 1/4 $t_{c(C)}$	ns
$t_{d(SE-TD)}$	End of SCLK sample to new TXD data	2 1/4 $t_{c(C)}$	ns
$t_{d(CL-S)}$	Clockout low to SCLK transition	$t_{c(C)}$	ns
$t_{c(S)}$	SCLK cycle time	6 $t_{c(C)}$	ns
$t_{w(SH)}$	SCLK out high	2 $t_{c(C)}$	ns
$t_{w(SL)}$	SCLK out low	4 $t_{c(C)}$	ns

4.6 TMS70C08 and TMS70C48 Specifications

Table 4–31. Absolute Maximum Ratings over Operating Free-Air Temperature Range (Unless Otherwise Noted)

Supply voltage range, V_{CC}†	– 0.3 V to 7 V
Input voltage range	– 0.3 V to V_{CC}+0.3 V
Output voltage range	– 0.3 V to V_{CC}+0.3 V
Maximum buffer sink current (per pin)	±10 mA
Storage temperature range	– 55°C to 150°C
I_{CC}, I_{SS} (maximum into pin 25 or 40)	±60 mA

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Stresses beyond those listed under "Absolute Maximum Ratings" 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 beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

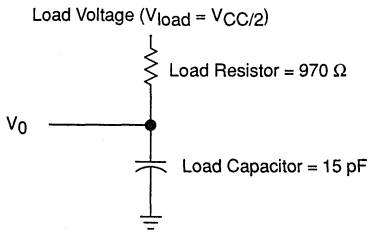
Table 4–32. Recommended Operating Conditions

			Min	Nom	Max	Unit
V_{CC}	Supply voltage		3.0		6.0	V
V_{IH}	High-level input voltage	$5.5\ V \le V_{CC} \le 6\ V$	$V_{CC}-1.0$		V_{CC}	V
		$4.5\ V \le V_{CC} \le 5.5\ V$	$V_{CC}-0.7$		V_{CC}	V
		$3.5\ V \le V_{CC} \le 4.5\ V$	$V_{CC}-0.5$		V_{CC}	V
		$2.5\ V \le V_{CC} \le 3.5\ V$	$V_{CC}-0.35$		V_{CC}	V
V_{IH}	High-level input voltage	$5\ V \le V_{CC} \le 6\ V$	$V_{CC}-0.5$		V_{CC}	V
		$4\ V \le V_{CC} < 5\ V$	$V_{CC}-0.4$		V_{CC}	V
		$3\ V \le V_{CC} < 4\ V$	$V_{CC}-0.3$		V_{CC}	V
		$2.5\ V \le V_{CC} < 3\ V$	$V_{CC}-0.2$		V_{CC}	V
V_{IH}	High-level input voltage	$5\ V \le V_{CC} \le 6\ V$	$V_{CC}-1.3$		V_{CC}	V
		$4\ V \le V_{CC} < 5\ V$	$V_{CC}-1.0$		V_{CC}	V
		$3\ V \le V_{CC} < 4\ V$	$V_{CC}-0.7$		V_{CC}	V
		$2.5\ V \le V_{CC} < 3\ V$	$V_{CC}-0.4$		V_{CC}	V
V_{IL}	Low-level input voltage	$5.5\ V \le V_{CC} \le 6\ V$	0		1.00	V
		$4.5\ V \le V_{CC} \le 5.5\ V$	0		0.70	V
		$3.5\ V \le V_{CC} \le 4.5\ V$	0		0.50	V
		$2.5\ V \le V_{CC} \le 3.5\ V$	0		0.35	V
V_{IL}	Low-level input voltage	$5\ V \le V_{CC} \le 6\ V$	0		0.5	V
		$4\ V \le V_{CC} < 5\ V$	0		0.4	V
		$3\ V \le V_{CC} < 4\ V$	0		0.3	V
		$2.5\ V \le V_{CC} < 3\ V$	0		0.2	V
V_{IL}	Low-level input voltage	$5\ V \le V_{CC} \le 6\ V$	0		1.5	V
		$4\ V \le V_{CC} < 5\ V$	0		1.1	V
		$3\ V \le V_{CC} < 4\ V$	0		0.7	V
		$2.5\ V \le V_{CC} < 3\ V$	0		0.3	V
T_A	Operating temperature	Commercial (N)	0		70	°C
f_{osc}	Oscillator frequency		0.5		7.5	MHz

Table 4–33. Electrical Characteristics over Full Range of Operating Conditions

Parameter		Test Conditions	Min	Typ†	Max	Unit
I_I	Input leakage current	MC pin, $V_{IN} = V_{SS}$ or V_{CC} All others, $V_{IN} = V_{SS}$ to V_{CC}		±0.1	±5	µA
C_I	Input capacitance			5		pF
V_{OH}	High-level output voltage	V_{CC} = 5.0 V, I_{OH} = – 1 mA	2.5	4.5		V
		V_{CC} = 5.0 V, I_{OH} = – 0.3mA	4.5	4.8		V
V_{OL}	Low-level output voltage	V_{CC} = 5.0 V, I_{OL} = 1.7 mA		0.3	0.4	V
I_{OH}	High-level output source current	V_{CC} = 5.0 V V_{OH} = 4.5 V	– 0.3	– 1.4		mA
		V_{CC} = 4.0 V V_{OH} = 3.5 V	– 0.2	– 1.0		mA
		V_{CC} = 3.0 V V_{OH} = 2.5 V	– 0.1	– 0.7		mA
		V_{CC} = 2.5 V V_{OH} = 2.0 V	– 0.05	– 0.3		mA
		V_{CC} = 5.0 V V_{OH} = 2.5 V	– 1.0	– 5.0		mA
I_{OL}	Output sink current	V_{CC} = 5.0 V V_{OL} = 0.4 V	1.7	2.8		mA
		V_{CC} = 4.0 V V_{OL} = 0.4 V	1.2	2.4		mA
		V_{CC} = 3.0 V V_{OL} = 0.4 V	0.7	2.0		mA
		V_{CC} = 2.5 V V_{OL} = 0.4 V	0.2	1.3		mA

Figure 4–29. Output Loading Circuit for Test



Note: Rise and fall times are measured between the maximum low level and the minimum high level using the 10% and 90% points.

Table 4–34. AC Characteristics for Input/Output Ports†

Parameter		Test Conditions	Min	Typ†	Max	Unit
$t_{r(IO)}$	I/O port output rise time	C_{load} = 15 pF, V_{CC} = 5 V		35	60	ns
$t_{f(IO)}$	I/O port output fall time	C_{load} = 15 pF, V_{CC} = 5 V		20	50	ns

† Rise and fall times are measured between the maximum low level and the minimum high level using the 10% and 90% points.

Figure 4–30. Measurement Points for Switching Characteristics

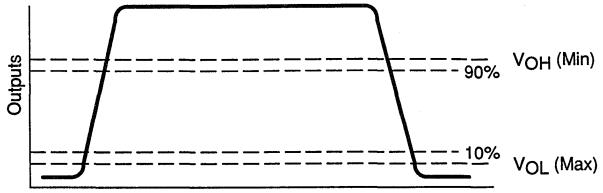


Table 4–35. Supply Current Requirements

Parameter		Test Conditions		Min	Typ	Max	Unit
I_{CC}	Operating mode	f_{osc} = 7.0 MHz,	V_{CC} = 5.0 V		17	24.5	mA
		f_{osc} = 3.0 MHz,	V_{CC} = 5.0 V		7.2	10.5	mA
		f_{osc} = 0.5 MHz,	V_{CC} = 5.0 V		1.2	1.8	mA
		f_{osc} = Z MHz,	V_{CC} = 5.0 V		2.4	3.5	mA/ MHz
		f_{osc} = 0.5 MHz,	V_{CC} = 2.5 V		0.4	1.2	mA
		f_{osc} = 8 MHz,	V_{CC} = 5.0 V (see Note 3)		20.0	36.0	mA
I_{CC}	Wake-up mode 1 (one timer and UART active)	f_{osc} = 7.0 MHz,	V_{CC} = 5.0 V		2400	5600	µA
		f_{osc} = 3.0 MHz,	V_{CC} = 5.0 V		1200	3300	µA
		f_{osc} = 0.5 MHz,	V_{CC} = 5.0 V		250	800	µA
		f_{osc} = 8 MHz,	V_{CC} = 5.0 V (see Note 3)		2600	5800	µA
I_{CC}	Wake-up mode 2 (one timer active and UART inactive)	f_{osc} = 7.0 MHz,	V_{CC} = 5.0 V		960	3400	µA
		f_{osc} = 3.0 MHz,	V_{CC} = 5.0 V		480	2000	µA
		f_{osc} = 0.5 MHz,	V_{CC} = 5.0 V		140	550	µA
		f_{osc} = 8 MHz,	V_{CC} = 5.0 V (see Note 3)		1100	3700	µA
I_{CC}	Wake-up mode 3 (UART active only)	f_{osc} = 7.0 MHz,	V_{CC} = 5.0 V		1500	2400	µA
		f_{osc} = 3.0 MHz,	V_{CC} = 5.0 V		800	1500	µA
		f_{osc} = 0.5 MHz,	V_{CC} = 5.0 V		180	600	µA
		f_{osc} = 8 MHz,	V_{CC} = 5.0 V (see Note 3)		1600	3000	µA
I_{CC}	Halt OSC-On	f_{osc} = 7.0 MHz,	V_{CC} = 5.0 V		560	1280	µA
		f_{osc} = 3.0 MHz,	V_{CC} = 5.0 V		240	560	µA
		f_{osc} = 1.0 MHz,	V_{CC} = 5.0 V		80	200	µA
		f_{osc} = Z MHz				See Note 2	µA
		f_{osc} = 8 MHz,	V_{CC} = 5.0 V (see Note 3)		800	1460	µA
I_{CC}	Halt OSC-Off				5	10	µA

Notes: 1) All inputs = V_{CC} or V_{SS} (except XTAL2). All I/O and output pins are open.
2) Maximum current = 180(Z) + 20 µA.
3) 8 MHz capability is defined at test level. Such devices should be ordered under SPEC2 special request.

Table 4–36. Recommended Crystal/Clockin Operating Conditions over Full Operating Range

Parameter		Min	Typ†	Max	Unit
f_{osc}	Crystal frequency	0.5		6.0	MHz
	CLKIN duty cycle	45		55	%
$t_{c(P)}$	Crystal cycle time ‡	167		2000	ns
$t_{c(C)}$	Internal state cycle time	333		4000	ns
$t_{w(PH)}$	CLKIN pulse duration high	70			ns
$t_{w(PL)}$	CLKIN pulse duration low	70			ns
t_r	CLKIN rise time			30	ns
t_f	CLKIN fall time			30	ns
$t_{d(PL-CH)}$	CLKIN fall to CLKOUT rise delay		110	250	ns

† V_{CC} = 5 V, T_A = 25°C
‡ See Section 3.4 for Recommended Clock Connections.

Figure 4–31. Clock Timing

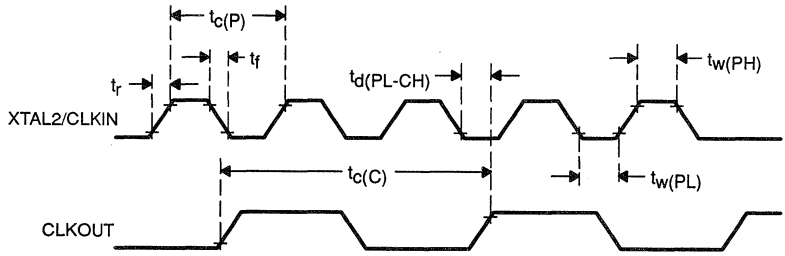
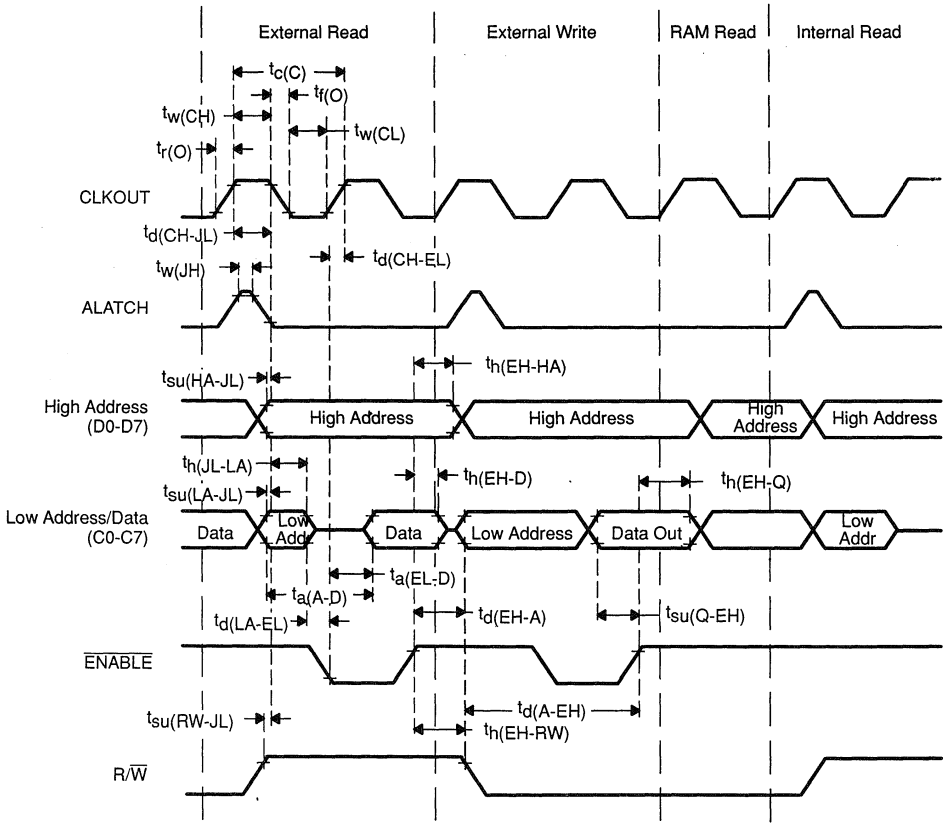


Table 4–37. Memory Interface Timings†

	Parameter	Min	Typ†	Max	Unit
$t_{c(C)}$	CLKOUT cycle time		t_c		ns
$t_{w(CH)}$	CLKOUT high pulse duration	$0.5t_c-100$	$0.5t_c$	$0.5t_c+100$	ns
$t_{w(CL)}$	CLKOUT low pulse duration	$0.5t_c-100$	$0.5t_c$	$0.5t_c+100$	ns
$t_{c(R)}$	Read cycle time		$2t_c$		ns
$t_{c(W)}$	Write cycle time		$2t_c$		ns
$t_{d(CH-JH)}$	Clockout high to ALATCH high	$0.25t_c-40$	$0.25t_c$		ns
$t_{d(CH-JL)}$	Clockout high to ALATCH low	$0.5t_c-50$	$0.5t_c$		ns
$t_{d(CH-HA)}$	Clockout high to high address valid	$0.25t_c-40$	$0.25t_c$		ns
$t_{d(CH-EL)}$	Clockout high to $\overline{\text{ENABLE}}$ low	−10	30		ns
$t_{a(EL-D)}$	$\overline{\text{ENABLE}}$ active to data in	$0.75t_c-160$	$0.75t_c$		ns
$t_{h(EH-D)}$	Data in hold after $\overline{\text{ENABLE}}$ inactive	0			ns
$t_{a(A-D)}$	Data in from address valid	$1.5t_c-200$	$1.5t_c-100$		ns
$t_{W(JH)}$	ALATCH active	$0.25t_c-50$	$0.25t_c$		ns
$t_{su(HA-JL)}$	High address to ALATCH fall	$0.25t_c-50$	$0.25t_c$		ns
$t_{su(LA-JL)}$	Low address to ALATCH fall	$0.25t_c-55$	$0.25t_c$		ns
$t_{h(JL-LA)R}$	Low address hold from ALATCH fall (RD)	$0.25t_c-50$	$0.25t_c$		ns
$t_{h(EH-RW)}$	$\overline{\text{ENABLE}}$ inactive to R/$\overline{\text{W}}$	$0.5t_c-100$	$0.5t_c$		ns
$t_{d(EH-JH)}$	$\overline{\text{ENABLE}}$ inactive to ALATCH high	$0.5t_c-60$	$0.5t_c$		ns
$t_{d(EH-A)}$	$\overline{\text{ENABLE}}$ inactive to low address drive	$0.5t_c-100$	$0.5t_c$		ns
$t_{h(EH-HA)}$	High address hold from $\overline{\text{ENABLE}}$ inactive	$0.5t_c-100$	$0.5t_c$		ns
$t_{su(Q-EH)}$	Data out to $\overline{\text{ENABLE}}$ inactive	$0.5t_c-50$	$0.5t_c$		ns
$t_{h(EH-Q)}$	Data out hold from $\overline{\text{ENABLE}}$ inactive	$0.5t_c-60$	$0.5t_c$		ns
$t_{d(LA-EL)}$	Low address high-Z to $\overline{\text{ENABLE}}$ active	0	$0.25t_c$		ns
$t_{d(A-EH)}$	Low address to $\overline{\text{ENABLE}}$ high	$1.5t_c-100$	$1.5t_c$		ns
$t_{h(JL-LA)w}$	Low address hold from ALATCH fall (WR)	$0.75t_c-100$	$0.75t_c$		ns
$t_{su(RW-JL)}$	R/$\overline{\text{W}}$ valid before ALATCH fall	$0.25t_c-60$	$0.25t_c$		ns
$t_{w(E)}$	$\overline{\text{ENABLE}}$ pulse width	$0.75t_c-80$	$0.75t_c$		ns

† V_{CC} = 5 V ±10%, t_c = 2/freq

Figure 4–32. Read and Write Cycle Timing



4.7 SE70CP160A Specifications

These specifications are for wide-voltage operation. For operation at 5 V ±10%, see Section 4.2. Be sure to use an EPROM that uses similar supply voltage specifications.

Table 4–38. Absolute Maximum Rating over Operating Free-Air Temperature Range (Unless Otherwise Noted)

Supply voltage, V_{CC}† . . . . . – 0.3 V to 7 V
All input voltages . . . . . – 0.3 V to V_{CC} + 0.3 V
All output voltages . . . . . – 0.3 V to V_{CC} + 0.3 V
Maximum I/O buffer current (per pin) . . . . . ±10 mA
Storage temperature range . . . . . – 55°C to 150°C
I_{CC}, I_{SS} current (maximum into pins 25 and 40) . . . . . ±60 mA

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Stresses beyond those listed under "Absolute Maximum Ratings" 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 beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

Table 4–39. Recommended Operating Conditions

			Min	Nom	Max	Unit
V_{CC}	Supply voltage		2.5		6.0	V
V_{IH}	High-level input voltage	XTAL2 pin, V_{CC} = 2.5 to 6.0 V	0.8 V_{CC}			V
		All other pins, V_{CC} = 3.0 to 6.0 V	0.7 V_{CC}			V
		All other pins, V_{CC} = 2.5 to 3.0 V	0.75 V_{CC}			V
V_{IL}	Low-level input voltage	XTAL2 pin, V_{CC} = 2.5 to 6.0 V			0.2 V_{CC}	V
		All other pins, V_{CC} = 2.5 to 6.0 V			0.3 V_{CC}	V
T_A	Operating temperature range		0		55	°C

Table 4–40. Electrical Characteristics over Full Range of Operating Conditions

Parameter		Test Conditions		Min	Typ†	Max	Unit
I_I	Input leakage current	$V_{IN} = V_{SS}$ to V_{CC}			±0.10	±5	µA
C_I	Input capacitance				5		pF
V_{OH}	High-level output voltage ‡	V_{CC} = 2.5 V,	I_{OH} = – 50 µA	2.25	2.4		V
		V_{CC} = 4.0 V,	I_{OH} = – 0.4 mA	3.2	3.6		V
		V_{CC} = 5.0 V,	I_{OH} = – 0.7 mA	3.9	4.5		V
		V_{CC} = 6.0 V,	I_{OH} = – 1.0 mA	4.6	5.4		V
V_{OL}	Low-level output voltage ‡	V_{CC} = 2.5 V,	I_{OL} = 0.4 mA		0.2	0.35	V
		V_{CC} = 4.0 V,	I_{OL} = 1.6 mA		0.4	0.8	V
		V_{CC} = 5.0 V,	I_{OL} = 2.5 mA		0.6	1.1	V
		V_{CC} = 6.0 V,	I_{OL} = 3.4 mA		0.8	1.4	V
I_{OH}	Output source current	V_{CC} = 2.5 V,	V_{OH} = 2.25 V	– 0.05	– 0.2		mA
		V_{CC} = 4.0 V,	V_{OH} = 3.2 V	– 0.4	– 1.4		mA
		V_{CC} = 5.0 V,	V_{OH} = 3.9 V	– 0.7	– 2.2		mA
		V_{CC} = 6.0 V,	V_{OH} = 4.6 V	– 1.0	– 3.3		mA
I_{OL}	Output sink current	V_{CC} = 2.5 V,	V_{OL} = 0.35 V	0.4	0.9		mA
		V_{CC} = 4.0 V,	V_{OL} = 0.8 V	1.6	3.5		mA
		V_{CC} = 5.0 V,	V_{OL} = 1.1 V	2.5	5.5		mA
		V_{CC} = 6.0 V,	V_{OL} = 1.4 V	3.4	8.0		mA

† V_{CC} = 5 V, T_A = 25°C
‡ Output levels ensure 400 mV of noise margin over specified input levels.

Table 4–41. Supply Current Requirements

Parameter		Test Conditions	Min	Typ	Max	Unit
I_{CC}	Operating mode	f_{osc} = 6.0 MHz, V_{CC} = 5 V		9.0	14.4	mA
		f_{osc} = 3.0 MHz, V_{CC} = 5 V		4.5	7.2	mA
		f_{osc} = 0.5 MHz, V_{CC} = 5 V		0.8	1.2	mA
		f_{osc} = Z MHz, V_{CC} = 5 V		1.5	2.4	mA/ MHz
		f_{osc} = 0.5 MHz, V_{CC} = 2.5 V		370	800	µA
I_{CC}	Wake-up mode (timer active)	f_{osc} = 6.0 MHz, V_{CC} = 5 V		960	1920	µA
		f_{osc} = 3.0 MHz, V_{CC} = 5 V		480	960	µA
		f_{osc} = 0.5 MHz, V_{CC} = 5 V		80	160	µA
		f_{osc} = Z MHz, V_{CC} = 5 V		160	320	µA/ MHz
		f_{osc} = 0.5 MHz, V_{CC} = 2.5 V		40	80	µA
I_{CC}	Halt OSC-On	f_{osc} = 6.0 MHz, V_{CC} = 5 V		480	980	µA
		f_{osc} = 3.0 MHz, V_{CC} = 5 V		240	500	µA
		f_{osc} = 0.5 MHz, V_{CC} = 5 V		45	100	µA
		f_{osc} = Z MHz, V_{CC} = 5 V		See Note 2		µA
		f_{osc} = 0.5 MHz, V_{CC} = 2.5 V		25	60	µA
I_{CC}	Halt OSC-Off	V_{CC} = 2.5 to 6 V		1	10	µA

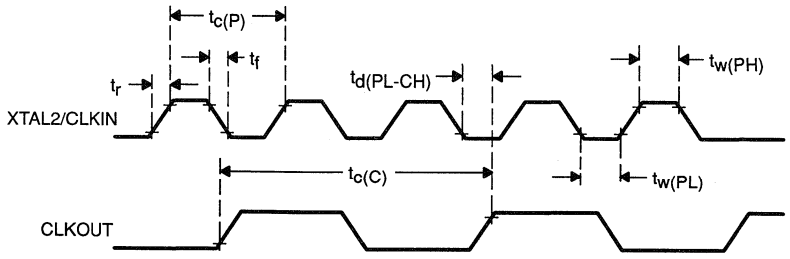
Notes: 1) All inputs = V_{CC} or V_{SS} (except XTAL2). All output pins are open.
2) Maximum current = 160(Z) + 20 µA.
3) I_{CC} applies to the supply current of the SE70CP160A without an EPROM device installed.

Table 4–42. Recommended Crystal/Clockin Operating Conditions over Full Operating Range

Parameter		Test Conditions	Min	Typ†	Max	Unit
f_{osc}	Crystal frequency	V_{CC} = 2.5 V	0.5		0.8	MHz
		V_{CC} = 4.0 V	0.5		4.0	MHz
		V_{CC} = 5.0 V	0.5		6.0	MHz
		V_{CC} = 6.0 V	0.5		6.5	MHz
	CLKIN duty cycle		45		55	%
$t_{c(P)}$	Crystal cycle time ‡	V_{CC} = 2.5 V	1250		2000	ns
		V_{CC} = 4.0 V	250		2000	ns
		V_{CC} = 5.0 V	166		2000	ns
		V_{CC} = 6.0 V	153		2000	ns
$t_{c(C)}$	Internal state cycle time	V_{CC} = 2.5 V	2500		4000	ns
		V_{CC} = 4.0 V	500		4000	ns
		V_{CC} = 5.0 V	333		4000	ns
		V_{CC} = 6.0 V	306		4000	ns
$t_{w(PH)}$	CLKIN pulse duration high		50			ns
$t_{w(PL)}$	CLKIN pulse duration low		50			ns
t_r	CLKIN rise time				30	ns
t_f	CLKIN fall time				30	ns
$t_{d(PL-CH)}$	CLKIN fall to CLKOUT rise delay			140	250	ns

† V_{CC} = 5 V, T_A = 25°C
‡ See Section 3.4 for Recommended Clock Connections.

Figure 4–33. Clock Timing



4.8 TMS77C82 Specifications

Table 4–43. Absolute Maximum Ratings over Operating Free-Air Temperature Range (Unless Otherwise Noted)

Supply voltage range, V_{CC}†	– 0.3 V to 7 V
Input voltage range	– 0.3 V to V_{CC}+0.3 V
Output voltage range	– 0.3 V to V_{CC}+0.3 V
Maximum I/O buffer current (per pin)	±10 mA
Storage temperature range	– 55°C to 150°C
I_{CC}, I_{SS} (maximum into pin 25 or 40)	±60 mA
Supply voltage range V_{PP} (MC pin)	– 0.3 to 14 V

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Stresses beyond those listed under "Absolute Maximum Ratings" 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 beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute maximum rated conditions for extended periods may affect device reliability.

Table 4–44. Recommended Operating Conditions

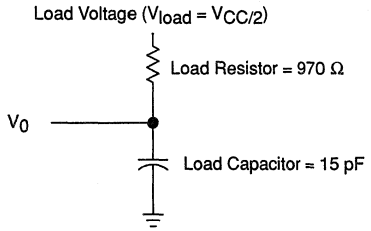
			Min	Nom	Max	Unit
V_{CC}	Supply voltage		3.0		6.0	V
V_{PP}	Programming supply voltage (MC pin)		12.0		13.0	V
	INT1, INT3, RESET, and XTAL Pins					
V_{IH}	High-level input voltage	5.5 V ≤ V_{CC} ≤ 6 V	V_{CC} – 1.0		V_{CC}	V
		4.5 V ≤ V_{CC} ≤ 5.5 V	V_{CC} – 0.7		V_{CC}	V
		3.5 V ≤ V_{CC} ≤ 4.5 V	V_{CC} – 0.5		V_{CC}	V
		3 V ≤ V_{CC} ≤ 3.5 V	V_{CC} – 0.35		V_{CC}	V
	MC Pin					
V_{IH}	High-level input voltage	5 V ≤ V_{CC} ≤ 6 V	V_{CC} – 0.5		V_{CC}	V
		4 V ≤ V_{CC} < 5 V	V_{CC} – 0.4		V_{CC}	V
		3 V ≤ V_{CC} < 4 V	V_{CC} – 0.3		V_{CC}	V
		3 V ≤ V_{CC} < 3 V	V_{CC} – 0.2		V_{CC}	V
	Port (All Other Pins)					
V_{IH}	High-level input voltage	5 V ≤ V_{CC} ≤ 6 V	V_{CC} – 1.3		V_{CC}	V
		4 V ≤ V_{CC} < 5 V	V_{CC} – 1.0		V_{CC}	V
		3 V ≤ V_{CC} < 4 V	V_{CC} – 0.7		V_{CC}	V
		3 V ≤ V_{CC} < 3 V	V_{CC} – 0.4		V_{CC}	V
	INT1, INT3, RESET, and XTAL Pins					
V_{IL}	Low-level input voltage	5.5 V ≤ V_{CC} ≤ 6 V	0		1.00	V
		4.5 V ≤ V_{CC} ≤ 5.5 V	0		0.70	V
		3.5 V ≤ V_{CC} ≤ 4.5 V	0		0.50	V
		3 V ≤ V_{CC} ≤ 3.5 V	0		0.35	V
	MC Pin					
V_{IL}	Low-level input voltage	5 V ≤ V_{CC} ≤ 6 V	0		0.5	V
		4 V ≤ V_{CC} < 5 V	0		0.4	V
		3 V ≤ V_{CC} < 4 V	0		0.3	V
		3 V ≤ V_{CC} < 3 V	0		0.2	V
	Port (All Other Pins)					
V_{IL}	Low-level input voltage	5 V ≤ V_{CC} ≤ 6 V	0		1.5	V
		4 V ≤ V_{CC} < 5 V	0		1.1	V
		3 V ≤ V_{CC} < 4 V	0		0.7	V
		3 V ≤ V_{CC} < 3 V	0		0.3	V
T_A	Operating free-air temperature	Commercial (N)	0		70	°C
		Industrial (NA)	– 40		85	°C
f_{osc}	Oscillator frequency		0.5		7.5	MHz

Table 4–45. Electrical Characteristics over Full Range of Operating Conditions

Parameter		Test Conditions	Min	Typ†	Max	Unit
I_I	Input current	MC pin, $V_{IN} = V_{SS}$ or V_{CC} All others, $V_{IN} = V_{SS}$ to V_{CC}		±0.10	±5	µA
C_I	Input capacitance			5		pF
V_{OH}	High-level output voltage ‡	V_{CC} = 5.0 V, I_{OH} = – 1.0 mA	2.5	4.5		V
		V_{CC} = 5.0 V, I_{OH} = – 0.3 mA	4.5	4.8		V
V_{OL}	Low level output voltage	V_{CC} = 5.0 V, I_{OL} = 1.7 mA		0.3	0.4	V
I_{OH}	Output source current	V_{CC} = 5.0 V, V_{OH} = 4.5 V	– 0.3	– 1.4		µA
		V_{CC} = 4.0 V, V_{OH} = 3.5 V	– 0.2	– 1.0		mA
		V_{CC} = 3.0 V, V_{OH} = 2.5 V	– 0.1	– 0.7		mA
		V_{CC} = 5.0 V, V_{OH} = 2.5 V	– 1.0	– 5.0		mA
I_{OL}	Output sink current	V_{CC} = 5.0 V, V_{OL} = 0.4 V	1.7	2.8		mA
		V_{CC} = 4.0 V, V_{OL} = 0.4 V	1.2	2.4		mA
		V_{CC} = 3.0 V, V_{OL} = 0.4 V	0.7	2.0		mA

† V_{CC} = 5 V, T_A = 25°C
‡ Output levels ensure 400 mV of noise margin over specified input levels.

Figure 4–34. Output Loading Circuit for Test



Note: Rise and fall times are measured between the maximum low level and the minimum high level using the 10% and 90% points.

Table 4–46. Supply Current Requirements

Parameter		Test Conditions	Min	Typ	Max	Unit
I_{CC}	Supply current	f_{osc} = 6.0 MHz, V_{DD} = 5.0 V		16.1	24.8	mA
		f_{osc} = 3.0 MHz, V_{DD} = 5.0 V		14.0	21.2	mA
		f_{osc} = 1.0 MHz, V_{DD} = 5.0 V		11.2	17.2	mA
I_{CC}	Wake-up modes, 1 and 5	f_{osc} = F MHz		1.2F+0.3	2F+0.5	mA
	(See Note 2.)	f_{osc} = 6.0 MHz, V_{DD} = 5.0 V		7.5	12.5	mA
I_{CC}	Wake-up mode 2	f_{osc} = F MHz		0.86F+0.3	1.44F+0.5	mA
	(See Note 2.)	f_{osc} = 6.0 MHz, V_{DD} = 5.0 V		5.5	9.1	mA
I_{CC}	Wake-up modes, 3 and 4	f_{osc} = F MHz		0.52F+0.3	0.8F+0.5	mA
	(See Note 2.)	f_{osc} = 6.0 MHz, V_{DD} = 5.0 V		3.4	5.8	mA
I_{CC}	Halt OSC-On	f_{osc} = F MHz		0.1F+0.3	0.17F+0.5	mA
		f_{osc} = 6.0 MHz, V_{DD} = 5.0 V		0.9	1.52	mA
I_{CC}	Halt OSC-Off XTAL option	V_{DD} = 5.0 V			20	µA
I_{CC}	Halt OSC-Off RC option	V_{DD} = 5.0 V			7	µA

Notes: 1) All inputs = V_{CC} or V_{SS} (except XTAL2). All output pins are open.
2) Refer to table below:

Mode	Timer-1	Timer-2	UART
Wake-up 1	use	use	use
Wake-up 2	Timer-1 or 2 use		use
Wake-up 3	off	off	use
Wake-up 4	Timer-1 or 2 use		off
Wake-up 5	use	use	off
Halt-Off	off	off	off
Halt-On	off	off	off

Table 4–47. Recommended Crystal/Clockin Operating Conditions over Full Operating Range

Parameter		Test Conditions	Min	Typ†	Max	Unit
f_{osc}	Crystal frequency	V_{CC} = 3.0 V	0.5		3.0	MHz
		V_{CC} = 4.0 V	0.5		5.0	MHz
		V_{CC} = 5.0 V	0.5		7.0	MHz
		V_{CC} = 6.0 V	0.5		7.5	MHz
	CLKIN duty cycle		45		55	%
$t_{c(P)}$	Crystal cycle time ‡	V_{CC} = 2.5 V	1250		2000	ns
		V_{CC} = 4.0 V	200		2000	ns
		V_{CC} = 5.0 V	143		2000	ns
		V_{CC} = 6.0 V	133		2000	ns
$t_{c(C)}$	Internal state cycle time	V_{CC} = 2.5 V	2500		4000	ns
		V_{CC} = 4.0 V	400		4000	ns
		V_{CC} = 5.0 V	286		4000	ns
		V_{CC} = 6.0 V	267		4000	ns
$t_{w(PH)}$	CLKIN pulse duration high		50			ns
$t_{w(PL)}$	CLKIN pulse duration low		50			ns
t_r	CLKIN rise time				30	ns
t_f	CLKIN fall time				30	ns
$t_{d(PL-CH)}$	CLKIN fall to CLKOUT rise delay			110	250	ns

† V_{CC} = 5 V, T_A = 25°C

‡ See Section 3.4 for Recommended Clock Connections.

Figure 4–35. Clock Timing

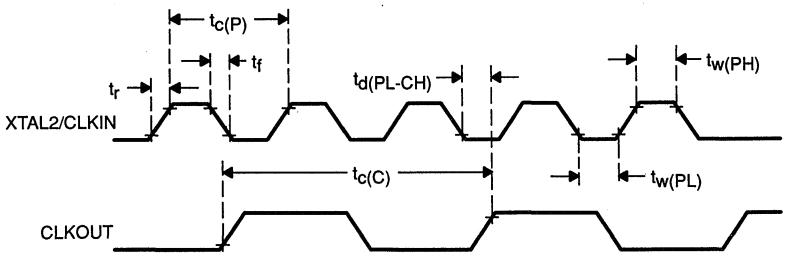


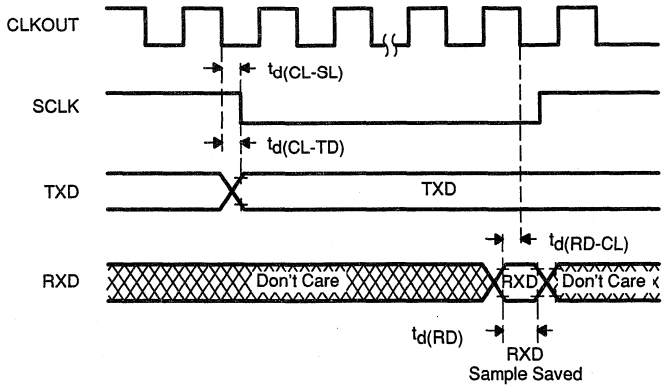
Table 4–48. Memory Interface Timings

	Parameter	Min	Typ†	Max	Unit
$t_{c(C)}$	CLKOUT cycle time		t_c		ns
$t_{w(CH)}$	CLKOUT high pulse duration	$0.5t_c - 100$	$0.5t_c$	$0.5t_c+100$	ns
$t_{w(CL)}$	CLKOUT low pulse duration	$0.5t_c - 100$	$0.5t_c$	$0.5t_c+100$	ns
$t_{c(R)}$	Read cycle time		$2t_c$		ns
$t_{c(W)}$	Write cycle time		$2t_c$		ns
$t_{d(CH-JH)}$	Clockout high to ALATCH high	$0.25t_c - 40$	$0.25t_c$		ns
$t_{d(CH-JL)}$	Clockout high to ALATCH low	$0.5t_c - 50$	$0.5t_c$		ns
$t_{d(CH-HA)}$	Clockout high to high address valid	$0.25t_c - 40$	$0.25t_c$		ns
$t_{d(CH-EL)}$	Clockout high to $\overline{\text{ENABLE}}$ low	-10	30		ns
$t_{a(EL-D)}$	$\overline{\text{ENABLE}}$ active to data in	$0.75t_c - 60$	$0.75t_c$		ns
$t_{h(EH-D)}$	Data in hold after $\overline{\text{ENABLE}}$ inactive	0			ns
$t_{a(A-D)}$	Data in from address valid	$1.5t_c - 200$	$1.5t_c - 100$		ns
$t_{w(JH)}$	ALATCH active	$0.25t_c - 50$	$0.25t_c$		ns
$t_{su(HA-JL)}$	High address to ALATCH fall	$0.25t_c - 50$	$0.25t_c$		ns
$t_{su(LA-JL)}$	Low address to ALATCH fall	$0.25t_c - 55$	$0.25t_c$		ns
$t_{h(JL-LA)R}$	Low address hold from ALATCH fall (RD)	$0.25t_c - 50$	$0.25t_c$		ns
$t_{h(EH-RW)}$	$\overline{\text{ENABLE}}$ inactive to R/$\overline{\text{W}}$	$0.5t_c - 100$	$0.5t_c$		ns
$t_{d(EH-JH)}$	$\overline{\text{ENABLE}}$ inactive to ALATCH high	$0.5t_c - 60$	$0.5t_c$		ns
$t_{d(EH-A)}$	$\overline{\text{ENABLE}}$ inactive to low address drive	$0.5t_c - 100$	$0.5t_c$		ns
$t_{h(EH-HA)}$	High address hold from $\overline{\text{ENABLE}}$ inactive	$0.5t_c - 100$	$0.5t_c$		ns
$t_{su(Q-EH)}$	Data out to $\overline{\text{ENABLE}}$ inactive	$0.5t_c - 50$	$0.5t_c$		ns
$t_{h(EH-Q)}$	Data out hold from $\overline{\text{ENABLE}}$ inactive	$0.5t_c - 60$	$0.5t_c$		ns
$t_{d(LA-EL)}$	Low address high-Z to $\overline{\text{ENABLE}}$ active	0	$0.25t_c$		ns
$t_{d(A-EH)}$	Low address to $\overline{\text{ENABLE}}$ high	$1.5t_c - 100$	$1.5t_c$		ns
$t_{h(JL-LA)W}$	Low address hold from ALATCH fall (WR)	$0.75t_c - 100$	$0.75t_c$		ns
$t_{su(RW-JL)}$	R/$\overline{\text{W}}$ valid before ALATCH fall	$0.25t_c - 60$	$0.25t_c$		ns
$t_{w(E)}$	$\overline{\text{ENABLE}}$ pulse width	$0.75t_c - 80$	$0.75t_c$		ns

† V_{CC} = 5 V ±10%, t_c = 2/freq

4.8.1 Serial Port Timing

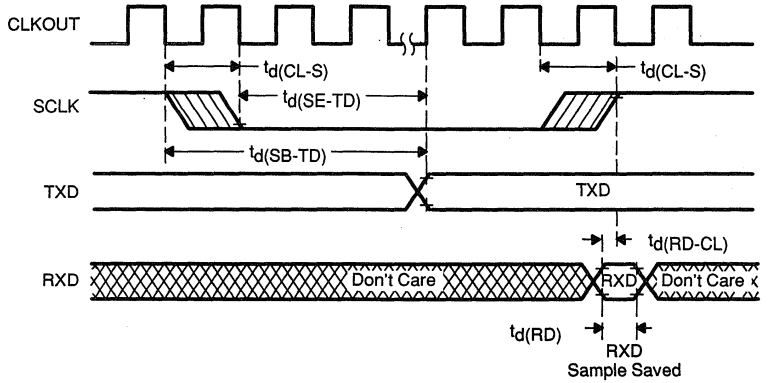
4.8.1.1 Internal Serial Clock



Notes: 1) The CLKOUT signal is not available in single-chip mode.
2) CLKOUT = $t_{c(C)}$.

Parameter		Typ	Unit
$t_{d(CL-SL)}$	CLKOUT low to SCLK low	1/4 $t_{c(C)}$	ns
$t_{d(CL-TD)}$	CLKOUT low to new TXD data	1/4 $t_{c(C)}$	ns
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns
$t_{c(S)}$	SCLK cycle time	4 $t_{c(C)}$	ns
$t_{w(SH)}$	SCLK out high	2 $t_{c(C)}$	ns
$t_{w(SL)}$	SCLK out low	2 $t_{c(C)}$	ns

4.8.1.2 External Serial Clock



Notes: 1) The CLKOUT signal is not available in single-chip mode.
2) CLKOUT = $t_{c(C)}$.
3) SCLK sampled; if SCLK = 1 then 0, fall transition found.
4) SCLK sampled; if SCLK = 0 then 1, rise transition found.

Parameter		Typ	Unit
$t_{d(RD-CL)}$	RXD data valid before CLKOUT low	1/4 $t_{c(C)}$	ns
$t_{d(RD)}$	RXD data valid time	1/2 $t_{c(C)}$	ns
$t_{d(SB-TD)}$	Start of SCLK sample to new TXD data	3 1/4 $t_{c(C)}$	ns
$t_{d(CL-S)}$	Clockout low to SCLK transition	$t_{c(C)}$	ns
$t_{c(S)}$	SCLK Cycle Time	6 $t_{c(C)}$	ns
$t_{w(SH)}$	SCLK High	2 $t_{c(C)}$	ns
$t_{w(SL)}$	SCLK Low	4 $t_{c(C)}$	ns

4.9 SE70CP168 Specifications

Table 4–49. Absolute Maximum Ratings over Operating Free-Air Temperature Range (unless otherwise noted)

Supply voltage range, V_{CC}† . . . . . – 0.3 V to 7 V
Input voltage range . . . . . – 0.3 V to V_{CC}+0.3 V
Output voltage range . . . . . – 0.3 V to V_{CC}+0.3 V
Storage temperature range . . . . . – 55°C to 150°C
Input current . . . . . ±10 mA
Continuous power dissipation . . . . . 0.5 W

† Unless otherwise noted, all voltages are with respect to V_{SS}.

Stresses beyond those listed under "Absolute Maximum Ratings" 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 beyond those indicated in the "Recommended Operating Conditions" section of this specification is not implied. Exposure to absolute maximum rated conditions for extended periods may affect device reliability.

Table 4–50. Recommended Operating Conditions

			Min	Nom	Max	Unit
V_{CC}	Supply voltage		2.5		6.0	V
INT1, INT3, RESET, and XTAL Pins						
V_{IH}	High-level input voltage	5.5 V ≤ V_{CC} ≤ 6 V	V_{CC} – 1.0		V_{CC}	V
		4.5 V ≤ V_{CC} ≤ 5.5 V	V_{CC} – 0.7		V_{CC}	V
		3.5 V ≤ V_{CC} ≤ 4.5 V	V_{CC} – 0.5		V_{CC}	V
		2.5 V ≤ V_{CC} ≤ 3.5 V	V_{CC} – 0.35		V_{CC}	V
MC Pin						
V_{IH}	High-level input voltage	5 V ≤ V_{CC} ≤ 6 V	V_{CC} – 0.5		V_{CC}	V
		4 V ≤ V_{CC} < 5 V	V_{CC} – 0.4		V_{CC}	V
		3 V ≤ V_{CC} < 4 V	V_{CC} – 0.3		V_{CC}	V
		2.5 V ≤ V_{CC} < 3 V	V_{CC} – 0.2		V_{CC}	V
Port (All Other Pins)						
V_{IH}	High-level input voltage	5 V ≤ V_{CC} ≤ 6 V	V_{CC} – 1.3		V_{CC}	V
		4 V ≤ V_{CC} < 5 V	V_{CC} – 1.0		V_{CC}	V
		3 V ≤ V_{CC} < 4 V	V_{CC} – 0.7		V_{CC}	V
		2.5 V ≤ V_{CC} < 3 V	V_{CC} – 0.4		V_{CC}	V

Table 4–50. Recommended Operating Conditions (Continued)

			Min	Nom	Max	Unit
INT1, INT3, RESET, and XTAL Pins						
V_{IL}	Low-level input voltage	5.5 V ≤ V_{CC} ≤ 6 V	0		1.00	V
		4.5 V ≤ V_{CC} ≤ 5.5 V	0		0.70	V
		3.5 V ≤ V_{CC} ≤ 4.5 V	0		0.50	V
		2.5 V ≤ V_{CC} ≤ 3.5 V	0		0.35	V
MC Pin						
V_{IL}	Low-level input voltage	5 V ≤ V_{CC} ≤ 6 V	0		0.5	V
		4 V ≤ V_{CC} < 5 V	0		0.4	V
		3 V ≤ V_{CC} < 4 V	0		0.3	V
		2.5 V ≤ V_{CC} < 3 V	0		0.2	V
Port (All Other Pins)						
V_{IL}	Low-level input voltage	5 V ≤ V_{CC} ≤ 6 V	0		1.5	V
		4 V ≤ V_{CC} < 5 V	0		1.1	V
		3 V ≤ V_{CC} < 4 V	0		0.7	V
		2.5 V ≤ V_{CC} < 3 V	0		0.3	V
T_A	Operating free-air temperature	Commercial (N)	0		70	°C
f_{osc}	Oscillator frequency		0.5		7.5	MHz

Table 4–51. Electrical Characteristics over the Full Range of Operating Conditions

Parameter		Test Conditions	Min	Typ†	Max	Unit
I_I	Input current	MC pin, V_{IN} = V_{SS} or V_{CC} All others, V_{IN} = V_{SS} to V_{CC}		±0.10	±5	µA
C_I	Input capacitance			5		pF
V_{OH}	High-level output voltage	V_{CC} = 5.0 V, I_{OH} = – 1.0 mA	2.5	4.5		V
		V_{CC} = 5.0 V, I_{OH} = – 0.3 mA	4.5	4.8		V
V_{OL}	Low level output voltage	V_{CC} = 5.0 V, I_{OL} = 1.7 mA		0.3	0.4	V
I_{OH}	Output source current	V_{CC} = 5.0 V, V_{OH} = 4.5 V	– 0.3	– 1.4		µA
		V_{CC} = 4.0 V, V_{OH} = 3.5 V	– 0.2	–1.0		mA
		V_{CC} = 3.0 V, V_{OH} = 2.5 V	– 0.1	– 0.7		mA
		V_{CC} = 2.5 V, V_{OH} = 2.0 V	– 0.05	– 0.3		mA
		V_{CC} = 5.0 V, V_{OH} = 2.5 V	– 1.0	– 5.0		mA
I_{OL}	Output sink current	V_{CC} = 5.0 V, V_{OL} = 0.4 V	1.7	2.8		mA
		V_{CC} = 4.0 V, V_{OL} = 0.4 V	1.2	2.4		mA
		V_{CC} = 3.0 V, V_{OL} = 0.4 V	0.7	2.0		mA
		V_{CC} = 2.5 V, V_{OL} = 0.4 V	0.2	1.3		mA

Figure 4–36. Output Loading Circuit for Test

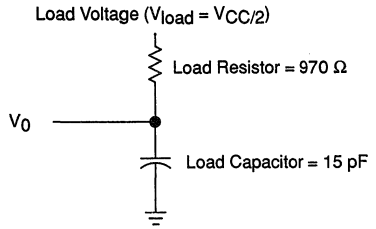


Table 4–52. Supply Current Requirements

Parameter		Test Conditions		Min	Typ	Max	Unit
I_{CC}	Operating mode	f_{osc} = 7.0 MHz,	V_{CC} = 5.0 V		17	24.5	mA
		f_{osc} = 3.0 MHz,	V_{CC} = 5.0 V		7.2	10.5	mA
		f_{osc} = 0.5 MHz,	V_{CC} = 5.0 V		1.2	1.8	mA
		f_{osc} = Z MHz,	V_{CC} = 5.0 V		2.4	3.5	mA/MHz
		f_{osc} = 0.5 MHz,	V_{CC} = 2.5 V		0.4	1.2	mA
I_{CC}	Wake-up mode 1 (one timer and UART active)	f_{osc} = 7.0 MHz,	V_{CC} = 5.0 V		2400	5600	µA
		f_{osc} = 3.0 MHz,	V_{CC} = 5.0 V		1200	3300	µA
		f_{osc} = 0.5 MHz,	V_{CC} = 5.0 V		250	800	µA
I_{CC}	Wake-up mode 2 (one timer active and UART inactive)	f_{osc} = 7.0 MHz,	V_{CC} = 5.0 V		960	3400	µA
		f_{osc} = 3.0 MHz,	V_{CC} = 5.0 V		480	2000	µA
		f_{osc} = 0.5 MHz,	V_{CC} = 5.0 V		140	550	µA
I_{CC}	Wake-up mode 3 (UART active only)	f_{osc} = 7.0 MHz,	V_{CC} = 5.0 V		1500	2400	µA
		f_{osc} = 3.0 MHz,	V_{CC} = 5.0 V		800	1500	µA
		f_{osc} = 0.5 MHz,	V_{CC} = 5.0 V		180	600	µA
I_{CC}	Halt OSC-On	f_{osc} = 7.0 MHz,	V_{CC} = 5.0 V		560	1280	µA
		f_{osc} = 3.0 MHz,	V_{CC} = 5.0 V		240	560	µA
		f_{osc} = 1.0 MHz,	V_{CC} = 5.0 V		80	200	µA
		f_{osc} = Z MHz				See Note 2	µA
I_{CC}	Halt OSC-Off				5	10	µA

Notes: 1) All inputs = V_{CC} or V_{SS} (except XTAL2). All I/O and output pins are open.
2) Maximum current = 180(Z) + 20 µA.

Chapter 5

The TMS7000 Assembler

TMS7000 assembly language instructions are mnemonic operation codes (or mnemonics) that correspond directly to binary machine instructions. An assembly language program (source program) must be converted to a machine language program (object program) by a process called *assembling* before a computer can execute it. Assembling converts the mnemonics to binary values and associates those values with binary addresses, creating machine language instructions. Assembler directives, discussed in Section 5.5, control this process, place data in the object program, and assign values to the symbols used in the object program.

TMS7000 assembly language is processed by a two-pass macro assembler that executes on a host computer. During the first pass the assembler:

1) Maintains the location counter,
2) Builds a symbol table, and
3) Produces a copy of the source code.

During the second pass the assembler:

1) Reads the copy of the source code and
2) Assembles the object code using the opcodes and symbol table produced during the first pass.

This section discusses the following topics:

Section		Page
5.1	Source Statement Format	5-2
5.2	Constants	5-6
5.3	Symbols	5-8
5.4	Expressions	5-10
5.5	Assembler Directives	5-13
5.6	Symbolic Addressing Techniques	5-49
5.7	Assembler Output	5-50
5.8	Object Code	5-55
5.9	Assembling Files – Examples	5-63

5.1 Source Statement Format

An assembly language source program consists of source statements that may contain assembler directives, machine instructions, pseudo-instructions, or comments. Source statements may contain four ordered fields — label, command, operand, and comment. Source statements that have an asterisk (*) in the first character position are comments and do not affect the assembly.

The syntax for source statements other than comment lines is:

```
[<label>]  <mnemonic>   [<operand>]   [<comment>]
```

where:

- The label and comments fields are optional.
- One or more blank spaces must separate each field.
- A statement must start with either a label or a blank space.

Figure 5–1 illustrates a typical example of TMS7000 source code. The action of the assembler is to produce a *listing file* which is a copy of the source file with three additional columns; a line number to allow error messages to be referenced to specific lines, the absolute program addresses assigned to each line, and the object code generated for each line. (Listing file for this example is shown in Figure 5–2.) The assembler also produces an *object file* which contains the hex object code which can be executed by the TMS7000. (The object file for this example is shown in Figure 5–4) An optional symbol cross-reference can also be produced during assembly, which is appended to the listing file. (The cross reference file for this example is shown in Figure 5–3.)

Figure 5–1. TMS7000 Source Code Example

```
      ******

*
*  FOR USE WITH XASM7 PC ASSEMBLER
*  this example is for use with a TMS7XCX2
*  it exercises ports A,B,C & D
*
        IDT     'EXAMPLE'    EMBED NAME 'EXAMPLE' IN OBJECT
*
LABEL   EQU     R25          )
APORT   EQU     P4           )   READABLE NAMES ASSIGNED TO
BPORT   EQU     P6           )   REGISTER AND PORT LOCATIONS
CPORT   EQU     P8           )
DPORT   EQU     P10          )
*
        AORG    >F806        ABSOLUTE START ADDRESS ASSIGNED
*
START   EQU     $            LABEL 'START' ASSIGNED TO BEGINNING
        MOVP    %>FF,P5      APORT SET TO ALL OUTPUTS
        MOVP    %>FF,P9      CPORT SET TO ALL OUTPUTS
        MOVP    %>FF,P11     DPORT SET TO ALL OUTPUTS
LOOP    CLR     LABEL        REGISTER 'LABEL' CLEARED
        CALL    @OUT         OUTPUT ROUTINE CALLED
        INV     LABEL        REGISTER 'LABEL' SET TO ALL 1's
        CALL    @OUT         OUTPUT ROUTINE CALLED
        JMP     LOOP         REPEAT FROM 'LOOP'
*
*  output routine ( called from main program )
*
OUT     MOV     LABEL,A      MOVE CONTENTS OF 'LABEL' INTO 'A'
        MOVP    A,APORT      OUTPUT ACCUMULATOR TO APORT PINS
        MOVP    A,BPORT      OUTPUT ACCUMULATOR TO BPORT PINS
        MOVP    A,CPORT      OUTPUT ACCUMULATOR TO CPORT PINS
        MOVP    A,DPORT      OUTPUT ACCUMULATOR TO DPORT PINS
        RETS                 RETURN TO MAIN ROUTINE
*
*  load reset vector with address of start of program
*
        AORG    >FFFE
        DATA    START
*
        END
```

Figure 5–2. TMS7000 Listing File Example

```
EXAMPLE     7000 FAMILY MACRO ASSEMBLER  PC3.1 88.005

0001              *
0002              *  FOR USE WITH XASM7 PC ASSEMBLER
0003              *  this example is for use with a TMS7XCX2
0004              *  it exercises ports A,B,C & D
0005              *
0006                       IDT     'EXAMPLE'
0007              *
0008        0019  LABEL    EQU     R25          )
0009        0004  APORT    EQU     P4           )   READABLE NAMES ASSIGNED TO
0010        0006  BPORT    EQU     P6           )   REGISTER AND PORT LOCATIONS
0011        0008  CPORT    EQU     P8           )
0012        000A  DPORT    EQU     P10          )
0013              *
0014 F806                  AORG    >F806        ABSOLUTE START ADDRESS ASSIGNED
0015              *
0016        F806  START    EQU     $            LABEL 'START' ASSIGNED TO BEGINNING
0017 F806   A2             MOVP    %>FF,P5      APORT SET TO ALL OUTPUTS
     F807   FF
     F808   05
0018 F809   A2             MOVP    %>FF,P9      CPORT SET TO ALL OUTPUTS
     F80A   FF
     F80B   09
0019 F80C   A2             MOVP    %>FF,P11     DPORT SET TO ALL OUTPUTS
     F80D   FF
     F80E   0B
0020 F80F   D5   LOOP      CLR     LABEL        REGISTER 'LABEL' CLEARED
     F810   19
0021 F811   8E             CALL    @OUT         OUTPUT ROUTINE CALLED
     F812 F81B
0022 F814   D4             INV     LABEL        REGISTER 'LABEL' SET TO ALL 1's
     F815   19
0023 F816   8E             CALL    @OUT         OUTPUT ROUTINE CALLED
     F817 F81B
0024 F819   E0             JMP     LOOP         REPEAT FROM 'LOOP'
     F81A   F4
0025              *
0026              *  output routine ( called from main program )
0027              *
0028 F81B   12   OUT       MOV     LABEL,A      MOVE CONTENTS OF 'LABEL' INTO 'A'
     F81C   19
0029 F81D   82             MOVP    A,APORT      OUTPUT ACCUMULATOR TO APORT PINS
     F81E   04
0030 F81F   82             MOVP    A,BPORT      OUTPUT ACCUMULATOR TO BPORT PINS
     F820   06
0031 F821   82             MOVP    A,CPORT      OUTPUT ACCUMULATOR TO CPORT PINS
     F822   08
0032 F823   82             MOVP    A,DPORT      OUTPUT ACCUMULATOR TO DPORT PINS
     F824   0A
0033 F825   0A             RETS                 RETURN TO MAIN ROUTINE
0034              *
0035              *  load reset vector with address of start of program
0036              *
0037 FFFE                  AORG    >FFFE
0038 FFFE F806             DATA    START
0039              *
0040                       END
NO ERRORS, NO WARNINGS
```

5.1.1 Label Field

The label field is optional for machine instructions and for many assembler directives. If it is not used, the first character position must contain a blank. The label begins in the first character position of the source statement and extends to the first blank. It contains a symbol of up to 6 alphanumeric characters; the first character must be a letter.

A source statement that contains only a label field is a valid statement. It assigns the current value of the location counter to the label, which is equivalent to the following directive statement:

```
<label>  EQU  $
```

5.1.2 Command Field

The command field begins after the blank that terminates the label field. It is terminated by one or more blanks and may not extend past the right margin. If the label is omitted, the command can start in the second character position. The command field can contain one of the following opcodes:

- Machine-instruction mnemonic
- User-defined instruction
- Assembler directive

5.1.3 Operand Field

The operand field begins following the blank that ends the command field. It may not extend past the right margin of the source record. The operand field may contain one or more constants or expressions (described in Section 5.2 and Section 5.4) separated by commas. It is terminated by one or more blanks.

5.1.4 Comment Field

The comment field begins after the blank that terminates the operand field (or the blank that terminates the command field, if there are no operands). The comment field can extend to the end of the source record, if required, and can contain any ASCII character including blanks. The comment field contents (up to the end of the input record) are listed in the assembly source listing but do not affect the assembly.

5.2 Constants

The assembler recognizes five types of constants, each internally maintained as a 16-bit quantity:

- Decimal integer constants
- Binary integer constants
- Hexadecimal integer constants
- Character constants
- Assembly-time constants

5.2.1 Decimal Integer Constants

Decimal integer constants are written as strings of decimal digits, ranging from –32,768 to +65,535. Positive decimal integer constants in the range 32,768 to 65,535 are considered negative when interpreted by functions needing 2's complement values.

These are valid decimal constants:

1000	Constant equal to 1000 or >3E8
–32768	Constant equal to –32768 or >8000
25	Constant equal to 25 or >19
65535	Constant equal to 65535 to >FFFF

5.2.2 Binary Integer Constants

Binary integer constants are written as strings of up to 16 binary digits (0/1) preceded by a question mark (?). If less than 16 digits are specified, the assembler right justifies the bits.

These are valid binary constants:

?00010011	Constant equal to 19 or >13
?0111111111111111	Constant equal to 32767 or >7FFF
?11110	Constant equal to 30 or >001E

5.2.3 Hexadecimal Integer Constants

Hexadecimal integer constants are written as strings of up to four hexadecimal digits preceded by a greater than sign (>). Hexadecimal digits include the decimal values 0 through 9 and the letters A through F.

These are valid hexadecimal constants:

>78	Constant equal to 120
>F	Constant equal to 15
>37AC	Constant equal to 14252

5.2.4 Character Constants

Character constants are written as strings of one or two alphabetic characters enclosed in single quotes. Two consecutive single quotes are required to represent a single quote in a character constant. The characters are represented internally as 8-bit ASCII characters. A character constant consisting of only two single quotes (no letter) is valid and is assigned the value >0000.

These are valid character constants:

'AB'	Represented internally as >4142
'C'	Represented internally as >43 or >0043
'N'	Represented internally as >4E or >004E
''D'	Represented internally as >2744

5.2.5 Assembly-Time Constants

Assembly-time constants are symbols assigned values by an EQU directive (see the EQU directive). The symbol value is determined at assembly time. It is considered to be absolute or relocatable according to the relocatability of the expression, not according to the relocatability of the location counter value. Absolute value symbols may be assigned values with expressions using any of the above constant types.

5.3 Symbols

Symbols are used in the label field and the operand field. A symbol is a string of alphanumeric characters (A—Z, 0—9, and $). The first character in a symbol must be A—Z or $. No character may be blank. When more than six characters are used in a symbol, the assembler prints all the characters, but only recognizes the first six characters during processing (the assembler also prints a symbol truncation warning). Therefore, the first six characters of a symbol should be unique. User-defined symbols are valid only during the assembly in which they are defined.

Symbols used in the label field become symbolic addresses. They are associated with locations in the program and must not be used in the label field of other statements. Mnemonic opcodes and assembler directive names may be used as valid user-defined symbols in the label field.

Symbols used in the operand field must be defined in the assembly, usually by appearing in the label field of a statement or in the operand field of a REF or SREF directive.

These are examples of valid symbols:

START
ADD
OPERATION

Each of these symbols will be assigned the value of the location where it appears in the label field. Note that the symbol OPERATION will be truncated to OPERAT.

5.3.1 Predefined Symbols

The dollar sign ($), register (Rn), and port (Pn) symbols are predefined. The dollar sign represents the current value of the location counter. Register and port symbols are in the form Rn and Pn, respectively, where n is a constant in the range 0—255. All registers and peripheral file addresses should be defined before they are used in instructions.

These are examples of valid predefined symbols:

$	The current location
R0	Register 0
P22	Peripheral Register 22

The symbol ST (Status Register) is reserved and may not be redefined.

5.3.2 Terms

Terms are used in the operand field of machine instructions and assembler directives. A term may be a binary, character, decimal or hexadecimal constant, an absolute assembly-time constant or a label having an absolute value.

5.3.3 Character Strings

Several assembler directives require character strings as operands. A character string is a string of characters enclosed in single quotes. Single quotes *within* a character string are represented by two consecutive single quotes. The maximum length of a string is defined for each directive that requires a character string. The characters are represented internally as 8-bit ASCII characters.

These are valid character strings:

'SAMPLE PROGRAM'
Defines a 14-character string, SAMPLE PROGRAM

'PLAN "C"' Defines an 8-character string, PLAN 'C'

'OPERATOR MESSAGE : PRESS START SWITCH'
Defines a 37-character string, OPERATOR MESSAGE : PRESS START SWITCH

5.4 Expressions

Expressions are used in the operand fields of assembler directives and machine instructions. An expression is a constant or symbol, a series of constants or symbols, or a series of constants and symbols separated by arithmetic operators. Each constant or symbol may be preceded by a unary minus sign (–), a unary plus sign (+), or the unary invert symbol (#). The # symbol causes the value of the logical complement of the following constant or symbol to be used. An expression may not contain embedded blanks. Symbols defined as external references may be operands of arithmetic instructions within certain limits, as described in subsection 5.4.1.

5.4.1 Arithmetic Operators in Expressions

The arithmetic operators used in expressions are:

+	Addition
–	Subtraction
×	Multiplication
/	Signed division
#	Logical not (inversion)

When the assembler evaluates an expression, it first negates symbols or constants preceded by a minus (–) sign and then performs arithmetic operations from left to right. The assembler does not assign precedence to any operation other than unary plus or unary minus. All operations are integer operations; any fractions produced by division are truncated.

For example, the expression 4+5*2 is evaluated as 18, not 14. The expression 7+1/2 is evaluated as 4; the expression 1/2+7 is evaluated as 7 (note truncation).

The assembler checks for overflow conditions when arithmetic operations are performed. It issues a warning message when an overflow occurs or when the sign of the result is not as expected in respect to the operands and the operation performed. Examples where a "VALUE TRUNCATED" message is given are:

```
-2*>4000      >FFFE+2      -1*>8001
>8000*2       ->8000-1     -2*>8000
```

When the immediate value is greater than >7F and you precede the value with %#, signifying immediate and unary negation operations, the assembler correctly calculates the value but issues an error message. Ignore the `EXPRESSION OUT OF BOUNDS` error message. (Note that this problem has been fixed in version 2.3 of the assembler.) The following example illustrates this condition.

```
TEST     TMS7000 MACRO ASSEMBER
PAGE 0001
0001                *
0002                *  DX-10 X-SUPPORT TEST SOFTWARE
0003                *
0004                       IDT     'TEST'
0005 F000                  AORG    >F000
0006 F000  52              MOV     %>10,B
     F001  10
0007 F002  0D              LDSP
0008 F003  01              IDLE
0009 F004  28              ADD     %#>40,A
     F005  BF
0010 F006  28              ADD     %#>7F,A
     F007  80
0011 F008  28              ADD     %#>80,A
     F009  7F
********EXPRESSION OUT OF BOUNDS
0012                       END
0001 ERROR, 0000 WARNINGS, LAST ERROR AT 0011
```

5.4.2 Logical Operands in Expressions

If a pound sign (#) precedes a number or an expression it is complemented. All other arithmetic operations have precedence over the logical not (#) operation, except where modified by parentheses.

5.4.3 Parentheses in Expressions

Use parentheses to alter the order of expression evaluation. Parenthetical expressions can be nested up to eight levels. The portion of an expression within the innermost parentheses is evaluated first, then the next innermost pair is evaluated, etc. When all parenthetical phrases have been evaluated, the expression is evaluated from left to right. Evaluation of parenthetical phrases at the same nesting level may be considered to be simultaneous.

This expression is evaluated as follows:

```
LAB1+((4+3)*7)
```

1) Add 4 to 3
2) Multiply 7 by 7
3) Add the value of LAB1 to 49

5.4.4 Well-Defined Expressions

Some assembler directives require well-defined expressions in operand fields. Well-defined expressions contain only symbols or assembly-time constants that are defined before they are encountered in the expression. The evaluation of a well-defined expression must be absolute. A well-defined expression must not contain a character constant.

5.4.5 Relocatable Symbols in Expressions

An expression that contains a relocatable symbol or relocatable constant immediately following a multiplication or division operator is illegal. When the re-

sult of evaluating an expression up to a multiplication or division operator is relocatable, the expression is illegal.

If the current value of an expression is relocatable with respect to one relocatable section, a symbol of another section may not be included until the value of the expression becomes absolute. Some examples of relocatable symbols used in expressions are:

BLUE+1	The sum of the value of symbol BLUE plus one.
GREEN-4	The result of subtracting four from the value of symbol GREEN.
2*16+RED	The sum of the value of symbol RED plus the product of two and 16.
440/2-RED	The result of dividing 440 by two and subtracting the value of symbol RED from the quotient. RED must be absolute.

Table 5–1 defines the relocatability of the result for each type of operator.

Table 5–1. Results of Operations on Absolute and Relocatable Items in Expressions

A	B	A+B	A–B	A×B	A/B
ABS	ABS	ABS	ABS	ABS	ABS(B<>0)
ABS	RELOC	RELOC	illegal	†	illegal
RELOC	ABS	RELOC	RELOC	‡	§
RELOC	RELOC	illegal	¶	illegal	illegal

† Illegal unless A equals zero or one. If A is one, the result is relocatable. If A is zero, the result is an absolute zero.
‡ Illegal unless B equals zero or one. If B is one, the result is relocatable. If B is zero, the result is an absolute zero.
§ Illegal unless B equals one. If B equals one, the result is relocatable.
¶ Illegal unless A and B are in the same relocatable segment. If A and B are in the same section, the result is absolute.

5.4.6 Externally Defined Symbols in Expressions

Externally defined symbols (defined in REF and SREF directives) are allowed in expressions under the following conditions:

1) Only one externally referenced symbol may be used in an expression.
2) The character preceding the referenced symbol must be a plus sign, a blank, or a comma (the @ sign is not considered). The portion of the expression preceding the symbol, if any, must be added to the symbol.
3) The portion of the expression following the referenced symbol must not include multiplication, division, or logical operations on the symbol (as for a relocatable symbol described in subsection 5.4.5).
4) The remainder of the expression following the referenced symbol must be absolute.

The assembler limits the total number of external referenced symbols to 255 per module. Modules using more than 255 external symbols must be broken into smaller modules for assembly and linked using the link editor.

5.5 Assembler Directives

Assembler directives control the assembly process. This section discusses the various categories of directives supported by the TMS7000 assembler and defines the directives in alphabetical order.

Directives that Affect the Location Counter

As the assembler reads program source statements it increments its location counter. The location counter contents correspond to the memory locations assigned to the resulting object code. Twelve directives, listed in Table 5–2 on page 5-14, affect the location counter. BES and BSS advance the location counter to provide a block of program memory for the object code. The EVEN directive ensures an even address word boundary. The remaining nine directives initialize the location counter and define its value as relocatable, absolute, or dummy.

Directives in this category include:

— AORG — CEND — DORG — PEND
— BES — CSEG — DSEG — PSEG
— BSS — DEND — EVEN — RORG

Directives that Affect Assembler Output

Directives that affect assembler output are mainly used to improve program usability. The IDT directive supplies a program identifier; the five other directives affect the source listing.

— IDT — PAGE
— LIST — TITL
— OPTION — UNL

Directives that Initialize Constants

These directives assign values to successive bytes or words of the object code (BYTE, DATA), place text characters in object code for display purposes (TEXT), or initialize constants to be used during the assembly (EQU).

— BYTE — EQU
— DATA — TEXT

Directives for Linking Programs

The link editor resolves externally referenced symbols and definitions. These directives help the link editor by identifying symbols and definitions that may be used or defined by another program module. This allows separate program modules to be assembled separately and integrated into an executable program.

— DEF — REF
— LOAD — SREF

Miscellaneous Directives

This category includes those assembler directives not applicable to the other categories:

— COPY
— END
— MLIB

Table 5–2. Summary of Assembler Directives

Directives that Affect the Location Counter		
Mnemonic	**Directive**	**Syntax**
AORG	Absolute origin	[<label>] AORG [<wd–exp> [<comment>]]
BES	Block ending with symbol	[<label>] BES <wd–exp> [<comment>]
BSS	Block starting with symbol	[<label>] BSS <wd–exp> [<comment>]
CEND	Common segment end	[<label>] CEND [<comment>]
CSEG	Common segment	[<label>] CSEG ['<string>' [<comment>]]
DEND	Data segment end	[<label>] DEND [<comment>]
DORG	Dummy origin	[<label>] DORG [<exp> [<comment>]]
DSEG	Data segment	[<label>] DSEG [<comment>]
EVEN	Even boundary	[<label>] EVEN [<comment>]
PEND	Program segment end	[<label>] PEND [<comment>]
PSEG	Program segment	[<label>] PSEG [<comment>]
RORG	Relocatable origin	[<label>] RORG [<exp> [<comment>]]
Directives that Affect Assembler Output		
Mnemonic	**Directive**	**Syntax**
IDT	Program identifier	[<label>] IDT '<string>' [<comment>]
LIST	Restart source listing	[<label>] LIST [<comment>]
OPTION	Output options	[<label>] OPTION <option list> [<comment>]
PAGE	Page eject	[<label>] PAGE [<comment>]
TITL	Page title	[<label>] TITL '<string>' [<comment>]
UNL	Stop source listing	[<label>] UNL [<comment>]
Directives that Initialize Constants		
Mnemonic	**Directive**	**Syntax**
BYTE	Initialize byte	[<label>] BYTE <exp>[,<exp>] [<comment>]
DATA	Initialize word	[<label>] DATA <exp>[,<exp>] [<comment>]
EQU	Define assembly-time constant	[<label>] EQU <exp> [<comment>]
TEXT	Initialize text	[<label>] TEXT [–]'<string>' [<comment>]

Table 5–2. Summary of Assembler Directives (Concluded)

Directives for Linking Programs		
Mnemonic	**Directive**	**Syntax**
DEF	External definition	[<label>] DEF <symbol>[,<symbol>] [<comment>]
LOAD	Force load	[<label>] LOAD <symbol>[,<symbol>] [<comment>]
REF	External reference	[<label>] REF <symbol>[,<symbol>] [<comment>]
SREF	Secondary external reference	[<label>] SREF <symbol>[,<symbol>] [<comment>]
Miscellaneous Directives		
Mnemonic	**Directive**	**Syntax**
COPY	Copy source file	[<label>] COPY <filename> [<comment>]
END	Program end	[<label>] END [<symbol> [<comment>]]
MLIB	Define macro library	[<label>] MLIB '<pathname>' [<comment>]

Syntax

[<label>] AORG [<wd–exp> [<comment>]]

Fields

Label Optional; if used, the label assumes the current value of the location counter.

Operand Optional; if used, the operand field must contain a well-defined expression (<wd–exp>).

Operand Optional; may only be used with the operand field.

Description

AORG loads the location counter with the first address of a segment of absolute code. This address is usually specified by the operand. If no operand is used, the value in the location counter equals the length of all preceding absolute code. When no AORG directive is entered, the object program does not include absolute addresses.

Example 1

```
AORG  >1000+X
```

Symbol X must be absolute and previously defined. If X has a value of 6, the location counter is set to >1006. If a label had been included, it would have been assigned the value >1006.

Avoid using AORG in object modules which will be linked. Linking a module that contains an AORG directive may produce an *Illegal immediate tag encountered* error at link time. Use the PSEG, CSEG, and DSEG directives instead to identify the locations in the source code. Use the PROGRAM, COMMON, and DATA commands in the link control file to define the locations.

The link control file will look similar to this example:

Example 2

```
TASK    MYPROG
PROGRAM >F006    Program starting point (PSEG)
DATA    >FFD0    Trap and vector table stg pt (DSEG)
COMMON           Additional starting location (CSEG)
INCLUDE FILE1
INCLUDE FILE2
END
```

Syntax [<label>] BES <wd–exp> [<comment>]

Fields **Label** Optional; if used, the label is assigned the value of the location following the block.

Operand Contains a well-defined expression that represents the number of bytes to be added to the location counter.

Comment Optional

Description BES increments the location counter by the operand value.

Example

```
BUFF2   BES  >10
```

A 16-byte buffer is reserved. If the location counter had contained >100 when the directive was processed, BUFF2 would have been assigned the value >110.

Syntax [<label>] BSS <wd–exp> [<comment>]

Fields

Label Optional; if used, a label is assigned the value of the location of the first byte in the block.

Operand Contains a well-defined expression that represents the number of bytes to be added to the location counter.

Comment Optional

Description

BSS increments the location counter by the operand value.

Avoid using the BSS directive for defining register names. Using BSS in this manner may produce a *Pass1/Pass2 operand conflict* error at assembly time. Use the EQU directive for defining register names.

Example

```
BUFF1   BSS  80  Card input buffer
```

An 80-byte buffer is reserved starting at location BUFF1.

Syntax [<label>] BYTE <exp>[,<exp>] [<comment>]

Fields

Label Optional; if used, the label is assigned the location where the assembler places the first byte.

Operand Contains one or more expressions separated by commas. These expressions cannot contain external references. The assembler evaluates each expression and places the value in a byte as an 8-bit number. If truncation is required, the assembler prints a truncation warning message and puts the 8 LSbs of the value in the byte.

Comment Optional

Description BYTE places one or more values in one or more successive bytes of memory.

Example

```
KONS BYTE >F+1,-1,'D'-'=',0,'AB'-'AA'
```

This example initializes five bytes, starting with a byte at location KONS. The contents of the resulting bytes are 00010000, 11111111, 00000111, 00000000, and 00000001.

Syntax [<label>] CEND [<comment>]

Fields

Label Optional; if used, the label is assigned the value of the location counter before modification.

Operand Not used

Comment Optional

Description

CEND terminates the definition of a block of common-relocatable code by placing a value in the location counter and defining succeeding locations as program-relocatable. The location counter is set to one of the following values:

- The maximum value the location counter has ever attained by assembling any preceding block of program-relocatable code.
- Zero, if no program-relocatable code was previously assembled.

If encountered in data- or program-relocatable code, this directive functions as a DEND or PEND. CEND is invalid when used in absolute code.

Syntax [<label>] COPY <filename> [<comment>]

Fields

Label Optional

Operand Names a file that source statements are read from. The file name may be:

- An access name recognized by the operating system
- A synonym form of an access name

Comment Optional

Description

COPY changes the source input for the assembler. A COPY directive may be placed in a file being copied. Nested copying of files can be performed by placing a COPY directive in a file being copied. The assembler limits such nesting to eight levels; the host operating system may place additional restrictions on nesting capabilities.

Example

```
COPY   SFILE
```

This example causes the assembler to take its source statements from a file SFILE. At the end-of-file for SFILE, the assembler resumes processing source statements from the file or device previous to the COPY directive.

Note:

If a source file is to be copied into a main source file intended for assembly, ensure that it **does not** contain an END statement or the assembler will respond to it by terminating prematurely.

Syntax [<label>] CSEG ['<string>'[,<exp>] [<comment>]]

Fields

Label Optional; if used, the label is assigned the value placed in the location counter.

Operand Optional (see preceding Description).

Comment Optional; may only be used with the operand field.

Description

CSEG begins or continues a common-relocatable segment (relocatable with respect to a common segment) at the address in the location counter. If the operand is not used, the CSEG directive defines the beginning of (or continuation of) the blank common segment of the program.

When used, the operand field contains a character string of up to six characters enclosed in quotes. (The assembler truncates strings that are longer than six characters and prints a truncation error message.) If this string did not previously appear as the operand of a CSEG directive, the assembler:

1) Associates a new relocation section number with the operand,
2) Sets the location counter to zero, and
3) Defines succeeding locations as relocatable with respect to the new relocatable section.

If the operand string was previously used in a CSEG, the succeeding code represents a continuation of the particular common segment associated with the operand. The location counter is restored to the maximum value attained during the previous assembly of any portion of that particular common segment. The second operand, <exp>, specifies the memory alignment for the beginning of the section.

Common-relocatable code is normally terminated by a CEND directive, but can also be terminated by the PSEG, DSEG, AORG, and END directives. The CEND and PSEG directives define succeeding locations as program-relocatable. The DSEG and AORG directives terminate the common segment by beginning a data or an absolute segment. The END directive terminates the common segment and the program.

The CSEG directive permits construction and definition of independently relocatable segments of data that several programs can access or reference at execution time. Information placed in the object code by the assembler permits the link editor to relocate all common segments independently and make appropriate adjustments to all addresses that reference locations within common segments. Locations within a common segment may be referenced by several different programs if each program contains a CSEG directive with the same operand or no operand.

Example

```
COM1A   CSEG    'ONE'
          .
          .
<Common-relocatable section, type 'ONE'>
          .
          .
        CEND
          .
COM2A   CSEG    'TWO'
          .
          .
<Common-relocatable section, type 'TWO'>
          .
          .
COM2B   CEND
          .
COM1C   CSEG   'ONE'
          .
          .
<Common-relocatable section, type 'ONE'>
          .
          .
COM1B   CEND
          .
COM1L   DATA COM1B-COM1A  LENGTH OF SEGMENT 'ONE'
COM2L   DATA COM2B-COM2A  LENGTH OF SEGMENT 'TWO'
```

The three blocks of code between the CSEG and CEND directives are common-relocatable.

The first and third blocks are relocatable with respect to one common relocation type; the second is relocatable with respect to another. The first and third blocks comprise the common segment 'ONE'; the value of the symbol COM1L is the length in bytes of this segment.

The symbol COM2A is the symbolic address of the first word of the first word of common segment 'TWO'; COM2B is the common-relocatable (type 'TWO') byte address of the location following the segment. (Note that the symbols COM2B and COM1C are of different relocation types and possibly different values.) The value of the symbol COM2L is the length in bytes of common segment 'TWO'.

Syntax [<label>] DATA <exp>[,<exp>] [<comment>]

Fields

Label Optional; if used, the label is assigned the location where the assembler places the first word.

Operand Contains one or more expressions separated by commas. The assembler evaluates each expression and places the value in a word as a 16-bit number. Words are stored most significant byte first, that is, at the lower address.

Comment Optional

Description DATA places one or more values in one or more successive 2-byte words of memory.

Example

```
KONS1 DATA 3200,1+'AB',-'AF',>F4A0,'A'
```

This example initializes five words, starting with a word at location KONS1. The contents of the resulting words are >0C80, >4143, >BEBA, >F4A0, and >0041.

In a DATA directive statement with an operand of multiple fields, the assembled value of the location counter symbol ($) will **not** be correctly calculated if the $ is not in the first field (that is, a correct value will be calculated for $ if it is in the first field of the DATA statement.) The following example shows both cases. This example is for assembler revision 2.1.

Example

```
TEST    7000 FAMILY MACRO ASSEMBLER  DX2.1  83.074  15:23:38     7/25/84
                                                                 PAGE 0001
 0001                ************************************
 0002                * This is an example which produces   *
 0003                * correct values for $.               *
 0004                ************************************
 0005  0000
 0006                IDT   'TEST $'
 0007  0000  0009    DATA  9
 0008  0002  0008    DATA  8
 0009  0004  0004'   DATA  $          CORRECT VALUE FOR $
 0010  0006  0006'   DATA  $,8,9      CORRECT VALUE FOR $
       0008  0008
       000A  0009
 0011  000C  000F'   DATA  $+3,7+1    CORRECT VALUE FOR $
       000E  0008
 0012  0010  0008    DATA  7+1
 0013  0012  0015'   DATA  $+3        CORRECT VALUE FOR $
 0014  0014
 0015                ************************************
 0016                * This is an example which produces   *
 0017                * incorrect values for $              *
 0018                ************************************
 0019  0014
 0020  0014  0009    DATA  9,8,$      INCORRECT VALUE FOR $
       0016  0008
       0018  0014'
 0021  001A  0008    DATA  7+1,$+3    INCORRECT VALUE FOR $
       001C  001D'
 0022                END
NO ERRORS, NO WARNINGS
```

Syntax [<label>] DEF <symbol>[,<symbol>] [<comment>]

Fields

Label Optional; if used, the label assumes the current value of the location counter.

Operand Contains one or more symbols, separated by commas, to be defined in the program being assembled.

Comment Optional

Description DEF makes one or more symbols available to other programs for reference. All symbols used in the DEF statement must be defined in the same module.

Example

```
DEF  ENTER,ANS
```

This example causes the assembler to include symbols ENTER and ANS in the object code; these symbols are available to other programs.

Syntax [<data>] DEND [<comment>]

Fields

Label Optional; if used, the label is assigned the value of the location counter before modification.

Operand Not used

Comment Optional

Description

DEND terminates a block of data-relocatable code and defines succeeding locations as program-relocatable. One of two values is placed in the location counter:

1) The maximum value attained by the location counter as a result of assembling the preceding block of program-relocatable code
2) Zero, if no program-relocatable code was previously assembled.

If encountered in common-relocatable or program-relocatable code, DEND functions as a CEND or PEND, and the assembler issues a warning message. Like CEND and PEND, DEND is invalid in absolute code.

Syntax [<label>] DORG [<exp> [<comment>]]

Fields

Label Optional; when used, the label is assigned the same value that is placed in the location counter.

Operand Optional; when used, it contains an expression <exp> that can be either absolute or relocatable. Any symbol in the expression must have been previously defined.

When the operand field is absolute, the location counter is assigned the absolute value. When the operand is relocatable, the location counter is assigned the relocatable value and the same relocation type as the operand. When this occurs, space is reserved in the section that has that relocation type.

Comment Optional

Description

DORG loads the location counter with the beginning address of a dummy block or section. This address is specified by the operand. The assembler does not generate code for a dummy section, but operates normally in all other respects. The symbols that describe the dummy section layout are available when the remainder of the program is assembled.

Example 1

```
DORG  0
```

The assembler assigns values relative to the start of the dummy section to the labels within the dummy section. This example is appropriate for defining a data structure. The executable portion of the module (following the RORG directive) should use the labels of the dummy section as relative addresses. In this manner, the data is available to the procedure regardless of the memory area into which the data is loaded.

Example 2

```
DORG  0
 .
 .
 .     (code as desired)
 .
DORG  $
  .
  .
  .     (data segment)
  .
END
  .
```

This is appropriate for the executable portion (procedure division) of a procedure that is common to more than one task. The code corresponding to the dummy section must be assembled in another program module. In this manner, separate data portions (dummy sections) are available to the procedure portion.

The DORG directive may also be used with data-relocatable or common-relocatable operands to specify dummy data or common segments.

Example 3

```
CSEG  'COM1'

        DORG  $     "$" has a common-relocatable
                     value
         .
         .
         .
LAB1    DATA  $
MASK    DATA  >F000
         .
         .
         .
        CEND
```

In this example, no object code is generated to initialize the common segment COM1, but space is reserved and all common-relocatable labels describing the structure of the common block (including LAB1 and MASK) are available for use throughout the program.

Syntax [<label>] DSEG [<comment>]

Fields **Label** Optional; if used, the label is assigned the data-relocatable value placed in the location counter.

Operand Not used

Comment Optional

Description DSEG begins a block of data-relocatable code at the address in the location counter. Data-relocatable blocks comprise the data segment of a program. The data segment can be relocated independently of the program segment at link-edit time. This separates modifiable data from executable code.

A data-relocatable block is normally terminated by a DEND directive. It can also be terminated by a PSEG, CSEG, AORG, or END directive. The PSEG and DEND directives identify succeeding locations as program-relocatable. The CSEG and AORG directives terminate the data segment by beginning a common or an absolute segment, respectively. The END directive terminates the data segment and the program.

The location counter is initially set to zero.

Example 1

```
RAM     DSEG            Start of data area

  .
  .
  .
 <Data-relocatable code>
  .
  .
  .
  .
ERAM    DEND
LRAM    EQU    ERAM-RAM
```

The block of code between the DSEG and DEND directives is data-relocatable. RAM is the symbolic address of the first word of this block; ERAM is the data-relocatable byte address of the location following the code block. The value of the symbol LRAM is the length in bytes of the block.

Syntax [<label>] END [<symbol> [<comment>]]

Fields

Label Optional; if used, the label assumes the current value of the location counter.

Operand Optional; when used, the operand contains a program-relocatable or absolute symbol that specifies the program entry-point. If the operand is not used, no entry point is placed in the object code.

Comment Optional; may only be used with the operand field.

Description END terminates the assembly. It should be **the last source statement of a program.** Any source statements following the END directive are considered part of the next assembly.

Example

```
END   START
```

This example terminates program assembly. The assembler also places the value of START in the object code as an entry point.

Syntax <label> EQU <exp> [<comment>]

Fields

Label A symbol that will be assigned the operand's value.

Operand An expression whose value is assigned to the label.

Comment Optional

Description EQU assigns a value to a symbol.

> **Note:**
>
> <exp> may not contain a REF'd symbol or forward references.

Example 1

```
SUM     EQU     R5
```

This example assigns an absolute value to the symbol SUM, making SUM available to use as a register address. A register should always be defined before it is used.

Example 2

```
TIME    EQU     HOURS
```

This example assigns the value of the previously defined symbol HOURS to the symbol TIME. When HOURS appears in the label field of a machine instruction in a relocatable block of the program, the value is a relocatable value. The two symbols may be used interchangeably. Symbols in the operand field must be previously defined.

Syntax [<label>] EVEN [<comment>]

Fields **Label** Optional; if used, the label is assigned the value in the location counter after the directive is processed.

Operand Not used

Comment Optional

Description EVEN places the location counter on the next word boundary (even byte address). When the location counter is already on an even boundary, the location counter is not altered.

Example

```
WRF1    EVEN
```

Assures that the location counter contains an even boundary address and assigns the location counter address to label WRF1.

Syntax [<label>] IDT '<string>' [<comment>]

Fields

Label Optional; if used, the label is assigned the current value of the location counter.

Operand Contains the module name <string>, a character string of up to eight characters enclosed in single quotes. The assembler truncates strings that are longer than eight characters and prints a truncation error message.

Comment Optional

Description IDT assigns a name to the object module produced.

Example

```
IDT  'CONVERT'
```

This example assigns the name CONVERT to the module being assembled. The module name is printed in the source listing as the operand of the IDT directive and appears in the page heading of the source listing. The module name is also placed in the object code and is used by the link editor for automatic entry-point resolution. A routine whose entry point is to be automatically resolved by the link editor must be declared as the 'string' on the IDT statement for that module. The entry point must also be REF'd in this case.

Note:

Although the assembler accepts lowercase letters and special characters within the quotes, ROM loaders (for example) will not. Therefore, only uppercase letters and numerals are recommended.

Syntax [<label>] LIST [<comment>]

Fields

Label Optional; if used, the label assumes the current value of the location counter.

Operand Not used

Comment Optional; if used, the assembler does not print the comment.

Description LIST restores printing of the source listing after it was cancelled by a UNL directive. This directive is not printed in the source listing, but the line counter increments.

Syntax [<label>] LOAD <symbol>[,<symbol>] [<comment>]

Fields

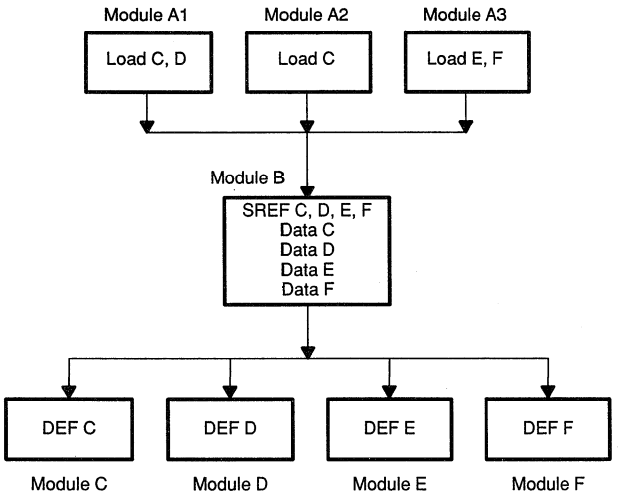
Label Optional

Operand Contains one or more symbols, separated by commas, to be used in the operand field of a subsequent source statement.

Comment Optional

Description The LOAD directive is like a REF, but the symbol does not need to be used in the module containing the LOAD. The symbol used in the LOAD must be defined in some other module. LOADs are used with SREFs. If one-to-one matching of LOAD and DEF symbols does not occur, then unresolved references will occur during link editing.

Example 5–1.



- Module A1 uses a branch table in module B to obtain one module C, D, E, or F.
- Module A1 knows which of module C, D, E, and F it requires.
- Module B has an SREF for C, D, E, and F.
- Module C has a DEF for C.
- Module D has a DEF for D.
- Module E has a DEF for E.
- Module F has a DEF for F.
- Module A1 has a LOAD for the modules C and D it needs.
- Module A2 has a LOAD for the module C it needs.
- Module A3 has a LOAD for the modules E and F it needs.

The LOAD and SREF directives permit module B to be written to handle a highly involved case and still be linked together without unnecessary modules since A1 only has LOAD directives for the modules it needs.

When a link edit is performed, automatic symbol resolutions will pull in the modules appearing in the LOAD directives.

If the link control file included A1 and A2, modules C and D would be pulled in while modules E and F would not be pulled in. If the link control file included A3, modules E and F would be pulled in while modules C and D would not be pulled in. If the link control file included A2, module C would be pulled in while modules D, E, and F would not be pulled in.

Syntax [<label>] MLIB '<pathname>' [<comment>]

Fields

Label Optional; if used, the label assumes the current value of the location counter.

Operand Contains the pathname, a character string of up to 48 characters enclosed in single quotes. Longer strings produce truncation error messages.

Comment Optional

Description

The MLIB directive provides the assembler with the name of a library containing macro definitions. The operand is a directory pathname (constructed according to the host operating system conventions) enclosed in single quotes (see IDT and TITL directives). This directive is defined only for hosts that support libraries on hard disks.

Note:

Neither the assembler nor its runtime support have access to the operating system's synonym table, and so cannot expand pathnames. The use of synonyms prevents finding any macros in that library.

Example 1

```
MLIB 'MYVOLUME.MACDIR.CMPXMACS.NEWMACS'
MLIB 'USER32.BIGPROJ.MYTASK.MACROS'
```

This example causes the macro function, when the program finds a macro call SUBMAC (not previously defined), to search first for a file named

`USER32.-BIGPROJ.-MYTASK.-MACROS.SUBMAC,`

and then if that file isn't found, to search for a file named

`MYVOLUME.-MACDIR.-CMPXMACS.-NEWMACS.SUBMAC,` in that order.

On a VAX/VMS system, a pathname would be specified as follows:

```
MLIB 'DRCO:[MOORE.ASM32]'
```

Example 2

The following program segment illustrates macro library use for an MS/PC-DOS system.

```
MLIB 'E:'    Pathname must be a drive name
  .          Typical assembly code
  .
XMAC         First macro call
  .
YMAC         Another macro call
  .
END
```

The assembler will search the drive specified by the MLIB directive for a file with the same name as the macro. The macro name cannot have an extension. Only one macro is allowed per file.

Syntax OPTION <option-list>

Fields

Label Not used

Operand <option-list> (see preceding Description)

Comment Not used

Description

OPTION selects several options for the assembler listing output. The <option-list> operand is a list of keywords separated by commas. Each keyword selects one of the following listing features:

BUNLST:	Limit the listing of BYTE directives to one line
DUNLST:	Limit the listing of DATA directives to one line
TUNLST:	Limit the listing of TEXT directives to one line
FUNLST:	Turn off all unlist options
XREF:	Produce a symbol cross-reference listing
NOLIST:	Inhibit all listing output (this overrides the LIST directive)
SYMLST:	Produce a symbol listing in the object file, no symbols are put in the listing file

Syntax [<page>] PAGE [<comment>]

Fields

Label Optional; if used, the label assumes the current value of the location counter.

Operand Not used

Comment Optional; if used, the assembler does not print the comment.

Description PAGE prints the source program listing on a new page. The PAGE directive is not printed in the source listing, but the line counter increments.

Example

```
PAGE
```

The assembler begins a new page of the source listing. The next source statement is the first statement listed on the new page. Using the PAGE directive to separate source listing into logical divisions improves program documentation.

Syntax [<label>] PEND [<comment>]

Fields

Label Optional; if used, the label is assigned the value of the location counter before modification.

Operand Not used

Comment Optional

Description

The PEND directive is the program-segment counterpart of the DEND and CEND directives. It begins a section of program-relocatable code at the address in the location counter. The value placed in the location counter is the maximum value it attained by assembling all preceding program-relocatable code. It is invalid when used in absolute code.

Syntax [<label>] PSEG [<comment>]

Fields

Label Optional; if used, the label is assigned the value placed in the location counter.

Operand Optional

Comment Optional

Description

PSEG begins a program-relocatable segment at the address in the location counter. The location counter is set to one of the following values:

- The maximum value the location counter has attained by assembling any preceding block of program-relocatable code.
- Zero, if no program-relocatable code was previously assembled.

The PSEG directive is the program-segment counterpart of the DSEG and CSEG directives. Together, the three directives provide a consistent method of defining the various types of relocatable segments. The following sequences of directives are functionally equivalent.

```
SEQUENCE 1                          SEQUENCE 2

DSEG                                DSEG
  .                                   .
  .                                   .
  .                                   .
<Data-relocatable code>             <Data-relocatable code>
  .                                   .
  .                                   .
  .                                   .
DEND                                  .
CSEG                                CSEG
  .                                   .
  .                                   .
  .                                   .
<Common-relocatable code>           <Common-relocatable code>
  .                                   .
  .                                   .
  .                                   .
CEND                                  .
PSEG                                PSEG
  .                                   .
  .                                   .
  .                                   .
<Program-relocatable code>          <Program-relocatable code>
  .                                   .
  .                                   .
PEND                                  .
  .                                   .
END                                 END
```

Syntax [<label>] REF <symbol>[,<symbol>] [<comment>]

Fields

Label Optional; if used, the label assumes the current value of the location counter.

Operand Contains one or more symbols, separated by commas, to be used in the operand field of a subsequent source statement.

Comment Optional

Description REF provides access to one or more symbols defined in other programs. If a symbol is listed in the REF statement, then a corresponding symbol must also be present in a DEF statement in another source module. If the symbol is not defined in another module, then an error occurs at link edit time. The system generates a summary list of all unresolved references.

Example

```
REF  ARG1,ARG2
```

This example causes the assembler to include symbols ARG1 and ARG2 in the object code so that the corresponding addresses may be obtained from other programs.

Syntax [<label>] RORG [<exp> [<comment>]]

Fields

Label Optional; if used, the label is assigned the same value that is placed in the location counter.

Operand Optional; when used, the operand must be a relocatable expression (<exp>). It can only contain previously defined symbols.

Comment Optional; may only be used with the operand field.

Description

RORG places a value in the location counter. If encountered in absolute code, RORG also defines succeeding locations as program-relocatable. The operand usually specifies the value placed in the location counter. If the operand is not used, the location counter is replaced by:

- The *current maximum length of the program segment* of the program, if RORG appears in absolute or program-relocatable code.
- The *maximum length of the data segment* if RORG appears in data-relocatable code.
- The *maximum length of the common segment* if RORG appears in common-relocatable code.

The length of the program-, data-, or common-relocatable segment, at any time during assembly, is determined by either of the following:

1) The maximum value the location counter has ever attained as a result of the assembly of any preceding block of relocatable code.
2) Zero, if no relocatable code has been previously assembled.

Since the location counter begins at zero, the length of a segment and the next available address within that segment are identical.

If RORG appears in absolute code, a relocatable operand must be program-relocatable. In relocatable code, the operand's relocation type (data, common, or program) must match that of the current location counter.

In absolute code RORG places the operand value in the location counter and changes the location counter's relocation type to program-relocatable. In relocatable code RORG places the operand value in the location counter but does not change the location counter's relocation type.

Example 1

```
RORG  $-10      Overlay ten bytes
```

The $ symbol contains the value of the current location. This example sets the location counter to the current location less ten bytes. The instructions and directives following the RORG directive replace the ten previously assembled words of relocatable code, permitting correction of the program without removing source records. If a label had been included, the label would have been assigned the value placed in the location counter.

Example 2

```
SEG2    RORG
```

The location counter contents depend upon preceding source statements. Assume that after defining data for a program that occupies >44 bytes, an AORG directive initiates an absolute block of code. The absolute block is followed by the RORG directive from the preceding example. This places >0044 in the location counter and defines the location counter as relocatable. Symbol SEG2 is a relocatable value, >0044. The RORG directive from the above example would have no effect except at the end of an absolute block or a dummy block.

Syntax [<label>] SREF <symbol>[,<symbol>] [<comment>]

Fields

Label Optional; if used, the label assumes the current value of the location counter.

Operand Contains one or more symbols, separated by commas, to be used in the operand field of a subsequent source statement.

Comment Optional

Description SREF provides access to one or more symbols defined in other programs. Unlike REF, SREF does not require a symbol to have a corresponding symbol listed in a DEF statement of another source module. The SREF'd symbol will be an unresolved reference but no error message will be produced.

Example

```
SREF    ARG1,ARG2
```

This example causes the link editor to include symbols ARG1 and ARG2 in the object code so that the corresponding addresses may be obtained from other programs.

Syntax [<label>] TEXT [–]'<string>' [<comment>]

Fields

Label Optional; if used, the label is assigned the location where the assembler places the first character.

Operand Contains a character string of up to 52 characters enclosed in single quotes; it may be preceded by a unary minus sign.

Comment Optional

Description TEXT places one or more characters in successive bytes of memory. The assembler negates the last character of the string when the string is preceded by a minus (–) sign (unary minus).

Example 1

```
MSG1   TEXT  'EXAMPLE'  Message heading
```

This example places the 8-bit ASCII representations of the characters in successive bytes. When the location counter is on an even address, the result is >4558, >414D, >504C, and >45xx. >xx, the contents of the rightmost byte of the fourth word, are determined by the next source statement. The label MSG1 is assigned the value of the first byte address, containing >45.

Example 2

```
MSG2  TEXT  -'NUMBER'
```

When the location counter is on an even address, the result is >4E55, >4D42, and >45AE. The label MSG2 is assigned the value of the byte address in which >4E is placed.

Syntax [<label>] TITL '<string>' [<comment>]

Fields

Label Optional; if used, the label assumes the current value of the location counter.

Operand Contains the title (<string>), a character string of up to 50 characters enclosed in single quotes. The assembler truncates a string longer than 50 characters and prints a truncation error message.

Comment Optional; the assembler does not print the comment but does increment the line counter.

Description

TITL supplies a title to be printed in the heading of each page of the source listing. The title is printed on the next page after TITL is processed, and on subsequent pages until another TITL directive is processed. The TITL directive must be the first source statement submitted to the assembler if a title heading is desired on the listing's first page. This directive is not printed in the source listing.

Example

```
TITL  '**REPORT GENERATOR**'
```

This example prints the title **REPORT GENERATOR** in the page headings of the source listing.

Syntax [<label>] UNL [<comment>]

Fields

Label Optional; if used, the label assumes the value of the location counter.

Operand Not used

Comment Optional; if used, the assembler does not print the comment.

Description

UNL halts the source listing output until a LIST directive is processed. It is not printed in the source listing, but the source line counter is incremented. This directive is frequently used in MACRO definitions to inhibit the listing of the macro expansion. It is useful for reducing assembly time and the size of the source listing.

5.6 Symbolic Addressing Techniques

The assembler processes symbolic memory addresses for addressing registers.

The following example illustrates this type of coding:

```
SUM         EQU     R33             Assign SUM for
                                    register 33
QUAN        EQU     R34             Assign QUAN for
                                    register 34

            ADD     QUAN,SUM        Add QUAN to SUM
*                                   Store in SUM
              .
              .
```

The two initial EQU directives assign meaningful labels to be used as register addresses in the subroutine.

5.7 Assembler Output

This section discusses assembler output, including source listings, error messages, a cross reference listing, and object code.

5.7.1 Source Listing

A source listing shows source statements and the object code they produce.

Each page of the source listing has a title line at the top. Any title supplied by a TITL directive is printed on this line. A page number is printed to the right of the title. A blank line follows the title line; subsequent lines contain the assembled source statements. Each assembled source statement contains a source statement number, a program counter value, the object code assembled, and the source statement as entered. If a source statement produces more than one byte of object code, the assembler prints the program counter value and object code on a separate line for each additional byte. Each added line is printed following the source statement line.

```
1       2       3       4       5
0018    F156    42      MOV     R10,R5
        F157    0A
        F158    05
```

The source statement number, 0018 in the example, is a 4-digit decimal number. Source records are numbered in the order in which they are entered including those source records that are not printed in the listing (TITL, LIST, UNL, and PAGE directives are not listed; source records between a UNL directive and a LIST directive are not listed). The difference between two consecutive source record numbers indicates if a source record was entered but not listed.

The next field on a line of the listing contains the program counter value (hexadecimal). In the example, F156 is the program counter value. Not all directives affect the program counter; the field is blank for those directives that do not affect it (the IDT, REF, DEF, EQU, SREF, and END directives leave the program counter field blank).

The third field normally contains a single blank. However, the assembler places a dash in this field when warning errors are detected.

The fourth field contains the hexadecimal representation of the object code, 420A05 in the preceding example. Note that the assembler produces a line containing the program counter value and the assembled object code for each byte of object code. All machine instructions and the BYTE, DATA, and TEXT directives use this field for object code. The EQU directive places the value corresponding to the label in the object code field.

The fifth field contains the characters of the source statement as they were scanned by the assembler. Spacing in this field is determined by the spacing

in the source statement. The four source statement fields will be aligned in the listing only when they are aligned in the source statements or when tab characters are used.

5.7.2 Normal Completion Error Messages

The assembler issues two types of error messages: normal completion messages and abnormal completion messages (subsection 5.7.3). When the assembler completes an assembly, it indicates any errors it encounters in the assembly listing. The assembler indicates errors following the source line in which they occur. At the end of a module (IDT–END pair), the corresponding messages are printed.

Table 5–3 lists error, warning, and information messages.

Table 5–3. Assembly Listing Errors

Nonfatal Errors	
Message	**Explanation/Response**

WARNING — 'CEND' ASSUMED
WARNING — 'DEND' ASSUMED
WARNING — 'PEND' ASSUMED
WARNING — 'DSEG' ASSUMED
	This is a warning that the following two statements will produce:
	`CSEG 'DATA'`
	`DSEG:`

WARNING — SYMBOL TRUNCATED	The maximum length for a symbol is six characters.
WARNING — STRING TRUNCATED	Check the syntax for the directive in question to determine the maximum length for the string.
WARNING — TRAILING OPERAND(S)	
WARNING — BYTE VALUE TRUNCATED	A value that is to be used as a byte value was larger than can be loaded into a byte.
*LAST WARNING	
Fatal Errors	
Message	**Explanation/Response**
ABSOLUTE VALUE REQUIRED DISPLACEMENT TOO BIG	An instruction with an operand with a fixed upper limit was encountered that overflowed this limit.
INVALID EXPRESSION	This may indicate invalid use of a relocatable symbol in arithmetic.
EXPRESSION OUT OF BOUNDS	There is a range limit for the value being used that was exceeded.
DUPLICATE DEFINITION	The symbol appears as an operand of a REF statement, as well as in the label field of the source, OR, the symbol appears more than once in the label field of the source.
INVALID RELOCATION TYPE	The type of variable isn't relocatable.
INVALID OPCODE	The second field of the source record contained an entry that is not a defined instruction, directive, pseudo-op, DXOP, DFOP, or macro name.

Table 5–3. Assembly Listing Errors (Continued)

Fatal Errors (Continued)	
Message	**Explanation/Response**
INVALID OPTION	The option given in the OPTION directive are invalid.
INVALID REGISTER VALUE	The given register value is too large or too small.
INVALID SYMBOL	The symbol being used has invalid characters in it.
VALUE TRUNCATED	The value used was too big for the field, so it has been truncated.
SYMBOL USED IN BOTH REF AND DEF	Symbol cannot be both referenced and defined in the same module.
COPY FILE OPEN ERROR	File does not exist or is already being used.
EXPRESSION SYNTAX ERROR	Unbalanced parentheses OR invalid operations on relocatable symbols.
INVALID ABSOLUTE CODE DIRECTIVE	The directive PEND, DEND and CEND have no meaning in absolute code.
LABEL REQUIRED BLANK MISSING	A blank is needed but one was not found. (Usually the blank is required in column 1.)
COMMA MISSING	Expected a comma but did not find one. Usually means that more operands were expected.
COPY FILENAME MISSING INDIRECT (*) MISSING	The indirect addressing (*) was needed.
SYMBOL REQUIRED OPERAND MISSING	There was no operand field.
REGISTER REQUIRED	A register should be used rather than a label or an absolute number.
CLOSE (') MISSING STRING REQUIRED	TEXT directive used with no text following.
PASS1/PASS2 OPERAND CONFLICT	The symbols in the symbol table did not have the same value in PASS1 and PASS2. Registers and peripheral files should be defined before they are used in an instruction. This error is also produced when the BSS directive is used to define a register name; use EQU instead.
SYNTAX ERROR	
UNDEFINED SYMBOL	The symbol being used has not been REF'ed or it has been DEF'ed but not used.
DIVIDE BY ZERO ILLEGAL SHIFT COUNT CANNOT INDEX BY REGISTER ZERO	The shift count being asked for is not valid.

Table 5–3. Assembly Listing Errors (Continued)

Information Messages	
Message	**Explanation/Response**
OPCODES REDEFINED	As a result of an MLIB directive, one or more assembler opcodes has been redefined by a MACRO within a MACRO directory. You should take action if this is not intended.
MACROS REDEFINED	As a result of an MLIB directive, one or more currently defined macros has been redefined by a MACRO (of the same name) with a MACRO directory. You should take action if this is not intended.

5.7.3 Abnormal Completion Error Messages

Most abnormal completion error messages are issued by the operating system under which the assembly runs (messages in this category include those concerned with file I/O errors). Refer to the applicable operating system reference manual for detailed information. Table 5–4 lists the abnormal error messages.

Table 5–4. Abnormal Completion Error Messages

UNEXPECTED END OF PARSE
ERROR MAPPING PARSE — ASSEMBLER BUG
INVALID OPERATION ENCOUNTERED
NO OPCODE
INVALID LISTING ERROR ENCOUNTERED
SYMBOL TABLE ERROR
INVALID LIB COMMAND ID
UNKNOWN ERROR PASSED, CODE = XXXX

5.7.4 Cross-Reference Listing

The assembler prints an optional cross-reference listing following the source listing, as specified by the assembler OPTION directive. The format of the listing is shown in Figure 5–3.

Figure 5–3. Cross-Reference Listing Format

```
EXAMPLE  7000 FAMILY MACRO ASSEMBLER  PC3.1 88.005

LABEL     VALUE     DEFN      REFERENCES
-----     -----     ----      ----------

APORT     0004      0011      0031
BPORT     0006      0012      0032
CPORT     0008      0013      0033
DPORT     000A      0014      0034
LABEL     0019      0010      0022  0024  0030
LOOP      F80F      0022      0026
OUT       F81B      0030      0023  0025
START     F806      0018      0040
```

As Figure 5–3 shows,

- The assembler prints each symbol defined or referenced in the assembly in the label column. If a single character follows the symbol, it represents the symbol attribute. These symbol-attribute characters and their meanings are listed in Table 5–5.
- The second (value) column contains a four-digit hexadecimal number, the value assigned to the symbol. The number of the statement that defines the symbol appears in
- The third (definition) column. This column is left blank for undefined symbols.
- The fourth (reference) column lists the source statement numbers that reference the symbol. A blank in this column indicates that the symbol was never used.

Table 5–5. Symbol Attributes

Character	Meaning
R	External reference (REF)
D	External definition (DEF)
U	Undefined
M	Macro name
S	Secondary reference (SREF)
L	Force load (LOAD)

5.8 Object Code

The assembler produces object code that may be linked to other code modules or programs, and loaded directly into an emulator, simulator, or EPROM programmer (to program a TMS77C82 for example). Object code consists of records containing up to 71 ASCII characters. You can correct record data manually for simple temporary changes for debugging. This prevents a lengthy re-assembly but it causes problems if you don't update the source. Figure 5–4 shows an example of object code.

Figure 5–4. Sample Object Code

This is the tagged object code generated from the source code of page 5–3.

```
K0000EXAMPLE 9F806BA2FFB05A2BFF09BA2FFB0BD5B198EBF81BBD419*8EBF81B7F01BFEXAMPLE1
BE0F4B1219B8204B8206B8208B820A*0A9FFFEBF8067F625F                      EXAMPLE2
:       EXAMPLE                          ASMMLP  PC3.1 88.005          EXAMPLE3
```

1) K — Begins each program
2) 0000 — Bytes of relocatable code, always 0 for final linked code
3) TESTPROG — Name from the IDT statement of the program
4) 9 — Address follows
5) F006 — Beginning address
6) B — 16-bit word follows
7) 327B — 16-bit word, MSB first
8) 7 — Checksum follows
9) F113 — Checksum (2's complement of the sum of all ASCII characters prior to and including the 7 tag)
10) F — End of line
11) * — 8-bit byte to follow
12) 8 — Ignore checksum – useful when object code patching
13) 1111 — Any 4 numbers can follow an 8 tag
14) 9 — Address follows
15) FFFE — Address of vector area
16) : — Last line of object module

Note: Table 5–6 provides an explanation of the tag characters.

5.8.1 Object Code Format

Formatted object code contains records made up of fields sandwiched between tag characters. The specific tag character, defined by the assembler or linker, specifies the function of the fields with which it is associated. A tag character occupies the first position on each line of object code and identifies the fields it precedes to the loader. Figure 5–4 details the various tag characters and their associated fields. Table 5–7 lists field and tag character information.

Table 5–6. Tag Characters

Tag Character	Description
K	Placed at the beginning of each program; followed by two fields. **Fields** – Field one contains the number of bytes of program relocatable code. – Field two contains the program identifier assigned to the program by an IDT directive. When no IDT directive is entered, field two is blank. The linker uses the program identifier to identify the program, and the number of bytes of program-relocatable code to determine the load bias for the next module or program.
M	Used when data or common segments are defined in the program; followed by three fields. **Fields** – Field one contains the length, in bytes, of data- or common-relocatable code. – Field two contains the data- or common-segment identifier, and field three contains a common number. The identifier is a six-character field containing the name $DATA (padded on the right by one blank) for data segments and $BLANK for blank common segments. If a named common segment appears in the program, an M tag will appear in the object code with an identifier field corresponding to the operand in the defining CSEG directive(s). – Field three consists of a four-character hexadecimal number defining a unique common number to be used by other tags that reference or initialize data of that particular segment. For data segments, this common number is always zero. For common segments (including blank common), the common numbers are assigned in increasing order, beginning at one and ending with the number of different common segments. The maximum number of common segments that a program may contain is 127.
1,2	Used with entry addresses. **Fields** – The associated field is used by the linker to determine the entry point in which execution starts when linking is complete. Tag character 1 is used when the entry address is absolute; tag character 2 when the address is relocatable. The field lists the address in hexadecimal form.

Table 5–6. Tag Characters (Continued)

Tag Character	Description
3,4,X	Tag characters 3, 4, and X are used for external references. Tag character 3 is used when the last appearance of the externally referenced symbol is in program-relocatable code; tag character 4 when it is in absolute code; and the X tag when it is in data- or common-relocatable code. Tag characters 3 and 4 are associated with two fields. Tag character X may identify one additional field. **Fields** – Field one contains the location of the last appearance of the symbol. – Field two contains the symbol itself. – Field three is only used to supply the common number for the X tag.
E	Used for external references. An E tag is used when a nonzero quantity is to be added to a reference. **Fields** – Field 1 identifies the reference by occurrence in the object code (0, 1, 2, ...). In other words, the value in field one is an index into references identified by 3, 4, V, X, Y and Z tags in the object code. The list is maintained by order of occurrence (i.e., the first entry in the list is the symbol located in field two of the first 3, 4, V, X, Y, or Z tag). – Field 2 contains the value to be added to the reference after the reference is resolved.
@	Used for external references of an 8-bit value. It serves the same purpose for 8-bit values that the E-tag serves for 16-bit values.
5, 6, W	Used for external definitions. Tag character 5 is used when the location is program-relocatable. Tag character 6 is used when the location is absolute. Tag character W is used when the location is data- or common-relocatable. The fields are used by the linker to provide the desired linking to the external definition. **Fields** – Field one contains the location of the last appearance of the symbol. – Field two contains the symbol of the external definition. – Field three of tag character W contains the common number.
7	Precedes the checksum, and is placed at the end of the set of fields in the record. The checksum is an error detection word and is formed as the record is being written. It is the two's complement of the sum of the 8-bit ASCII values of the characters of the record from the first tag of the record through the checksum tag, 7.
9, A, S, P	Used with load addresses, required for data words that are to be placed at other than the next immediate memory addresses. Tag character 9 is used when the load address is absolute. Tag character A is used when the load address is program-relocatable. Tag character S is used when the load address is data-relocatable. Tag character P is used when the load address is common-relocatable. **Fields** – Field one contains the load address. – Field two is only present for tag character P and contains the common number.

Table 5–6. Tag Characters (Concluded)

Tag Character	Description
*, B, C, T, N	Used with data words. Tag characters * and B are used when the data is absolute (i.e., an instruction word or a word that contains text characters or absolute constants). Tag * is used for absolute byte data (8 bits) and B is used for absolute word data (16 bits). Tag character C is used for a word that contains a program-relocatable address. Tag character T is used for a word that contains a data-relocatable address. Tag character N is used for a word that contains a common-relocatable address. **Fields** – Field one contains the data word. The linker places the data word in the memory location specified in the preceding load address field or in the memory location that follows the preceding data word. – Field two is only used with N and contains the common number.
G, H, J	Used when the symbol table option is specified. Tag character G is used when the location or value of the symbol is program-relocatable, tag character H is used when the location or value of the symbol is absolute, and tag character J is used when the location or value of the symbol is data- or common-relocatable. **Fields** – Field one contains the location or value of the symbol. – Field two contains the symbol to which the location is assigned. – Field three is used with tag character J only and contains the common number.
U	Generated by the LOAD directive. The symbol specified is treated as if it were the value specified in an INCLUDE command to the linker. **Fields** – Field one contains zeros. – Field two contains the symbol for which the loader will search for a definition.
V, Y, Z	Used for secondary external references. Tag character V is used when the last appearance of the externally referenced symbol is in program-relocatable code; tag character Y when it is in absolute code; and the Z tag when it is in data- or common-relocatable code. Tag characters V and Y are associated with two fields. Tag character Z may identify one additional field. **Fields** – Field one contains the location of the last appearance of the symbol. – Field two contains the symbol itself. – Field three is only used to supply the common number for the Z tag.
8	Also associated with the checksum field, but used when the checksum field is to be ignored.
D	Specifies a load bias. Its lone associated field contains the absolute address that will be used by a loader to relocate object code. The link editor does not accept the D tag.
F	Placed at the end of the record. It may be followed by blanks.

The end of each record is identified by the tag character 7 followed by the checksum field and the tag character F (this data is described above). The assembler fills the rest of the record with blanks and a sequence number and begins a new record with the appropriate tag character.

The last record of an object module has a colon (:) in the first character position of the record, followed by blanks or time and date identifying data.

Table 5–7 defines the object record format and tags.

Table 5–7. Object Record Format and Tags

Tag	1st Field	2nd Field	3rd Field
Module Definition			
K	PSEG Length	Program ID (8)	
M	DSEG Length	$DATA	0000
M	Blank Common Length	$BLANK	Common #
M	CSEG Length	Common Name (6)	Common #
Entry Point Definition			
1	Absolute Address		
2	P–R Address		
Load Address			
9	ABSOLUTE ADDRESS		
A	P–R Address		
S	D–R Address		
P	C–R Address	Common or CBSEG #	
Data			
*	Absolute 8–bit Value (2)		
B	Absolute 16–bit Value		
C	P–R Address		
T	D–R Address		
N	C–R Address	Common or CBSEG #	
External Definitions			
6	Absolute Value	Symbol (6)	
5	P–R Address	Symbol (6)	
W	D–R/C–R Address	Symbol (6)	Common #
External References			
3	P–R Address of Chain	Symbol (6)	
4	Absolute Address of Chain	Symbol (6)	
X	D–R/C–R Address of Chain	Symbol (6)	Common *
E	Symbol Index Number	Absolute Offset	
@	Symbol Index Number	Offset (2)	Mask (2)
Symbol Definitions			
G	P–R Address	Symbol (6)	
H	Absolute Value	Symbol (6)	
J	D–R/C–R Address	Symbol (6)	Common #
Force External Link			
U	0000	Symbol (6)	
Secondary External Reference			
V	P–R Address of Chain Entry	Symbol (6)	
Y	Absolute Address of Chain	Symbol (6)	
Z	D–R/C–R Address of Chain	Symbol (6)	Common #
Check Sum			
7	Value		

Table 5–7. Object Record Format and Tags (Continued)

Tag	1st Field	2nd Field	3rd Field
Ignore Check Sum			
8	Any Value		
Load Bias			
D	Absolute Address		
End of Record			
F			
End of Object Module			
:			

Notes: 1) All field widths are four characters unless otherwise specified by numbers in parenthesis.
2) If the first tag is 01 (hex), the file is in compressed object format.
3) P–R Program segment relative (address)
D–R Data segment relative (address)
C–R Common segment relative (address)

5.8.1.1 External References in Object Code

The link editor allows the use of external references in the object code. (See Chapter 7.)

5.8.1.2 Changing Object Code

In most cases, changing the object code is not the recommended way to correct errors in a program. All changes or corrections to a program should be made in the source code, then the program should be re-assembled. Failure to follow this procedure can make subsequent program correction or maintenance impossible. The information in the following paragraphs is intended for those rare instances when re-assembly is not possible. Any changes made directly to the object code should be thoroughly documented so that the programmers who come later can see what the program actually does, not what the source code says that it does.

To correct the object code without re-assembling a program, change the object code by changing or adding one or more records. One additional tag character is recognized by the loader to permit specifying an abolute address that will be used to relocate object code. The additional tag character, D, may be used in object records changed or added manually.

Tag character D is followed by a load bias (offset) value. The loader uses this value instead of the load bias computed by the loader itself. The loader adds the load bias to all relocatable entry addresses, external references, external definitions, load addresses, and data. The effect of the D tag character is to specify that area of memory into which the loader loads the program. The tag character D and the associated field must be placed ahead of the object code generated by the assembler.

Correcting the object code may require only changing a character or a word in an object code record. You may duplicate the record up to the character or word in error, replace the incorrect data with the correct data, and duplicate the remainder of the record up to the seven tag character. The changes will cause a checksum error when the checksum is verified as the record is loaded, so you must:

- Change the 7 tag character to an 8 tag character, in which case the checksum value is ignored, **or**
- Recalculate the checksum.

When more extensive changes are required, you may write an additional object code record or records. Begin each record with a tag character 9, A, S, or P, followed by an absolute load address or a relocatable load address. This may be an address into which an existing object code record places a different value. The new value on the new record will override the other value when the new record follows the other record in the loading sequence. Follow the load address with a tag character *, B, C, T, or N and an absolute data word or a relocatable data word. Additional data words preceded by appropriate tag characters may follow. When additional data is to be placed at a nonsequential address, write another load address tag character followed by the load address and data words preceded by tag characters. When the record is full, or all changes have been written, write tag character F to end the record.

When additional relocatable memory locations are loaded as a result of changes, you must change field one of tag character K, which contains the number of bytes of relocatable code. For example, if the object field written by the assembler contained 1000 hex bytes of relocatable code and you have added eight bytes in a new object record, additional memory locations will be loaded. You must find the K tag character in the object code file and change the value following the tag character from 1000 to 1008; you must also change the tag character 7 to 8 in that record, or recalculate the checksum.

When added records place corrected data in locations previously loaded, the added records must follow the incorrect records. The loader processes the records as they are read from the object file, and the last record that affects a given memory location determines the contents of that location at execution time.

The object code records that contain the external definition fields, the external reference fields, the entry address field, and the final program start field must follow all other object records. An additional field or record may be added to include reference to a program identifier. The tag character is 4, and the hexadecimal field contains zeros. The second field contains the first six characters of the IDT character string. External definitions may be added using tag character 5 or 6 followed by the relocatable or absolute address, respectively. The second field contains the defined symbol, filled to the right with blanks when the symbol contains less than six characters.

Note:

Both object code to be linked and object code to be downloaded can be changed without re-assembling the program. The link editor, though, will not accept tag character D in changed or added object records.

5.9 Assembling Files – Examples

The assembler is invoked by executing XASM7.EXE. The standard convention for filenames is

`FILENAME.ASM`	Source File
`FILENAME.LST`	Listing File
`FILENAME.MPO`	Object File

By executing XASM7 FILENAME; the assembler will assume a source file input of FILENAME.ASM and produce output files with the same name and default extensions .LST and .MPO. If the ";" is omitted, the assembler will prompt for user defined output file names.

If many object modules are to be assembled at the same time (prior to a linkage for example), a batch file can be set up in the following way.

ASM.BAT would contain :

```
XASM7 FILE1;
XASM7 FILE2;
XASM7 FILE3;
XASM7 FILE4;
      etc.
```

Note:

If the linker is intended to be used to create a single absolute object file from a set of object modules, AORG statements must *not* be used in any of the source files. If encountered by the linker, an AORG is likely to produce the error message ILLEGAL INTERMEDIATE TAG ENCOUNTERED. The final object code should be assigned absolute addresses by the linker using the associated link control file (see Chapter 7).

Chapter 6

Assembly Language Instruction Set

The TMS7000 assembly language instruction set and resulting object code are fully compatible with all TMS7000 family members described within this manual. This does *not*, however, mean that all devices are necessarily *software* compatible. Differences in ROM size, RAM size, available I/O, and *peripheral map* may require minor changes in the source code when migrating between TMS70CX0 devices and TMS7XCX2 or TMS70CX8 devices.

The TMS7000 instruction set contains 61 instructions that control input, output, data manipulation, data comparison, and program flow. The instruction set can be divided into eight functional categories:

- Arithmetic Instructions
- Branch and Jump Instructions
- Compare Instructions
- Control Instructions
- Load and Move Instructions
- Logical Instructions
- Shift Instructions
- I/O Instructions

Topics in this chapter include:

Section		Page
6.1	Definitions	6-2
6.2	Addressing Modes	6-3
6.3	Instruction Set Overview	6-8
6.4	Software Compatibility	6-71

6.1 Definitions

Table 6–1 lists and defines the symbols used in the instruction set.

Table 6–1. TMS7000 Symbol Definitions

Symbol	Definition	Symbol	Definition
A	Register A or R0 in register file	B	Register B or R1 in register file
Rn	Register n of register file	Pn	Port n of peripheral file ($0 \le n \le 255$)
s	Source operand	d	Destination operand
Rs	Source register in register file	Ps	Source register in peripheral file ($0 \le s \le 255$)
Rd	Destination register in register file	Pd	Destination in peripheral file ($0 \le d \le 255$)
Rp	Register pair	iop	Immediate operand
ST	Status register	SP	Stack pointer
PC	Program counter	pcn	Location of the next instruction
$	Current value of program counter	b	Bit number, as in b7 ($0 \le b \le 7$)
offset	Relative address (offset = ta – pcn)	ta	Target address (ta = offset + pcn)
@	Indicates an address or label	%	Indicates immediate operand
*	Indicates indirect register file addressing mode	<XADDR>	Indicates an extended address operand
?	Binary number	>	Hexadecimal number
MSB	Most significant byte	LSB	Least significant byte
MSb	Most significant bit	LSb	Least significant bit
cnd	Condition	( )	Contents of
→	Is assigned to	←	Becomes equal to
[ ]	Indicates an optional entry. The brackets themselves are not entered.	< >	Indicates something that must be typed in. For example, <offset> indicates that an offset must be entered. The brackets themselves are not entered.

6.2 Addressing Modes

TMS7000 Assembly Language supports eight addressing modes, listed in Table 6–2. Addressing modes that use 16-bit operands are sometimes referred to as extended addressing modes.

Table 6–2. TMS7000 Addressing Modes

Addressing Mode	Example		
Single register	LABEL	DEC	B
		INC	R45
		CLR	R23
Dual register	LABEL	MOV	B,A
		ADD	A,R17
		CMP	R32,R73
Peripheral file	LABEL	XORP	A,P17
		MOVP	P42,B
Immediate	LABEL	AND	%>C5,R55
		ANDP	%VALUE,P32
		BTJO	%>D6,R80,LABEL
Program counter relative	LABEL1	JMP	LABEL
		DJNZ	A,LABEL
		BTJO	%>16,R12,LABEL
		BTJOP	B,P7,LABEL
Direct memory	LABEL	LDA	@>F3D4
		CMPA	@LABEL
Register file indirect	LABEL	STA	*R43
Indexed	LABEL2	BR	@LABEL(B)

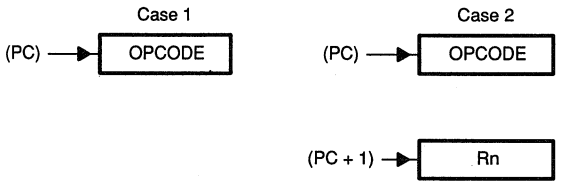
6.2.1 Single Register Addressing Mode

Single register addressing mode instructions use a single register that contains an 8-bit operand. The register can be specified as Rn, where n is the register file number in the range 0–127 or 0–255, depending upon the amount of on-chip RAM available.

A and B can denote R0 and R1, respectively. Single register addressing mode instructions that use registers A and B are also called *implied operand instructions.*

Single register addressing mode instructions that specify Rn are called *single operand instructions.* Figure 6–1 illustrates the object code generated by a single operand instruction for the the following cases:

Figure 6–1. Single Register Addressing Mode Object Code

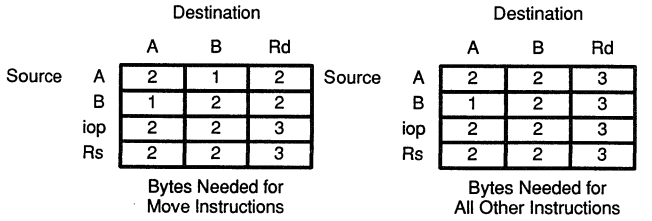



```
Case 1:     <inst>  A
            <inst>  B
Case 2:     <inst>  Rn
```

6.2.2 Dual Register Addressing Mode

Dual register addressing mode instructions use a source and a destination register that contain 8-bit operands. Assembly language syntax specifies the source register before the destination register. Figure 6–2 illustrates the byte requirements for all dual addressing instructions including the unique requirements of the move instructions using this addressing mode.

Figure 6–2. Dual Register Addressing Mode Byte Requirements



6.2.3 Peripheral-File Addressing Mode

Peripheral-file addressing mode instructions perform I/O tasks. Each PF register is an 8-bit port that can be referred to as Pn.

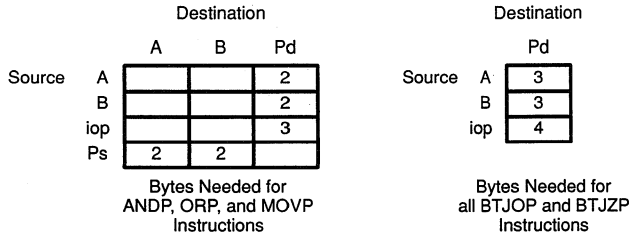
Four instructions use peripheral-file addressing mode:

- MOVP,
- ANDP,
- ORP, and
- XORP.

These instructions may use register A or B as the source register and Pn as the destination register. MOVP may also be executed using Pn as the source

register and A or B as the destination register. (BTJOP and BTJZP are also peripheral-file instructions but they have a different format.) Figure 6–3 illustrates the byte requirements of the instructions using the peripheral-file addressing mode.

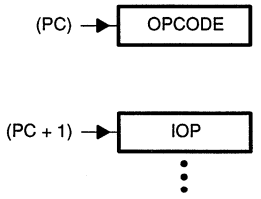
Figure 6–3. Peripheral-File Addressing Mode Byte Requirements



6.2.4 Immediate Addressing Mode

Immediate addressing mode instructions use an immediate 8-bit operand. The immediate operand can be a constant value or a label preceded by a percent sign (%). The MOVD instruction uses 16-bit immediate operands in two special formats. Figure 6–4 illustrates the simplest case of an instruction using this mode.

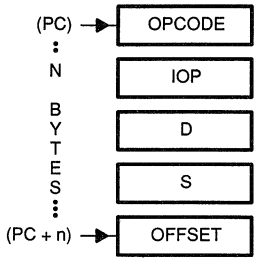
Figure 6–4. Immediate Addressing Mode Object Code



6.2.5 Program Counter Relative Addressing Mode

All jump instructions use program counter relative addressing mode. The assembly language source statement for a jump instruction always includes a target address (ta). The microcomputer uses the target address to calculate an offset as follows: *offset = ta – pcn*, where **pcn** is the location of the next instruction and $-128 \leq ra \leq 127$. Figure 6–5 illustrates object code generated by a jump instruction.

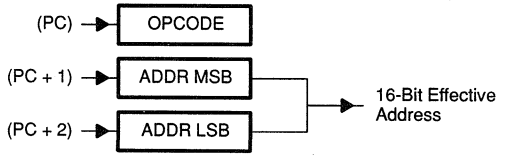
Figure 6–5. Program Counter Relative Addressing Mode Object Code



6.2.6 Direct Memory Addressing Mode

Direct addressing mode instructions use a 16-bit address that contains the operand. The 16-bit address is preceded by an @ sign and can be written as a constant value or as a label. Figure 6–6 shows how the object code produced by an instruction using the direct memory addressing mode generates a 16-bit effective address.

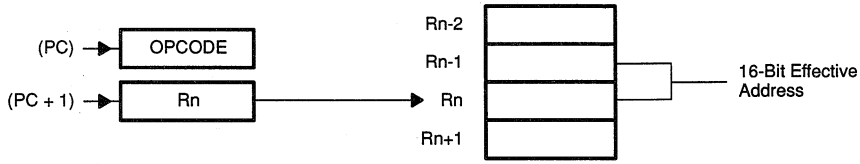
Figure 6–6. Direct Memory Addressing Mode Object Code



6.2.7 Register File Indirect Addressing Mode

Register file indirect addressing mode instructions use the contents of a register pair as a 16-bit effective address. The indirect register file address is written as a register number (Rn) preceded by an asterisk (*), that is, *Rn. The LSB of the address is contained in Rn, and the MSB of the address is contained in the previous register (Rn–1). Figure 6–7 shows how the object code produced by an instruction using register file indirect addressing mode generates a 16-bit effective address.

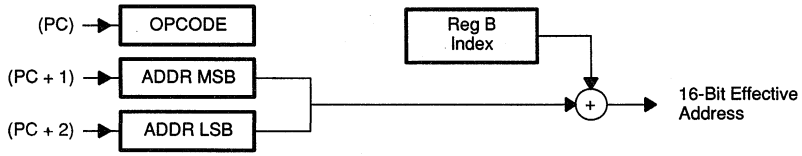
Figure 6–7. Register File Indirect Addressing Mode Object Code



6.2.8 Indexed Addressing Mode

Indexed addressing mode instructions generate a 16-bit address by adding the contents of the B register to a 16-bit direct memory address. The assembly language statement for the indexed addressing mode contains the direct memory address written as a 16-bit constant value or a label, preceded by an @ sign and followed by a B in parentheses: @LABEL(B). The addition automatically transfers any carries into the MSB. Figure 6–8 illustrates how the object code produced by an instruction using the indexed addressing mode generates a 16-bit effective address. Do not confuse this mode with the MOVD (Move Double) instruction's addressing mode.

Figure 6–8. Indexed Addressing Mode Object Code



6.3 Instruction Set Overview

Table 6–3 lists all instruction formats, opcodes, byte lengths, cycles/instruction, operand types, status bits affected, and an operational description.

The TMS7000 Assembly Language instructions are presented in alphabetical order following the instruction overview table. All instructions may have optional labels preceding the mnemonic and comments following the operands. Labels, mnemonics, operands, and comments must be separated by at least one space:

```
START   MOVP  %>00,P0 Initialize to single chip
```

The byte count for each instruction may be determined from its instruction type and its operands.

Table 6–3. TMS7000 Family Instruction Overview

Mnemonic		Opcode	Bytes	Cycles $T_{c(C)}$	Status C	N	Z	I	Operation Description
ADC	B,A	69	1	5	R	R	R	x	(s) + (d) + (C) → (d) Add the source, destination, and carry bit together. Store at the destination address.
	Rs,A	19	2	8					
	Rs,B	39	2	8					
	Rs,Rd	49	3	10					
	%iop,A	29	2	7					
	%iop,B	59	2	7					
	%iop,Rd	79	3	9					
ADD	B,A	68	1	5	R	R	R	x	(s) + (d) → (d) Add the source and destination operands at the destination address.
	Rs,A	18	2	8					
	Rs,B	38	2	8					
	Rs,Rd	48	3	10					
	%iop,A	28	2	7					
	%iop,B	58	2	7					
	%iop,Rd	78	3	9					
AND	B,A	63	1	5	0	R	R	x	(s) .AND. (d) → (d) AND the source and destination operands together and store at the destination address.
	Rs,A	13	2	8					
	Rs,B	33	2	8					
	Rs,Rd	43	3	10					
	%iop,A	23	2	7					
	%iop,B	53	2	7					
	%iop,Rd	73	3	9					
ANDP	A,Pd	83	2	10	0	R	R	x	(s) .AND. (Pn) → (Pn) AND the source and destination operands together, and store at the destination address.
	B,Pd	93	2	9					
	%iop,Pd	A3	3	11					

Note: Add two to cycle count if branch is taken.

Legend:

0	Status Bit set always to 0.
1	Status Bit set always to 1.
R	Status Bit set to a 1 or a 0 depending on results of operation.
x	Status Bit not affected.
b	Bit ( ) affected.
Ofst	Offset

Table 6–3. TMS7000 Family Instruction Overview (Continued)

Mnemonic		Opcode	Bytes	Cycles $T_{c(C)}$	Status C	N	Z	I	Operation Description
(1) BTJO	B,A,Ofst	66	2	7 (9)	0	R	R	x	If (s) .AND. (d) ≠ 0, then (PC) + offset → (PC) If the AND of the source and destination operands ≠ 0, the PC will be modified to include the offset.
	Rn,A,Ofst	16	3	10 (12)					
	Rn,B,Ofst	36	3	10 (12)					
	Rn,Rd,Ofst	46	4	12 (14)					
	%iop,A,Ofst	26	3	9 (11)					
	%iop,B,Ofst	56	3	9 (11)					
	%iop,Rn,Ofst	76	4	11 (13)					
(1) BTJOP	A,Pn,Ofst	86	3	11 (13)	0	R	R	x	If (s) .AND. (Pn) ≠ 0, then (PC) + offset → (PC) If the AND of the source and destination operands ≠ 0, the PC will be modified to include the offset.
	B,Pn,Ofst	96	3	10 (12)					
	%>iop,Pn,Ofst	A6	4	12 (14)					
(1) BTJZ	B,A,Ofst	67	2	7 (9)	0	R	R	x	If (s) .AND. NOT(d) ≠ 0, then (PC) + offset → (PC) If the AND of the source and NOT(destination operands ≠ 0, the PC will be modified to include the offset.
	Rn,A,Ofst	17	3	10 (12)					
	Rn,B,Ofst	37	3	10 (12)					
	Rn,Rf,Ofst	47	4	12 (14)					
	%>iop,A,Ofst	27	3	9 (11)					
	%>iop,B,Ofst	57	3	9 (11)					
	%>iop,Rn,Ofst	77	4	11 (13)					
(1) BTJZP	A,Pn,Ofst	87	3	11 (13)	0	R	R	x	If (s) .AND. NOT(Pn) ≠ 0, then (PC) + offset → (PC) If the AND of the source and NOT(destination) operands ≠ 0, the PC will be modified to include the offset.
	B,Pn,Ofst	97	3	10 (12)					
	%>iop,Pn,Ofst	A7	4	12 (14)					
BR	@Label	8C	3	10	x	x	x	x	(d) → (PC) The PC will be replaced with the contents of the destination operand.
	@Label(B)	AC	3	12					
	*Rn	9C	2	9					
CALL	@Label	8E	3	14	x	x	x	x	(SP) + 1 → (SP) (PC MSB) → ((SP)) (SP) + 1 → (SP) (PC LSB) → ((SP)) Operand Address → (PC)
	@Label(B)	AE	3	16					
	*Rn	9E	2	13					

Note: Add two to cycle count if branch is taken.

Legend:

0 Status Bit set always to 0.
1 Status Bit set always to 1.
R Status Bit set to a 1 or a 0 depending on results of operation.
x Status Bit not affected.
b Bit ( ) affected.
Ofst Offset

Table 6–3. TMS7000 Family Instruction Overview (Continued)

Mnemonic		Opcode	Bytes	Cycles $T_{c(C)}$	Status C	N	Z	I	Operation Description
CLR	A	B5	1	5	0	0	1	x	0 → (d) Clear the destination operand.
	B	C5	1	5					
	Rd	D5	2	7					
CLRC		B0	1	6	0	R	R	x	0 → (C) Clears the carry bit.
CMP	B,A	6D	1	5	R	R	R	x	(d) – (s) computed Set flags on the result of the source operand subtracted from the destination operand.
	Rn,A	1D	2	8					
	Rn,B	3D	2	8					
	Rn,Rn	4D	3	10					
	%iop,A	2D	2	7					
	%iop,B	5D	2	7					
	%iop,Rn	7D	3	9					
CMPA	@Label	8D	3	12	R	R	R	x	(A) – (s) computed Set flags on result of the source operand subtracted from A.
	@Label(B)	AD	3	14					
	*Rn	9D	2	11					
DAC	B,A	6E	1	7	R	R	R	x	(s) + (d) + (C) → (d) (BCD) The source, destination, and the carry bit are added, and the BCD sum is stored at the destination address.
	Rs,A	1E	2	10					
	Rs,B	3E	2	10					
	Rs,Rd	4E	3	12					
	%>iop,A	2E	2	9					
	%>iop,B	5E	2	9					
	%>iop,Rd	7E	3	11					
DEC	A	B2	1	5	R	R	R	x	(d) – 1 → (d) Decrement destination operand by 1.
	B	C2	1	5					
	Rd	D2	2	7					
DECD	A	BB	1	9	R	R	R	x	(rp) – 1 → (rp) Decrement register pair by 1. C = 0 on 0 – FFFF transition.
	B	CB	1	9					
	Rp	DB	2	11					
DINT		06	1	5	0	0	0	0	0 → (global interrupt enable bit). Clear the I bit.
(1) DJNZ	A,Ofst	BA	2	7 (9)	x	x	x	x	(d) – 1 → (d); If (d) ≠ 0, (PC) + offset → (PC)
	B,Ofst	CA	2	7 (9)					
	Rd,Ofst	DA	3	9 (11)					

Note: Add two to cycle count if branch is taken.

Legend:

0 Status Bit set always to 0.
1 Status Bit set always to 1.
R Status Bit set to a 1 or a 0 depending on results of operation.
x Status Bit not affected.
b Bit ( ) affected.
Ofst Offset

Table 6–3. TMS7000 Family Instruction Overview (Continued)

Mnemonic		Opcode	Bytes	Cycles $T_C(C)$	Status C	N	Z	I	Operation Description
DSB	B,A	6F	1	7	R	R	R	x	(d) – (s) – 1 + (C) → (d) (BCD) The source operand is subtracted from the destination; this sum is then reduced by 1 and the carry bit is then added to it. The result is stored as a BCD number.
	Rs,A	1F	2	10					
	Rs,B	3F	2	10					
	Rs,Rd	4F	3	12					
	%>iop,A	2F	2	9					
	%>iop,B	5F	2	9					
	%>iop,Rd	7F	3	11					
EINT		05	1	5	1	1	1	1	1 → (global interrupt enable bit). Set the I bit.
IDLE		01	1	6	x	x	x	x	(PC) → (PC) until interrupt (PC) + 1 → (PC) after return from interrupt Stops µC execution until an interrupt.
INC	A	B3	1	5	R	R	R	x	(d) + 1 → (d) Increase the destination operand by 1.
	B	C3	1	5					
	Rd	D3	2	7					
INV	A	B4	1	5	0	R	R	x	NOT(d) → (d) 1's complement the destination operand.
	B	C4	1	5					
	Rd	D4	2	7					
JMP	Ofst	E0	2	7	x	x	x	x	(PC) + offset → (PC) The PC is modified by an offset to create a new PC value.
(1)					x	x	x	x	
JC	Ofst	E3	2	5 (7)					If conditions are met, then (PC) + offset → (PC) If the needed conditions are met, the PC is modified by the offset to form a new PC value.
JEQ	Ofst	E2	2	5 (7)					
JHS	Ofst	E3	2	5 (7)					
JL	Ofst	E7	2	5 (7)					
JN	Ofst	E1	2	5 (7)					
JNC	Ofst	E7	2	5 (7)					
JNE	Ofst	E6	2	5 (7)					
JNZ	Ofst	E6	2	5 (7)					
JP	Ofst	E4	2	5 (7)					
JPZ	Ofst	E5	2	5 (7)					
JZ	Ofst	E2	2	5 (7)					
LDA	@Label	8A	3	11	0	R	R	x	(s) → (A) Move the source operand to A.
	@Label(B)	AA	3	13					
	*Rn	9A	2	10					

Note: Add two to cycle count if branch is taken.

Legend:

0 Status Bit set always to 0.
1 Status Bit set always to 1.
R Status Bit set to a 1 or a 0 depending on results of operation.
x Status Bit not affected.
b Bit ( ) affected.
Ofst Offset

Table 6–3. TMS7000 Family Instruction Overview (Continued)

Mnemonic		Opcode	Bytes	Cycles $T_{C(C)}$	Status C	N	Z	I	Operation Description
LDSP		0D	1	5	x	x	x	x	(B) → (SP) Load SP with Register B's contents.
MOV	A,B	C0	1	6	0	R	R	x	(s) → (d) Replace the destination operand with the source operand.
	A,Rd	D0	2	8					
	B,A	62	1	5					
	B,Rd	D1	2	7					
	Rs,A	12	2	8					
	Rs,B	32	2	8					
	Rs,Rd	42	3	10					
	%>iop,A	22	2	7					
	%>iop,B	52	2	7					
	%>iop,Rd	72	3	9					
MOVD	%>iop,Rp	88	4	15	0	R	R	x	(rp) → (rp) Copy the source register pair to the destination register pair.
	%>iop(B),Rp	A8	4	17					
	Rp,Rp	98	3	14					
MOVP	A,Pd	82	2	10	0	R	R	x	(s) → (d) Copy the source operand into the destination operand.
	B,Pd	92	2	9					
	%>iop,Pd	A2	3	11					
	Ps,A	80	2	9					
	Ps,B	91	2	8					
MPY	B,A	6C	1	44	0	R	R	x	(s) × (d) → (A,B) Multiply the source and destination operands, store the result in registers A (MSB) and B (LSB).
	Rs,A	1C	2	47					
	Rs,B	3C	2	47					
	Rn,Rn	4C	3	49					
	%>iop,A	2C	2	46					
	%>iop,B	5C	2	46					
	%>iop,Rn	7C	3	48					
NOP		00	1	4	x	x	x	x	(PC) + 1 → (PC) Add 1 to the PC.
OR	B,A	64	1	5	0	R	R	x	(s) .OR. (d) → (d) Logically OR the source and destination operands, and store the results at the destination address.
	Rs,A	14	2	8					
	Rs,B	34	2	8					
	Rs,Rd	44	3	10					
	%>iop,A	24	2	7					
	%>iop,B	54	2	7					
	%>iop,Rd	74	3	9					

Note: Add two to cycle count if branch is taken.

Legend:

0 Status Bit set always to 0.
1 Status Bit set always to 1.
R Status Bit set to a 1 or a 0 depending on results of operation.
x Status Bit not affected.
b Bit ( ) affected.
Ofst Offset

Table 6–3. TMS7000 Family Instruction Overview (Continued)

Mnemonic	Opcode	Bytes	Cycles $T_{c(C)}$	Status C N Z I	Operation Description
ORP A,Pd B,Pd %>iop,Pd	84 94 A4	2 2 3	10 9 11	0 R R x	(s) .OR. (d) → (d) Logically OR the source and destination operands, and store the results at the destination address.
POP A B Rd	B9 C9 D9	1 1 2	6 6 8	0 R R x	((SP)) → (d) (SP) –1 → (SP) Copy the last byte on the stack into the destination address.
POP ST	08	1	6	Loaded from stack	((SP)) → (ST) (SP) – 1 → (SP) Replace the status register with the last byte of the stack.
PUSH A B Rs	B8 C8 D8	1 1 2	6 6 8	x x x x	(SP) + 1 → (SP) (s) → (SP) Copy the operand onto the stack.
PUSH ST	0E	1	6	x x x x	(SP) + 1 → (SP) (Status register) → ((SP)) Copy the status register onto the stack.
RETI	0B	1	9	Loaded from stack	(SP) → (PC) LSByte (SP) – 1 → (SP) (SP) → (PC) MSByte (SP) – 1 → (SP) (SP) → status register (SP) – 1 → (SP)
RETS	0A	1	7	x x x x	(SP) → (PC LSB) (SP) – 1 → (SP) (SP) → (PC MSB) (SP) – 1 → (SP)
RL A B Rd	BE CE DE	1 1 2	5 5 7	b7 R R x	Bit(n) → Bit(n + 1) Bit(7) → Bit(0) and Carry
RLC A B Rd	BF CF DF	1 1 2	5 5 7	b7 R R x	Bit(n) → Bit(n + 1) Carry → Bit(0) Bit(7) → Carry

Note: Add two to cycle count if branch is taken.

Legend:

0 Status Bit set always to 0.
1 Status Bit set always to 1.
R Status Bit set to a 1 or a 0 depending on results of operation.
x Status Bit not affected.
b Bit ( ) affected.
Ofst Offset

Table 6–3. TMS7000 Family Instruction Overview (Continued)

Mnemonic		Opcode	Bytes	Cycles $T_{c(C)}$	Status C	N	Z	I	Operation Description
RR	A	BC	1	5	b0	R	R	x	Bit(n + 1) → Bit(n) Bit(0) → Bit(7) and Carry
	B	CC	1	5					
	Rd	DC	2	7					
RRC	A	BD	1	5	b0	R	R	x	Bit(n + 1) → Bit(n) Carry → Bit(7) Bit(0) → Carry
	B	CD	1	5					
	Rd	DD	2	7					
SBB	B,A	6B	1	5	R	R	R	x	(d) – (s) – 1 + (C) → (d) Destination minus source minus 1 plus carry; stored at the destination address.
	Rs,A	1B	2	8					
	Rs,B	3B	2	8					
	Rs,Rd	4B	3	10					
	%>iop,A	2B	2	7					
	%>iop,B	5B	2	7					
	%>iop,Rd	7B	3	9					
SETC		07	1	5	1	0	1	x	1 → (C) Set the carry bit.
STA	@Label	8B	3	11	0	R	R	x	(A) → (d) Store A at the destination.
	@Label(B)	AB	3	13					
	*Rd	9B	2	10					
STSP		09	1	6	x	x	x	x	(SP) → (B) Copy the SP into register B.
SUB	B,A	6A	1	5	R	R	R	x	(d) – (s) → (d) Store the destination operand minus the source operand into the destination.
	Rs,A	1A	2	8					
	Rs,B	3A	2	8					
	Rs,Rd	4A	3	10					
	%>iop,A	2A	2	7					
	%>iop,B	5A	2	7					
	%>iop,Rd	7A	3	9					
SWAP	A	B7	1	8	R	R	R	x	d(Hn,Ln) → d(Ln,Hn) Swap the operand's hi and lo nibbles.
	B	C7	1	8					
	Rn	D7	2	10					
TRAP	0–23	E8–FF	1	14	x	x	x	x	(SP) + 1 → (SP) (PC MSB) → (SP) (SP) + 1 → (SP) (PC LSB) → (SP) (Entry Vector) → (PC)

Note: Add two to cycle count if branch is taken.

Legend:

0 Status Bit set always to 0.
1 Status Bit set always to 1.
R Status Bit set to a 1 or a 0 depending on results of operation.
x Status Bit not affected.
b Bit ( ) affected.
Ofst Offset

Table 6–3. TMS7000 Family Instruction Overview (Concluded)

Mnemonic		Opcode	Bytes	Cycles $T_{C(C)}$	Status C	N	Z	I	Operation Description
TSTA		B0	1	6	0	R	R	x	0 → (C) Set carry bit; set sign and zero flags on the value of register A.
TSTB		C1	1	6	0	R	R	x	0 → (C) Set carry bit; set sign and zero flags on the value in register B.
XCHB	A Rn	B6 D6	1 2	6 8	0	R	R	x	(B) ←→ (d) Swap the contents of register B with (d).
XOR	B,A Rs,A Rs,B Rs,Rd %>iop,A %>iop,B %>iop,Rd	65 15 35 45 25 55 75	1 2 2 3 2 2 3	5 8 8 10 7 7 9	0	R	R	x	(s).XOR. (d) → (d) Logically exclusive OR the source and destination operands, store at the destination address.
XORP	A,Pd B,Pd %>iop,Pd	85 95 A5	2 2 3	10 9 11	0	R	R	x	(s) .XOR. (Pn) → (Pn) Logically exclusive OR the source and destination operands, store at the destination.

Note: Add two to cycle count if branch is taken.

Legend:

0 Status Bit set always to 0.
1 Status Bit set always to 1.
R Status Bit set to a 1 or a 0 depending on results of operation.
x Status Bit not affected.
b Bit ( ) affected.
Ofst Offset

Syntax

[<label>] ADC <s>,<Rd>

Execution

(s) + (Rd) + (C) → (Rd)

Status Bits

C Set to 1 on carry-out of (s) + (Rd) + (C)
Z Set on result
N Set on result

Description

ADC adds the contents of the source, the contents of the destination register, and the carry bit. It stores the result in the destination register.

Adding a 0 to the destination register is equivalent to a conditional increment (increment on carry).

ADC can implement multiprecision addition of signed or unsigned integers. For example, the 16-bit integer in register pair (R2,R3) may be added to the 16-bit integer in (A,B) as follows:

```
ADD     R3,B                Low order bytes added
ADC     R2,A                High order bytes added
```

Examples

```
LABEL1   ADC R66,R117       Adds the contents of
*                           register 66, register
*                           117, and the carry bit,
*                           and stores the sum in
*                           register 117

         ADC   B,A          Adds the contents of
*                           Register B, Register A,
*                           and the carry bit, and
*                           stores the sum in
*                           Register A

         ADC   %>3C,R29     Adds >3C, contents of
*                           register 29, and the
*                           carry bit, and stores
*                           the sum in register 29
```

Syntax [<label>] ADD <s>,<Rd>

Execution (s) + (Rd) → (Rd)

Status Bits

C Set to 1 on carry-out of (s) + (Rd)
Z Set on result
N Set on result

Description ADD adds two bytes and stores the result in the destination register. It can be used for signed 2's complement or unsigned addition.

Examples

```
LABEL    ADD A,B              Adds the contents of
*                             Registers A and B, stores
*                             the results in B

         ADD  R7,A            Adds the contents of R7
*                             and A, and stores the
*                             results in A

         ADD  %TOTAL,R13      Adds the contents of
*                             TOTAL to R13 and stores
*                             the result in R13
```

Syntax

[<label>] AND <s>,<Rd>

Execution

(s) .AND. (Rd) → (Rd)

Status Bits

C ← 0
N Set on result
Z Set on result

Description

AND logically ANDs the two 8-bit operands. Each bit in the first operand is ANDed with the corresponding bit in the second operand. This is useful for clearing and resetting bits. If you need to clear a bit in the destination operand, then put a 0 in the corresponding source bit. A 1 in a source bit will not change the corresponding destination bit.

This is the truth table for the AND instruction:

Source Bit	Destination Bit	AND Result
0	0	0
0	1	0
1	0	0
1	1	1

Examples

```
LABEL   AND   %>1,R12      Clear all bits in R12 except
                           Bit 0, which will remain
                           unchanged

        AND   R7,A         AND the contents of R7 to A
*                          and store the contents in A

        AND   B,A          AND the contents of B to A
*                          and store the contents in A
```

Syntax [<label>] ANDP <s>,<Pd>

Execution (s) .AND. (Pd) → (Pd)

Status Bits

C ← 0
N Set on result
Z Set on result

Description ANDP clears one or more bits in a peripheral-file register. It can reset an individual output line to zero when the source is an immediate operand serving as a mask field. Since the peripheral register is read before it is ANDed, it may not work with some peripheral locations which have a different function when reading than when writing. **The only valid source operands are A, B, and %>iop.**

Examples

```
LABEL   ANDP   %>DF,P6     Clear bit 5 of Port B (P6)

        ANDP   %>FE,P9     Clear Bit 0 of Port C Data
                           Direction Register
                           (CDDR - P9)

        ANDP   A,P33       AND the contents of A and
                           P33 and store in P33
```

Syntax [<label>] BR <XADDR>

Execution (XADDR) → (PC)

Status Bits None

Description BR branches to **any** location in the 64K memory space, including the on-chip RAM. BR supports three extended addressing modes:

- Direct
- Indirect
- Indexed

The powerful concept of computed GOTOs is supported by the `BR *Rn` instruction. An indexed branch instruction of the form `BR @TABLE(B)` is an extremely efficient way to execute one of several actions on the basis of a control input. This is similar to the Pascal CASE statement. For example, suppose register B contains a control value. The program can branch to label ACTION0 if B=0, ACTION1 if B=1, etc, for up to 128 different actions. This technique may also be used to transfer control on character inputs, error codes, etc.

Examples

```
LABEL1      BR  @THERE          Direct addressing

            BR   @TABLE(B)     Indexed addressing
            BR   *R14          Indirect addressing

LABEL2      EQU $              Start execution here
            MOV  R3,B          Move control input to B
            RL   B             Multiply by 2 to get
*                              table offset
            BR   @TABLE(B)     Branch to correct J<cnd>
                               statement

DISPATCH    EQU  $             Dispatch table
            JMP  ACTION0
            JMP  ACTION1
            ...
            JMP  ACTIONn
ACTION0     EQU  $
*                              <Code for action 0>
ACTION1     EQU  $
*                              <Code for action 1>
            ...
ACTIONn     EQU  $
*                              <Code for action n>
```

Syntax

[<label>] BTJO <s>,<Rn>,<offset>

Execution

If (s [Bit x]) .AND. (Rn [Bit x]) ≠ 0, then (PC) + offset → (PC)

Status Bits

C ← 0
N Set on (s) .AND. (Rn)
Z Set on (s) .AND. (Rn)

Description

BTJO tests for at least one bit position that contains a corresponding 1 in each operand. The source operand can be used as a bit mask to test for one or more 1 bits in the specified register. The operands are not changed by this instruction. If a corresponding 1 bit is found, the program branches to the offset.

Examples

```
LABEL    BTJO %>14,R4,ISSET       Jump to ISSET if R4
*                                 (bit 2) or R4 (bit
*                                 4) is a 1

         BTJO  %>1,A,LOOP         Jump to LOOP if bit
*                                 0 of Register A is
*                                 a 1

         BTJO  R37,R113,START     Jump to START if any
*                                 1 bit of R113 corre-
*                                 sponds to a 1 bit
*                                 in R37
```

Syntax

[<label>] BTJOP <s>,<Pn>,<offset>

Execution

If (s [Bit x]) .AND. (Pn [Bit x]) ≠ 0, then (PC) + offset → (PC)

Status Bits

C ← 0
N Set on (s) .AND. (Pn)
Z Set on (s) .AND. (Pn)

Description

BTJOP tests for at least one bit position that contains a corresponding 1 in each operand. The source operand can be used as a bit mask to test for at least one 1 bit in the peripheral-file register.

Examples

```
LABEL    BTJOP %>81,P4,THERE      Jump to THERE if
*                                 bit 0 or bit 7 of
*                                 Port A contain
*                                 a 1

         BTJOP  %>FF,P10,STORE    Test all bits of
*                                 Port D Data (P10);
*                                 jump to STORE if
*                                 any of the bits
*                                 are 1s

         BTJOP  B,P50,AGAIN       Jump to AGAIN if
*                                 any 1 bit of P50
*                                 corresponds to any
*                                 1 bit of the B
*                                 Register
```

Syntax [<label>] BTJZ <s>,<Rn>,<offset>

Execution If (s [Bit x]) .AND. NOT(Rn [Bit x]) ≠ 0, then (PC) + offset → (PC)

Status Bits

C ← 0
N Set on (s) .AND. (NOT Rn)
Z Set on (s) .AND. (NOT Rn)

Description BTJZ tests for at least one bit position which has a 1 in the source and a 0 in the destination. The source operand can be used as a bit mask to test for zero bits in the specified register. The operands are unchanged by this instruction. The jump is calculated starting from the opcode of the instruction just after the BTJZ.

Examples

```
LABEL    BTJZ A,R23,ZERO          If any 1 bits in A
*                                 correspond to 0 bits
*                                 in R23 then jump to
*                                 ZERO to 0 bits in R23
*                                 then jump to ZERO

         BTJZ  %>FF,A,NEXT        If A contains any 0
*                                 bits, jump to NEXT

         BTJZ  R7,R15,OUT         If any 0 bits in R15
*                                 correspond to 1 bits
*                                 in R7, jump to OUT
```

Syntax

[<label>] BTJZP <s>,<Pn>,<offset>

Execution

If (s [Bit x]) .AND. NOT(Pn [Bit x]) ≠ 0, then (PC) + offset → (PC)

Status Bits

C ← 0
N Set on (s) .AND. (NOT Pn)
Z Set on (s) .AND. (NOT Pn)

Description

BTJZP tests for at least one bit position which has a 1 in the source and an 0 in the peripheral-file register. The source operand can be used as a bit mask to test for zero bits in the peripheral-file register. The operands are unchanged by this instruction. The jump is calculated starting from the opcode of the instruction just after the BTJZP.

Examples

```
LABEL    BTJZP %>21,P4,THERE      Jump to THERE if P4
*                                 (bit 0) or P4 (bit
*                                 5) is 0

         BTJZP  %>FF,P28,STORE    Jump to STORE if P28
*                                 contains any 0s

         BTJZP  B,P37,NEXT        Jump to NEXT if P37
*                                 contains any 0 bits
*                                 corresponding to 1
*                                 bits in Register B
```

Syntax [<label>] CALL <XADDR>

Execution

(SP) + 1 → (SP)
(PC MSB) → ((SP))
(SP) + 1 → (SP)
(PC LSB) → ((SP))
(XADDR) → (PC)

Status Bits None

Description CALL invokes a subroutine and pushes the PC contents on the stack. The operand indicates the starting address of the subroutine. Use the PUSH and POP instructions to save, pass, or restore status or register values. The extended addressing modes of the CALL instruction allow powerful transfer of control functions.

Examples

```
LABEL   CALL @LABEL4          Direct addressing
        CALL  @LABEL5(B)      Indexed addressing
        CALL  *R12            Indirect addressing
```

CLR *Clear*

Syntax [<label>] CLR <Rd>

Execution 0 → (Rd)

Status Bits C ← 0
N ← 0
Z ← 1

Description CLR clears or initializes any file register including registers A and B.

Examples

```
LABEL   CLR B           Clear Register B
        CLR  A          Clear Register A
        CLR  R105       Clear register 105
```

Syntax [<label>] CLRC

Execution Set status bits

Status Bits

C ← 0
N Set on value of register A
Z Set on value of register A

Description CLRC clears the carry flag. This may be required before an arithmetic or rotate instruction. The logical and move instructions typically clear the carry bit. The CLRC opcode is equivalent to the TSTA opcode.

Examples

```
LABEL     CLRC              Clear the carry bit
```

Syntax [<label>] CMP <s>,<Rn>

Execution (Rn) – (s) computed but not stored

Status Bits
C 1 if (Rn) ≥ (s)
N Sign of result
Z 1 if (Rn) = (s)

Description CMP compares the destination operand to the source operand and sets the status bits. The CMP instruction is usually used in conjunction with a jump instruction; Table 6–4 shows which jump instructions can be used on status conditions set by CMP execution.

Table 6–4. Compare Instruction Examples — Status Bit Values

(S)	(Rn)	(Rn)–(S)	C	N	Z	Instructions That Will Jump
FF	00	01	0	0	0	JL JNC JNE JNZ JP JPZ
00	FF	FF	1	1	0	JHS JC JNE JNZ JN
00	7F	7F	1	0	0	JHS JC JNE JNZ JP JPZ
81	00	7F	0	0	0	JL JNC JNE JNZ JP JPZ
00	81	81	1	1	0	JHS JC JNE JNZ JN
80	00	80	0	1	0	JL JNC JNE JNZ JN
00	80	80	1	1	0	JHS JC JNE JNZ JN
7F	80	01	1	0	0	JHS JC JNE JNZ JP JPZ
80	7F	FF	0	1	0	JL JNC JNE JNZ JN
7F	7F	00	1	0	1	JHC JC JEQ JZ JPZ
7F	00	81	0	1	0	JL JNC JNE JNZ JN

Examples

```
LABEL   CMP  R13,R89      Set status bits on
*                         result of R89 minus R13

        CMP   B,R39       Set status bits on result
*                         of R39 minus (B)

        CMP   %>03,A      Set status bits on result
*                         of (A) minus >03
```

Syntax

[<label>] CMPA <XADDR>

Execution

(A) – (XADDR) computed but not stored

Status Bits

C 1 if (A) logically ≥ (XADDR)
N 1 if (A) arithmetically < (XADDR)
Z 1 if (A) = (XADDR)

Description

CMPA compares a long-addressed operand to the A register via direct, indirect, or indexed addressing modes. It is especially useful in table lookup programs that store the table either in extended memory or in program ROM. The status bits are set exactly as if register A were the destination and the addressed byte the source.

Examples

```
LABEL    CMPA @TABLE2          Direct addressing
         CMPA  @TABLE(B)       Indexed addressing
         CMPA  *R123           Indirect addressing
```

Syntax [<label>] DAC <s>,<Rd>

Execution (s) + (Rd) + (C) → (Rd), Produces a decimal result

Status Bits

C 1 if value of (s) + (Rd) + C ≥ 100
N Set on result
Z Set on result

Description DAC adds bytes in binary-coded decimal (BCD) form. Each byte is assumed to contain two BCD digits. DAC is not defined for non-BCD operands. DAC with an immediate operand of zero value is equivalent to a conditional increment of the destination operand (increment destination on carry). The DAC instruction automatically performs a decimal adjust on the binary sum of (s) + (Rd) + C. The carry bit is added to facilitate adding multibyte BCD strings, and so the carry bit must be cleared before execution of the first DAC instruction.

Examples

```
LABEL   DAC %>24,A          Add the packed BCD value 24,
*                           and the carry bit to the
*                           Register A carry bit to
*                           Register A

        DAC  R55,R7         Add the BCD value of R55,
*                           and the carry bit to the
*                           BCD value of R7

        DAC  B,A            Add the carry bit to the
*                           BCD value in Register B
*                           to Register A
```

Syntax

[<label>] DEC <Rd>

Execution

(d) – 1 → (Rd)

Status Bits

C 0 if (Rd) decrements from >00 to >FF; 1 otherwise
N Set on result
Z Set on result

Description

DEC subtracts 1 from any addressable operand. It is useful in counting and addressing byte arrays.

Examples

```
LABEL   DEC R102        Decrement R102 by 1

        DEC  A          Decrement Register A by 1

        DEC  B          Subtract 1 from the contents of
*                       Register B
```

Syntax

[<label>] DECD <Rp>

Execution

(Rp) – 1 → (Rp)

Status Bits

C 0 if most significant byte decrements from >00 to >FF; otherwise, C = 1
N Set on most significant byte of result
Z Set on most significant byte of result

Description

DECD decrements 16-bit indirect addresses stored in the register file. Tables longer than 256 bytes may be scanned using this instruction.

The JZ (Jump on Zero) command is often used in conjunction with the DECD command. Note that JZ jumps when the **MSB** equals zero — not just when both bytes equal zero.

Examples

```
LABEL    DECD R51              Decrement (R50,R51) register
*                              pair, R51=LSB
```

Syntax

[<label>] DINT

Execution

0 → (Global interrupt enable status bit)

Status Bits

I ← 0
C ← 0
N ← 0
Z ← 0

Description

DINT simultaneously disables all interrupts. Since the interrupt enable flag is stored in the status register, the POP ST or RETI instructions may re-enable interrupts even though a DINT instruction has been executed. During the interrupt service, the interrupt enable bit is automatically cleared after the old status register value has been pushed onto the stack.

Examples

```
LABEL    DINT          Disable global interrupt enable bit
```

Syntax [<label>] DJNZ <Rd>,<offset>

Execution (Rd) – 1 → (d)
If (Rd) ≠ 0, then (PC) + offset → (PC)

Status Bits None

Description DJNZ is used for looping control. It combines the DEC and the JNZ instructions, providing a faster and more compact instruction. DJNZ does not change the status bits.

Examples

```
LABEL   DJNZ R15,THERE          Decrement R15. If R15 p
*                               0, jump to THERE

        DJNZ  A,AGAIN           Decrement A; if A p 0,
*                               jump to AGAIN

        DJNZ  B,BACK            Decrement B; if B p 0,
*                               jump to BACK
```

Syntax

[<label>] DSB <s>,<Rd>

Execution

(Rd) – (s) – 1 + (C) → (Rd) (decimal result)

Status Bits

C 1 no borrow required, 0 if borrow required
N Set on result
Z Set on result

Description

DSB performs multiprecision decimal BCD subtraction. A DSB instruction with an immediate operand of zero value is equivalent to a conditional decrement of the destination operand. The carry bit functions as a borrow bit, so if no borrow in is required, the carry bit should be set to 1. This can be accomplished by executing the SETC instruction.

Examples

```
LABEL   DSB R15,R76          R76 minus R15 minus 1 plus
*                            the carry bit is stored
*                            in R76

        DSB  A,B             Register B minus Register
*                            A minus 1 plus the carry
*                            bit is stored in
*                            Register B

        DSB  B,R7            R7 minus Register B minus
*                            1 plus the carry bit
*                            stored in R7
```

Syntax [<label>] EINT

Execution 1 → (Global interrupt enable bit)

Status Bits

I ← 1
C ← 1
N ← 1
Z ← 1

Description EINT simultaneously enables all interrupts. Since the interrupt enable flag is stored in the status register, the POP ST or RETI instructions may disable interrupts even though an EINT instruction has been executed. During the interrupt service, the interrupt enable bit is automatically cleared after the old status register value has been pushed onto the stack. Thus, the EINT instruction must be included inside the interrupt service routine to permit nested or multilevel interrupts.

Examples

```
LABEL    EINT    All interrupts are enabled.
```

Syntax

[<label>] IDLE

Execution

(PC) → (PC) until interrupt
(PC) + 1 → (PC) after return from interrupt

Status Bits

None

Description

The IDLE instruction causes the device to enter one of two low-power modes, which use a fraction of the normal operating power. In wake-up mode, the on-chip oscillator remains active, and activating the timer interrupt or the external interrupts ($\overline{\text{RESET}}$, $\overline{\text{INT1}}$, or $\overline{\text{INT3}}$) releases the device from the low-power mode. In halt mode, using the OSC-Off clock option, the oscillator and timers are disabled; the device can only be released from halt mode by an external interrupt or reset. Using the OSC-On clock option in halt mode, the oscillator continues to operate and only the timers are disabled; the device can only be released from halt mode by an external interrupt or $\overline{\text{RESET}}$.

For more information about low-power modes, see Section 3.5.

Examples

```
LABEL    IDLE
```

Syntax [<label>] INC <Rd>

Execution (Rd) + 1 → (Rd)

Status Bits
C 1 if (Rd) incremented from >FF to >00; 0 otherwise
N Set on result
Z Set on result

Description INC increments the value of any register. It is useful for incrementing counters into tables.

Examples

```
LABEL   INC   A          Increment Register A by 1
        INC   B          Register B is increased by 1
        INC   R43        Register 43 is increased by 1
```

Syntax

[<label>] INV <Rd>

Execution

NOT(Rd) → (Rd)

Status Bits

C ← 0
N Set on result
Z Set on result

Description

INV performs a logical or 1s complement of the operand. A 2's complement of the operand can be made by following the INV instruction with an increment (INC). A 1s complement reverses the value of every bit in the destination.

Examples

```
LABEL   INV  A        Invert Register A (0s become
*                     1s, 1s become 0s)

        INV  B        Invert Register B

        INV  R82      Invert register 82
```

Syntax

[<label>] JMP <offset>

Execution

(PC) + offset → (PC)
(The PC contains the address of the instruction immediately following the jump.)

Status Bits

None

Description

JMP jumps unconditionally to the address specified in the operand. The second byte of the JMP instruction contains the 8-bit relative address of the operand. The operand address must therefore be within –128 to +127 bytes of the location of the instruction following the JMP instruction. The assembler will indicate an error if the target address is beyond –128 to +127 bytes from the next instruction. For a longer jump the BR (branch) instruction can be used.

Examples

```
LABEL    JMP THERE     Load the PC with the address
*                      of THERE
```

Syntax

[<label>] J<cnd> <offset>

Execution

If tested condition is true, (PC) + offset → (PC)
(The PC contains the address of the instruction immediately following the jump.)

Status Bits

None

Description

Conditional Jump Instructions

Instruction	Mnemonic	C	N	Z
Jump if Carry	JC	1	X	X
Jump if Equal	JEQ	X	X	1
Jump if Higher or Same	JHS	1	X	X
Jump if Lower	JL	0	X	X
Jump if Negative	JN	X	1	X
Jump if No Carry	JNC	0	X	X
Jump if Not Equal	JNE	X	X	0
Jump if Non-zero	JNZ	X	X	0
Jump if Positive	JP	X	0	0
Jump if Positive or Zero	JPZ	X	0	1
Jump if Zero	JZ	X	X	1

Use the J<cnd> instructions after a CMP instruction to branch according to the relative values of the operands tested. After MOV, MOVP, LDA, or STA operations, a JZ or JNZ may be used to test if the value moved was equal to zero. JN and JPZ may be used in this case to test the sign bit of the value moved.

Examples

```
LABEL   JNC   TABLE     If the carry bit is clear,
*                       jump to TABLE

        JP    HERE      If the negative and zero flags
*                       are clear, jump to HERE

        JZ    NEXT      If the zero flag is set, jump
*                       to NEXT
```

Syntax [<label>] LDA <XADDR>

Execution (XADDR) → (A)

Status Bits

C ← 0
N Set on value loaded
Z Set on value loaded

Description LDA reads values stored anywhere in the full 64K-byte memory space. LDA uses three extended addressing modes:

- Direct addressing mode provides an efficient means of directly accessing a variable in memory.
- Indexed addressing gives an efficient table look-up capability for most applications.
- Indirect addressing allows the use of very large look-up tables and the use of multiple memory pointers since any pair of registers can be used as the pointer.

Examples

```
LABEL   LDA  @LABEL4          Direct addressing
        LDA  @LABEL5(B)       Indexed addressing
        LDA  *R13             Indirect addressing
```

Syntax [<label>] LDSP

Execution (B) → (SP)

Status Bits None

Description LDSP copies the contents of register B to the stack pointer register. Use LDSP to initialize the stack pointer.

Examples

```
LABEL    LDSP              Copy Register B to the
*                          Stack Pointer
```

Syntax [<label>] MOV <s>,<Rd>

Execution (s) → (Rd)

Status Bits

C ← 0
N Set on value loaded
Z Set on value loaded

Description MOV transfers values within the register space. Immediate values may be loaded directly into the registers. A MOV that uses register A or B as an operand produces shorter and quicker moves.

Examples

```
LABEL     MOV A,B              Move the contents of Register
  *                            A to Register B

          MOV  R32,R105        Move the contents of register
  *                            32 to register 105

          MOV  %>10,R3         Move >10 to register 3
```

Syntax

[<label>] MOVD <s>,<Rp>

Execution

(s) → (Rp)

Status Bits

C ← 0
N Set on MSb moved
Z Set on MSb moved

Description

MOVD moves a two-byte value to the register pair indicated by the destination register number. (Note that Rp should be greater than 0 or the MSb may be lost.) The destination points to the LSB of the destination register pair. The source may be a 16-bit constant, another register pair, or an indexed address. For the latter case, the source must be of the form "%ADDR(B)" where ADDR is a 16-bit constant or address. This 16-bit value is added (via 16-bit addition) to the contents of the B register, and the result placed in the destination register pair. This stores an indexed address into a register pair, for use later in indirect addressing mode. This is not to be confused with the extended addressing instruction @LABEL(B).

Examples

```
LABEL   MOVD  %>1234,R3        Load register pair R2,R3
*                               with >1234

        MOVD  R5,R3             Copy R4,R5 to R2,R3;
*                               R5,R3 = LSB

        MOVD  %TAB(B),R3        Load register pair R2,R3
*                               with the effective
*                               address of TAB + B
```

Syntax

[<label>] MOVP <s>,<Pd>
or
[<label>] MOVP <Ps>,<d>

Execution

(s) → (Pd)
or
(Ps) → (d)

Status Bits

C ← 0
N Set on value moved
Z Set on value moved

Description

MOVP transfers values to and from the peripheral file. This may be used to input or output 8-bit quantities on the I/O ports. The peripheral file also contains control registers for the interrupt lines, the I/O ports, and the timer controls. The operands supported by this instruction are A, B and %>iop.

During peripheral-file instructions, a peripheral-file port is always read before a write. The read can include output operations such as `MOVP A,P6`. If this read is undesirable because of hardware configuration, use a STA (Store A) instruction with the memory-mapped address of the peripheral register.

Examples

```
LABEL     MOVP A,P6          Move the contents of
*                            Register A to Port B

RDPORT    MOVP  P4,B         Move Port A data into
*                            Register B

LOADD     MOVP  %>12,P27     Move the hex value 12 into
                             Register 27
```

Syntax

[<label>] MPY <s>,<Rn>

Execution

(s) × (Rn) → (A,B) Result always stored in A,B

Status Bits

C ← 0
N Set on MSb of results (Register A)
Z Set on MSb of results (Register A)

Description

MPY performs an 8-bit multiply for a general source and destination operand. The 16-bit result is placed in the A, B register pair with the most significant byte in A. Multiplying by a power of two is a convenient means of performing double-byte shifts. If a double byte shift is three places or less, then it may be faster to use RLC or RRC instead of multiply. If a single byte needs shifting then it is almost always faster to use RLC or RRC.

Examples

```
LABEL   MPY R3,A          Multiply (R3) with (A), store
*                         result in A, B register pair

        MPY  %>32,B       Multiply >32 with (B), store
*                         in register pair A, B

        MPY  R12,R7       Multiply (R12) with (R7) and
*                         store in A, B register pair
```

Syntax [<label>] NOP

Execution (PC) + 1 → (PC)

Status Bits None

Description NOP is useful as a pad instruction during program development, to "patch out" unwanted or erroneous instructions or to leave room for code changes during development. It is also useful in software timing loops.

Examples

```
LABEL    NOP
```

Syntax

[<label>] OR <s>,<Rd>

Execution

(s) .OR. (Rd) → (Rd)

Status Bits

C ← 0
N Set on result
Z Set on result

Description

OR logically ORs the two operands. Each bit of the 8-bit result follows the truth table below. The OR operation is used to set bits in a register. If a register needs a 1 in the destination then a 1 is placed in the corresponding bit location in the source operand.

This is the truth table for the OR instruction:

Source Bit	Destination Bit	OR Result
0	0	0
0	1	1
1	0	1
1	1	1

Examples

```
LABEL   OR    A,R12          OR the A Register with R12,
*                            store in R12

        OR    %>0F,A         Set lower nibble of A to 1s,
*                            leave upper nibble unchanged

        OR    R8,B           OR (R8) with (B), store in B
```

Syntax

[<label>] ORP <s>,<Pd>

Execution

(s) .OR. (Pd) → (Pd)

Status Bits

C ← 0
N Set on result
Z Set on result

Description

ORP logically ORs the source operand with a peripheral-file location, and write the result back to the peripheral file. This may be used to set an individual I/O bit of a peripheral register. Since the peripheral register is read before it is ORed, it may not work with some peripheral locations which have a different function when reading than when writing.

Examples

```
LABEL   ORP   A,P39          OR (A) with (P39), store
*                            in P39

        ORP   B,P90          OR (B) with (P90), store
*                            in P90
```

Syntax

[<label>] POP <Rd>

Execution

(Stack top) → (Rd)
(SP) – 1 → (SP)
(Move value then decrement SP)

Status Bits

C ← 0; Or restored from stack on a POP ST instruction.
N Set on value POPed
Z Set on value POPed

Description

POP pulls a value from the top of the stack. The data stack can be used to save or pass values, especially during subroutines and interrupt service routines.

The status register may be replaced with the contents on the stack by the statement `POP ST`. This one-byte instruction is usually executed in conjunction with a previously performed `PUSH ST` instruction.

Examples

```
LABEL   POP   R32              Load R32 with top of stack

        POP   ST               Load Status Register with
*                              top of stack
```

Syntax

[<label>] PUSH <Rs>

Execution

(SP) + 1 → (SP)
(Rs) → (top of stack)
(Increment SP then move value)

Status Bits

C ← 0, Or not affected by a PUSH ST instruction.
N Set on value PUSHed
Z Set on value PUSHed

Description

PUSH places a value on the top of the stack. The data stack is used to save or pass values, especially during subroutines and interrupt service routines.

The status register may be pushed on the stack with the statement `LABEL PUSH ST`. This one-byte instruction is usually executed in conjunction with a subsequently performed `POP ST` instruction. The status register is unaffected.

Examples

```
LABEL   PUSH  A          Move (A) to top of stack
        PUSH  ST         Move status to top of stack
```

Syntax

[<label>] RETI

Execution

((SP)) → (PC LSB)
(SP) – 1 → (SP)
((SP)) → (PC MSB)
(SP) – 1 → (SP)
((SP)) → (ST)
(SP) – 1 → (SP)

Status Bits

Status register is loaded from the stack

Description

RETI is typically the last instruction in an interrupt service routine. RETI restores the status register to its state immediately before the interrupt occurred and branches back to the program at the instruction boundary where the interrupt occurred. Registers A and B, if used, must be restored to original values before the RETI instruction.

Examples

```
LABEL    RETI          Restore to program control
```

Syntax [<label>] RETS

Execution

((SP)) → (PC LSB)
(SP) – 1 → (SP)
((SP)) → (PC MSB)
(SP) – 1 → (SP)

Status Bits None

Description RETS is typically the last instruction in a subroutine. RETS branches to the location immediately following the subroutine call instruction. In the called subroutine there must be an equal number of POPs and PUSHes so that the stack is pointing to the return address and not some other data.

Examples

```
LABEL    RETS          Return to program control
```

Syntax

[<label>] RL <Rd>

Execution

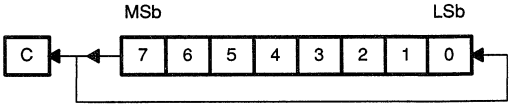
Bit(n) → Bit(n+1)
Bit(7) → Bit(0) and carry

Status Bits

C Set to bit 7 of the original operand
N Set on result
Z Set on result

Description

RL circularly shifts the destination contents one bit to the left. The MSb is shifted into the LSb; the carry bit is also set to the original MSb value.



For example, if register B contains the value >93, then RL changes the contents of B to >27 and sets the carry bit.

Examples

```
LABEL   RL  R102
        RL  A
        RL  B
```

Syntax

[<label>] RLC <Rd>

Execution

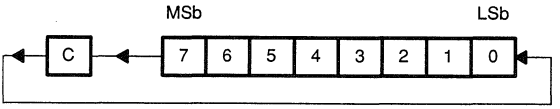
Bit(n) → Bit(n+1)
Carry → Bit(0)
Bit(7) → Carry

Status Bits

C Set to bit 7 of the original operand
N Set on result
Z Set on result

Description

RLC circularly shifts the destination contents one bit to the left and through the carry. The original carry bit contents shift into the LSb, and the original MSb shifts into the carry bit.



For example, if register B contains the value >93 and the carry bit is a zero, then the RLC instruction changes the operand value to >26 and the carry to one.

Rotating left effectively multiplies the value by 2. Using multiple rotates, any power of 2 (2, 4, 8, 16,...) can be achieved. This type of multiply is usually faster than the MPY (multiply) instruction. This instruction is also useful in rotates where a value is contained in more than one byte such as an address or in multiplying a large multibyte number by 2. Care must be taken to assure that the carry is at the proper value. The SETC or CLRC instructions may be used to setup the correct value.

Examples

```
LABEL  RLC  R72
       RLC  A
       RLC  B
```

Syntax

[<label>] RR <Rd>

Execution

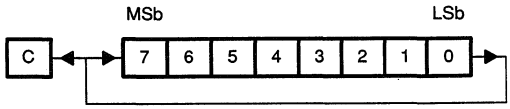
Bit(n+1) → Bit(n)
Bit(0) → Bit (7) and carry

Status Bits

C Set to bit 0 of the original value
N Set on result
Z Set on result

Description

RR circularly shifts the destination contents one bit to the right. The LSb is shifted into the MSb, and the carry bit is also set to the original LSb value.



For example, if register B contains the value >93, then the "RR B" instruction changes the contents of B to >C9 and sets the carry status bit.

Examples

```
LABEL    RR  A
```

Syntax

[<label>] RRC <Rd>

Execution

Bit(n+1) → Bit(n)
Carry → Bit(7)
Bit(0) → Carry

Status Bits

C Set to bit 0 of the original value
N Set on result
Z Set on result

Description

RRC circularly shifts the destination contents one bit to the right through the carry. The carry bit contents shift into the MSb, and the LSb is shifted into the carry bit.

For example, if register B contains the value >93 and the carry bit is zero, then RRC changes the operand value to >49 and sets the carry bit.

When the carry is 0 this instruction effectively divides the value by two. A value of >80 becomes >40. By using this instruction once more, the value can be divided by any power of two. Care must be taken to assure the correct value in the carry bit.

Examples

```
LABEL    RRC R32
```

Syntax

[<label>] SBB <s>,<Rd>

Execution

(Rd) – (s) – 1 + (C) → (Rd)

Status Bits

C Set to 1 if no borrow; 0 otherwise
N Set on result
Z Set on result

Description

SBB performs multiprecision 2's complement subtraction. An SBB instruction with an immediate operand of zero value is equivalent to a conditional decrement of the destination operand. If (s)=0 and (C)=0 then (Rd) is decremented, otherwise it is unchanged. A borrow occurs if the result is negative. In this case, the carry bit is set to 0. The carry bit can be thought of as the "no-borrow" bit.

Examples

```
LABEL   SBB  %>23,B          Subtract (B) from >23, sub-
*                           tract 1, add the carry bit;
*                           store in Register B

        SBB  B,A            (B) minus (A) minus 1 plus
*                           the carry bit is stored
*                           in Register A

        SBB  %>33,R6        Subtract (R6) from >33, sub-
*                           tract the inverse of the
*                           carry bit
```

Syntax [<label>] SETC

Execution 1 → (C)

Status Bits

C ← 1
N ← 0
Z ← 1

Description SETC sets the carry flag (if required) before an arithmetic or rotate instruction.

Examples

```
LABEL    SETC
```

Syntax [<label>] STA <XADDR>

Execution (A) → (XADDR)

Status Bits

C ← 0
N Set on value loaded
Z Set on value loaded

Description STA stores values anywhere in the 64K-byte memory address space. STA uses three extended addressing modes:

- Direct addressing provides an efficient means of directly accessing a variable in memory.
- Indexed addressing provides efficient table look-up.
- Indirect addressing allows the use of very large look-up tables and the use of multiple memory pointers since any pair of registers can be used as the pointer.

Examples

```
LABEL   STA   @VALUE          Direct addressing
        STA   @TABLE(B)       Indexed addressing
        STA   *R13            Indirect addressing
```

Syntax [<label>] STSP

Execution (SP) → (B)

Status Bits None

Description STSP copies the SP to register B. This instruction can be used to test the stack size. The indexed addressing mode may be used to reference operands on the stack. For example, `STSP`; then `LDA @>0000(B)` will put the present value on top of the stack into register A.

Examples

```
LABEL    STSP          Copy the SP to Register B
```

Syntax

[<label>] SUB <s>,<Rd>

Execution

(Rd) – (s) → (Rd)

Status Bits

C Set to 1 if result ≥ 0, otherwise set to 0
N Set on result
Z Set on result

Description

SUB performs 2's complement subtraction. The carry bit is set to 0 if a borrow is required. The carry bit could be renamed a "no-borrow" bit in this case.

Examples

```
LABEL   SUB  R19,B       (B) minus (R19) is
*                       stored in B

        SUB  %>76,A     (A) minus >76 is stored
*                       in A

        SUB  R4,R9      (R9) minus (R4) stored
*                       in R9
```

Syntax [<label>] SWAP <Rn>

Execution Bits (7,6,5,4, / 3,2,1,0) → Bits (3,2,1,0, / 7,6,5,4)

Status Bits

C Set to bit 0 of the result or bit 4 of the original
N Set on results
Z Set on results

Description SWAP exchanges the first four bits with the second four bits. This instruction is equivalent to four consecutive RL (rotate left) instructions. It manipulates four bit operands, especially useful for packed BCD operations.

Examples

```
LABEL   SWAP   R45          Switch Lo and Hi nibbles of R45
        SWAP   A            Switch Lo and Hi nibbles of A
        SWAP   B            Switch Lo and Hi nibbles of B
```

Syntax

[<label>] TRAP <n> where n = 0–23

Execution

(SP) + 1 → (SP)
(PC MSB) → ((SP))
(SP) + 1 → (SP)
(PC LSB) → ((SP))
(Entry vector) → (PC)

Status Bits

None

Description

Trap is a one-byte subroutine call. The operand <n> is a trap number which identifies a location in the trap vector table, addresses >FFD0 to >FFFF in memory. The contents of the two-byte vector location form a 16-bit trap vector to which a subroutine call is performed. TRAP is an efficient way to invoke a subroutine. The highest block of memory is the trap vector table, and can contain up to 23 subroutine addresses. The subroutine addresses are stored like all other addresses in memory, with the least significant byte in the higher-addressed location, as shown below.

Trap Vector Table

>FFD0	Trap 23 address	MSB
>FFD1	Trap 23 address	LSB
:	:	:
>FFE0	Trap 15 address	MSB
>FFE1	Trap 15 address	LSB
:	:	:
>FFFA	Trap 2 address	MSB
>FFFB	Trap 2 address	LSB
>FFFC	Trap 1 address	MSB
>FFFD	Trap 1 address	LSB
>FFFE	Trap 0 address	MSB
>FFFF	Trap 0 address	LSB

Note that TRAPs 0, 1, 2, and 3 correspond to the hardware-invoked interrupts 0, 1, 2, and 3, respectively. The hardware-invoked interrupts, however, push the program counter and the status register before branching to the interrupt routine, while the TRAP instruction pushes only the program counter. TRAP 0 will branch to the same code executed for a system reset but will not set or clear all the registers like the hardware RESET.

Examples

```
LABEL    TRAP 15
```

Syntax [<label>] TSTA

Execution C,N,Z bits set

Status Bits

C ← 0
N Set on value in register A
Z Set on value in register A

Description TSTA sets the status bits according to the value in register A. This instruction is equivalent to the CLRC (clear carry) instruction.

Examples

```
LABEL    TSTA     Test Register A
```

Syntax [<label>] TSTB

Execution C,N,Z bits set

Status Bits

C ← 0
N Set on value in register B
Z Set on value in register B

Description TSTB sets the status bits according to the value in register B. It may be used to clear the carry bit. This instruction is equivalent to the XCHB B (exchange B with B) instruction.

Examples

```
LABEL    TSTB          Test Register B
```

Syntax [<label>] XCHB <Rn>

Execution (B) ←→ (Rn)

Status Bits C ← 0
N Set on original contents of B
Z Set on original contents of B

Description XCHB exchanges a register with register B without going through an intermediate location. The XCHB instruction with the B register as the operand is equivalent to the TSTB instruction.

Examples

```
LABEL   XCHB  A       Exchange Register B with
*                     Register A

        XCHB  R3      Exchange Register B with R3
```

Syntax

[<label>] XOR <s>,<Rd>

Execution

(s) .XOR. (Rd) → (Rd)

Status Bits

C ← 0
N Set on result
Z Set on result

Description

XOR performs a bit-wise exclusive OR operation on the operands. The XOR instruction can be used to complement bits in the destination operand. Each bit of the 8-bit result follows the truth table below. This operation can also toggle a bit in a register. If the bit value in the destination needs to be the opposite from what it currently is, then the source should contain a 1 in that bit location.

This is the truth table for the XOR instruction:

Source Bit	Destination Bit	XOR Result
0	0	0
0	1	1
1	0	1
1	1	0

Examples

```
LABEL   XOR   R98,R125      XOR (R98) with (R125),
*                          store in R125

        XOR   %>1,R20       Toggle bit 0 in R20

        XOR   B,A           XOR (B) with (A), store
*                          in A
```

Syntax

[<label>] XORP <s>,<Pd>

Execution

(s) .XOR. (Pd) → (Pd)

Status Bits

C ← 0
N Set on result
Z Set on result

Description

XORP performs a bit-wise exclusive OR operation on the operands. The XORP instruction can be used to complement bits in the destination PF register. Since the peripheral register is read before it is XORed, it may not work with some peripheral locations which have a different function when reading than when writing.

Examples

```
LABEL  XORP  %>01,P9          Invert bit 0 of P9 (Port C
*                             DDR); this inverts the
*                             direction of the pin

       XORP  %>AA,P29         Toggle odd bits of P29

       XORP  B,P99            XOR (B) with (P99), store
*                             in P99
```

6.4 Software Compatibility

6.4.1 TMS70C42 and TMS70C82 Directly Compatible

- Fully software and pin compatible
- Peripheral map identical

6.4.2 TMS70C20, TMS70C40, TMS70CT20, TMS70CT40

- Software intended for mask in low cost devices can still be run on the TMS77C82 for prototypes/preproduction.
- Initialization
  - (P2) UART and Timer 2 interrupts must be disabled
  - (P1) External interrupts must be set for –ve edge only
  - (P5) A port must be set for input only
  - (P12) Timer 1 MSB reload register must be 00
  - (P14) Timer 1 Toggle output must be disabled
  - (P15) Timer prescale latch must be loaded with 4 × original value software
    Timer control register equates to P15 rather than P3
    Timer data register equates to P13 rather than P2

Chapter 7

Linking Program Modules

The TMS7000 assembler creates both absolute and relocatable object code that can be linked to form executable programs from separately assembled modules. An entire program need not be assembled at one time. A long program can be divided into separately assembled modules, avoiding a long assembly and reducing the symbol table size. Caution must be observed when assembling a long program with excessive labels; this may cause an assembler error from symbol table overflow. Modules that are common to several programs can be assembled once and accessed when needed. These separately-generated modules can be linked together by the link editor, forming a single linked object module that is stored in a library and/or loaded as required.

The ***Link Editor User's Guide*** (literature number SPNU037) contains a complete description of the link editor, related files, linker commands, linking examples, and error messages. This chapter provides all the information that most TMS7000 users need to link program modules.

Section **Page**

7.1 Relocation Capability 7-2
7.2 Link Editor Operation 7-4
7.3 Directives Used for Linking 7-6
7.4 Creating Linkable Files 7-7
7.5 Linking Files – Examples 7-12

7.1 Relocation Capability

Absolute code is appropriate for code that must be placed in dedicated areas of memory. It must always be loaded into the same memory area.

Relocatable code includes information that allows a loader to place the code in any available memory area, allowing the most efficient use of available memory.

Object code generated by an assembler contains machine language instructions, addresses, and data. The code may include **absolute** segments, **program-relocatable** segments, **data-relocatable** segments, and numerous **common-relocatable** segments. In assembly language source programs, symbolic references to locations within a relocatable segment are called *relocatable addresses.* These addresses are represented in the object code as displacements from the beginning of a specified segment. A *program-relocatable address*, for example, is a displacement into the program segment. At load time, all program-relocatable addresses are adjusted by a value equal to the load address. *Data-relocatable addresses* are represented by a displacement into the data segment. There may be several types of *common-relocatable addresses* in the same program, since distinct common segments may be relocated independently of each other.

Expressions may contain more than one symbol that is not previously defined. Expressions on either side of a multiplication or division symbol must be absolute; if they are relocatable, the expression is illegal. An expression in which the number of relocatable symbols or constants added to the expression exceeds the number of relocatable symbols or constants subtracted from the expression by more than one is illegal. That is, if:

NA = Number of relocatable values added, and

NS = Number of relocatable values subtracted

Then, if NA – NS =

0	The expression is absolute
1	The expression is relocatable
Neither	The expression is illegal

An expression containing relocatable symbols or constants of several different relocation types is absolute *if* it is absolute with respect to all relocation types. If it is relocatable with respect to one relocation type and absolute with respect to all other relocation types, it is relocatable.

Examples of valid expressions include:

BLUE+1	The value of symbol BLUE + 1
GREEN–4	The value of symbol GREEN – 4

2*16+RED 2 times 16 plus the value of symbol RED

440/2–RED 440 divided by two less the value of symbol RED. Red must be **absolute**.

Decimal, hexadecimal, and character constants are absolute. Assembly-time constants defined by absolute expressions are absolute, and assembly-time constants defined by relocatable expressions are relocatable.

Any symbol that appears in the label field of a source statement (other than an EQU directive) is **absolute** when the statement is in an *absolute block* of the program. Any symbol that appears in the label field of a source statement (other than an EQU directive) is **relocatable** when the statement is in a *relocatable block* of the program. The type of the label or an EQU directive is the type of an expression in an operand field.

7.2 Link Editor Operation

The link editor combines separate modules to produce a single linked output module. It resolves externally referenced symbols and definitions created by the REF and DEF directives. Without this function, all modules would have to be compiled or assembled at once. The link editor builds a list of symbols from the REF tags in the object modules that are to be included in the linking process. The link editor then resolves the references by matching DEF tag symbols with the REF tags and inserting the correct values for these symbols in the linked object code.

A **link control file**, which must be created before the assembly, controls the link editor operation. The link control file contains a set of link control commands (control stream) that direct the link editor in combining various object modules. Table 7–1 summarizes the linker commands most often used to link TMS7000 program modules.

The link control commands define which modules are to be linked and how they are to be linked. The link editor automatically resolves the REF and DEF tag symbols between object modules specified in the INCLUDE commands. The link editor links the object modules in the order specified by the link control commands. Thus, the structure of the control stream determines the structure of the linked object module.

Table 7–1. Linker Commands Used to Link TMS7000 Program Modules

Command	Syntax and Description
COMMON	Syntax: COMMON {<base>[,<name>][,<name>]..} Defines the starting address for the specified common segment (CSEG). Commons that are loaded at the specified address must be specifically identified within this command. COMMON is only valid when used with PROGRAM. <base> is the starting location of the common segment. It can be a decimal or a hexadecimal number. <name> is the name of the common segment.
DATA	Syntax: DATA <base> Defines the absolute starting address for the data segment (DSEG) in the linked output. DATA is only valid when used with PROGRAM. <base> is the starting location of the data segment.
END	Syntax: END Indicates the end of the link control stream. This command is required in every link control file.
INCLUDE	Syntax: INCLUDE {<acnm>[,<acnm>]...,(<name>) [,(<name>)]...} Defines one or more modules to be included in the linking process. This is a required command. More than one INCLUDE statement may be used. <acnm> is the access name of a file containing the object module(s) to be included in the linking process, and (<name>) is a member in a library.
PROGRAM	Syntax: PROGRAM <base> Defines the absolute starting address for the program segment (PSEG) in the linked output. <base> is the starting location of the program segment.
TASK	Syntax: TASK [<name>] Defines the name of the task; this becomes the IDT name, placed on the last record of the object module. <name> is the task module identifier, and can have up to eight characters. If omitted, the IDT name of the first included module is used as the task name.

7.3 Directives Used for Linking

The most commonly used assembler directives which are used in the linking process are shown below:

REF Declares symbols within a program module which are referred to by that module, but are not defined within that module (such a label would normally produce an assembly error if not declared). Used in conjunction with DEF.

DEF Declares symbols within a program module which are defined within that module (for example, a label), but which are known to be referred to by one or more other modules. Used in conjunction with REF.

PSEG Marks the beginning of the main program segment within a module which will be linked with the program segments of other modules and allocated an absolute address.

DSEG Marks the beginning of a data segment within a module which can be linked with data segments in other modules and allocated an absolute address. This is normally used to locate the reset and trap vectors in the correct position in memory.

CSEG Allows named sections to be marked within a module which can be linked with sections with the same name in other modules and allocated an absolute address. This is normally used for pulling together look-up tables from several modules and locating them in a specific area of memory.

For more information about directives, see Chapter 5, Assembler Directives.

7.4 Creating Linkable Files

Figure 7–1 shows an absolute source file of the form described in Chapter 5.

Figure 7–1. Absolute Source File Example

```
*
*  FOR USE WITH XASM7 PC ASSEMBLER
*  this example is for use with a TMS7XCX2
*  it exercises ports A,B,C & D
*
          IDT     'EXAMPLE'    EMBED NAME 'EXAMPLE' IN OBJECT
*
LABEL     EQU     R25          )
APORT     EQU     P4           )      READABLE NAMES ASSIGNED TO
BPORT     EQU     P6           )      REGISTER AND PORT LOCATIONS
CPORT     EQU     P8           )
DPORT     EQU     P10          )
*
          AORG    >F806        ABSOLUTE START ADDRESS ASSIGNED
*
START     EQU     $            LABEL 'START' ASSIGNED TO BEGINNING
          MOVP    %>FF,P5      APORT SET TO ALL OUTPUTS
          MOVP    %>FF,P9      CPORT SET TO ALL OUTPUTS
          MOVP    %>FF,P11     DPORT SET TO ALL OUTPUTS
LOOP      CLR     LABEL        REGISTER 'LABEL' CLEARED
          CALL    @OUT         OUTPUT ROUTINE CALLED
          INV     LABEL        REGISTER 'LABEL' SET TO ALL 1's
          CALL    @OUT         OUTPUT ROUTINE CALLED
          JMP     LOOP         REPEAT FROM 'LOOP'
*
*  output routine ( called from main program )
*
OUT       MOV     LABEL,A      MOVE CONTENTS OF 'LABEL' INTO 'A'
          MOVP    A,APORT      OUTPUT ACCUMULATOR TO APORT PINS
          MOVP    A,BPORT      OUTPUT ACCUMULATOR TO BPORT PINS
          MOVP    A,CPORT      OUTPUT ACCUMULATOR TO CPORT PINS
          MOVP    A,DPORT      OUTPUT ACCUMULATOR TO DPORT PINS
          RETS                 RETURN TO MAIN ROUTINE
*
*
          AORG    >FB00
          BYTE    >55          LOOK-UP TABLE
          BYTE    >55
          BYTE    >55
*
*  load reset vector with address of start of program
*
          AORG    >FFFE
          DATA    START
*
          END
```

Figure 7–2, Figure 7–3, and Figure 7–4 show how the above source file can be split into three relocatable files (no absolute addresses assigned) by using the assembler directives described above.

Figure 7–2. PROG1.ASM

```
*
*  FOR USE WITH XASM7 PC ASSEMBLER
*
          IDT       'PROG1'       EMBED NAME 'PROG1' IN OBJECT
*
          REF       OUT           'OUT' NOT DEFINED IN THIS MODULE
          DEF       START         REFERRED TO BY PROG3
*
          COPY      REGDEF.ASM    COPY IN REGISTER DEFINITIONS
*
          PSEG                    START OF RELOCATABLE PROGRAM SEGMENT
*
START     EQU       $             LABEL 'START' ASSIGNED TO BEGINNING
          MOVP      %>FF,P5       APORT SET TO ALL OUTPUTS
          MOVP      %>FF,P9       CPORT SET TO ALL OUTPUTS
          MOVP      %>FF,P11      DPORT SET TO ALL OUTPUTS
LOOP      CLR       LABEL         REGISTER 'LABEL' CLEARED
          CALL      @OUT          OUTPUT ROUTINE CALLED  ( IN PROG2 )
          INV       LABEL         REGISTER 'LABEL' SET TO ALL 1's
          CALL      @OUT          OUTPUT ROUTINE CALLED  ( IN PROG2 )
          JMP       LOOP          REPEAT FROM 'LOOP'
*
          END
```

Figure 7–3. PROG2.ASM

```
*
*  FOR USE WITH XASM7 PC ASSEMBLER
*
          IDT       'PROG2'       EMBED NAME 'PROG2' IN OBJECT
*
          COPY      REGDEF.ASM    COPY IN REGISTER DEFINITIONS
*
          DEF       OUT           REFERRED TO BY PROG1
*
*  output routine ( called from main program )
*
OUT       MOV       LABEL,A       MOVE CONTENTS OF 'LABEL' INTO 'A'
          MOVP      A,APORT       OUTPUT ACCUMULATOR TO APORT PINS
          MOVP      A,BPORT       OUTPUT ACCUMULATOR TO BPORT PINS
          MOVP      A,CPORT       OUTPUT ACCUMULATOR TO CPORT PINS
          MOVP      A,DPORT       OUTPUT ACCUMULATOR TO DPORT PINS
          RETS                    RETURN TO MAIN ROUTINE
*
          END
```

Figure 7–4. PROG3.ASM

```
*
*   FOR USE WITH XASM7 PC ASSEMBLER
*
          IDT       'PROG3'       EMBED NAME 'PROG3' IN OBJECT
*
          REF       START         NOT DEFINED IN THIS MODULE
*
          CSEG      'TABLE'       START OF SEGMENT 'TABLE'
*
          BYTE      >55           LOOK-UP TABLE
          BYTE      >55
          BYTE      >55
*
*   load reset vector with address of start of program
*
          DSEG                    START OF DATA SEGMENT
*
          DATA      START
*
          END
```

When each of these files are assembled, records are embedded in the object modules which indicate that certain references have not yet been resolved. Figure 7–5 shows the register definition file which has been copied into the relevant files at the time of assembly. Note that REGDEF.ASM does not have to be assembled separately.

Figure 7–5. REGDEF.ASM

```
LABEL   EQU     R25       )
APORT   EQU     P4        )     READABLE NAMES ASSIGNED TO
BPORT   EQU     P6        )     REGISTER AND PORT LOCATIONS
CPORT   EQU     P8        )
DPORT   EQU     P10       )
```

When the linker is executed, the link control file LINK.CTL (Figure 7–6) is invoked which joins together the relevant object code segments within the program modules and allocates the appropriate absolute addresses.

In this example the main program segments in PROG1 and PROG2 are joined together and located at >F806. The named section TABLE is then linked in from PROG3 and located at >FB00. Finally, the data segment containing the reset vector is pulled in from PROG3 and located at >FFFE.

During the linking process, all REF and DEF declarations are evaluated such that any label references between modules are resolved.

Figure 7–6. LINK.CTL

```
PHASE     0,LINK
PROGRAM   >F806
INCLUDE   PROG1.MPO
INCLUDE   PROG2.MPO
INCLUDE   PROG3.MPO
COMMON    >FB00,TABLE
DATA      >FFFE
END
```

The resulting single object file corresponds to exactly the same code as that was produced by the assembly of the absolute source file in Figure 5–1.

The linker also produces a link map which shows the final locations and sizes of the linked modules together with resolved reference information. Any errors or unresolved references found during the linkage will be highlighted within this file (Figure 7–7).

Figure 7–7. LINK.MAP

```
PC/CrossWare Family Linker v3.1 88.005                                    Page   1
Copyright (C) 1986, 1987 Texas Instruments Inc.  All Rights Reserved
Command List

PHASE     0,LINK
PROGRAM   >F806
INCLUDE   PROG1.MPO
INCLUDE   PROG2.MPO
INCLUDE   PROG3.MPO
COMMON    >FB00,TABLE
DATA      >FFFE
END
PC/CrossWare Family Linker v3.1 88.005                                    Page   2
Copyright (C) 1986, 1987 Texas Instruments Inc.  All Rights Reserved
Link Map

Control File = LINK.CTL

Linked Output File = LINK.LOD

List File = LINK.MAP

Output Format = ASCII

PC/CrossWare Family Linker v3.1 88.005                                    Page   3
Copyright (C) 1986, 1987 Texas Instruments Inc.  All Rights Reserved

Phase 0     LINK      Module  Origin = 0000    Length = 0000

Module    No   Origin      Length    Type         Date        Time      Creator

PROG1      1    F806*      0015      INCLUDE                            ASMMLP
PROG2      2    F81B*      000B      INCLUDE                            ASMMLP
PROG3      3    F826*      0000      INCLUDE                            ASMMLP
$DATA      3    FFFE*      0002
```

```
Common     No    Origin     Length
TABLE       3     FB00*      0003
                          D E F I N I T I O N S
Name     Value No     Name     Value No     Name     Value No     Name     Value No
OUT      F81B*  2     START    F806*  1
Length of Region for Task              =  0000
Number of Records for Module LINK      =    3
Total Records Written                  =    3
**** Linking Completed
```

7.5 Linking Files – Examples

The linker is invoked by executing LINKER.EXE. The standard convention for filenames is

`FILENAME.CTL`	Link Control File
`FILENAME.MAP`	Listing File
`FILENAME.LOD`	Final Object File

By executing LINKER FILENAME; the assembler will assume a link control file input of FILENAME.CTL and produce output files with the same name and default extensions .MAP and .LOD. If the ";" is omitted, the assembler will prompt for user defined output file names.

Chapter 8

Macro Language

The TMS7000 macro assembler supports a macro definition language. Macro definitions allow you to create your own commands. This is especially useful when a program executes a particular task several times. A macro definition contains source statements that are associated with a unique macro name. When the macro name is used as an opcode in a program source statement (referred to as a *macro call*), the macro definition's predefined source statements are substituted for the macro call statement.

This section discusses the following topics:

Section		Page
8.1	Defining Macros	8-2
8.2	Strings, Constants, and Operators	8-7
8.3	Variables	8-9
8.4	Keywords	8-13
8.5	Assigning Values to Parameters	8-15
8.6	Verbs	8-18
8.7	Model Statements	8-28
8.8	Macro Examples	8-29
8.9	Macro Error Messages	8-32

8.1 Defining Macros

A macro definition begins with a source statement like this:

```
<MACNAME>  $MACRO [<parm1>,<parm2>...] [<comment>]
```

where:

<MACNAME>	Names the macro; it may contain a maximum of six alphanumeric characters. It is placed in the source statement's label field.
$MACRO	Identifies this source statement as the first line of a macro definition; it appears in the opcode field.
<parms>	Parameters passed to the macro when called (not all macros will have parameters); they appear in the operand field.
<comment>	Optional.

There are three methods for defining macros:

1) Macros can be defined in the **source file** where they are used. Macros must be defined before they are called; it is good practice to place all the definitions at the top of the file. This provides easy reference to all the definitions because they are in one location.

2) Macros can also be defined in external files. These files are simply text files, like the assembler source file. Only one macro may be defined per external file. These external macro definition files are collected to form a **macro library**.

3) All macros can be placed in one file without the source program, and then the COPY directive can be used to include the macro file in the source program.

8.1.1 Using Macro Libraries

When a macro is called, the assembler searches several places for its definition. Let's assume that the directory file 'VOLUME.DIRECTORY.MACLIB' contains a library of macro definitions. The MLIB directive tells the assembler that a macro library exists. The MLIB directive syntax is:

```
MLIB  'VOLUME.DIRECTORY.MACLIB'
```

The quoted string names the macro library. (This string represents a directory name in the host operating system format.)

This library contains a definition for a macro named CPXADD. Assume that an assembly language source program contains the following macro call:

```
LABEL   CPXADD CX1,CX2
```

The assembler uses the following search order to find the macro definition:

1) The **in-memory macro table** is the first place searched. CPXADD will be in the macro table if:
   a) It was previously defined in the assembler source file or
   b) It has already been read from a macro file.
2) If CPXADD is not found in the macro table, the assembler searches the normal **assembler opcode/directive table.** If found there, the opcode will be assembled as a normal machine instruction.
3) If the definition is not in the opcode/directive table, **the macro name is appended to the macro library name.**

If more than one MLIB directive was encountered, the assembler searches the most recently defined library first, then the library defined before that, and so on.

If the file is found, the macro definition is copied into the assembler's macro file (in a compressed format), and an entry is made in the macro table for later use.

The search order prevents a macro defined in a library from automatically redefining a machine instruction because the assembler searches the opcode table before the libraries. This can be circumvented in two ways:

1) Define the macro in the source program or
2) Include another file in the macro library called an MLIST (macro list).

An MLIST file is a text file that contains the names of the opcodes and currently defined macros that are redefined by macros in the library.

A typical MLIST file might be constructed as follows; note that there is only one definition per line and each statement begins in column one.

```
file named  <MLIB directory name>.MLIST
record 1    ADD                 (opcode)
record 2    LACK                (opcode)
record 3    MOV                 (opcode)
record 4    FSUB                (macro)
eof                             (MLIST)
```

The MLIST is read (if provided) when the MLIB directive is processed. If a name found there matches a currently defined opcode or a name in the macro table, the matching entry is removed from its table. This forces a search of the libraries, since the name will not be found elsewhere. The following message is printed when a name is found that matches an opcode:

```
' ****  OPCODES REDEFINED'
```

The message appears after the printing of the MLIB statement. A similar message:

```
' **** MACROS REDEFINED'
```

appears when currently defined macros are redefined. If you do not want an opcode or macro to be redefined, you must delete the appropriate records from the MLIST file.

The name of a macro in a file should be the same as the file name, or the macros are not used efficiently. If the file named CPXADD contains a definition line such as

```
CPXMUL  $MACRO  MR, MD
```

the macro CPXMUL is entered into the macro table, and the next to CPXADD will be undefined and re-entered into the macro table as CPXMUL.

8.1.1.1 Using Macro Libraries on MS/PC-DOS Systems

The following program segment suggests a method for using macro libraries on an MS/PC-DOS system.

```
      MLIB 'E:'        The pathname must be a drive name
      ADD  R3,R4       Typical assembly code
      MOV  R6,R9             .
        .    .               .
        .    .               .
*
      XMAC             First macro call
*
      NOP
*
      YMAC             Another macro call
*
      NOP
      END
```

The assembler searches the drive specified by the MLIB directive for a file with the same name as the macro. The macro name cannot have an extension. Only one macro is allowed per file.

The assembler searches the current MS/PC-DOS directory structure for the drive specified in the MLIB directive. A possible example of macro library use is:

- Store all macros on the **A** drive in a directory named MACROS.
- Store the TMS7000 assembler on the **E** drive (or any drive other than A) in a directory named PROGRAMS. The assembler program name is XASM7.EXE.
- Store the source program on the **E** drive in a directory named ASSEMBLY. The source program name is CODE.ASM. It includes this directive statement:

  ```
  MLIB 'A:'
  ```

- Issue a path statement that includes the program directory:

  ```
  PATH E:\;E:\MSDOS;E:\PROGRAMS
  ```

- The following batch file will assemble the program:

  ```
  E:                  Insure execution from drive E:
  CD A:\MACROS        Change A: drive's directory
  ```

```
CD E:\ASSEMBLY      Change E: drive's directory
XASM7 CODE.ASM;     Assemble the file CODE.ASM
```

8.1.2 Sample Macros

Assume that a symbol representing a memory address, ADR, is set in a source file:

```
ADR     EQU     >F000
```

This is a simple example of a macro definition that increments ADR:

```
INCADR  $MACRO
        LDA     @ADR
        INC     A
        STA     @ADR
        $END
```

where:

`INCADR` Names a macro, INCADR.

`$MACRO` Identifies the beginning of the macro definition.

```
LDA   @ADR
INC   A
STA   @ADR
```

Are model statements that are substituted into the source program when the macro is called. A model statement "models" an assembler language statement. Such a statement is (or will form after macro substitution) a legal language statement.

`$END` Identifies the end of the macro definition.

The macro INCADR can now be used in the source program as often as necessary. Call the macro by entering the following line into the source file:

```
INCADR
```

The macro assembler replaces this line with the macro definition:

```
LDA   @ADR
INC   A
STA   @ADR
```

INCADR is limited because the macro can only be used with a single memory location, ADR. The following macro uses parameters and is more flexible. It can be used with any memory location.

```
INC   $MACRO  M
LDA   @:M.S:
INC   A
STA   @:M.S:
$END
```

where:

M Is a *macro parameter.* It is replaced by the actual parameter when the macro is called.

M.S Is the string component of this variable (the symbol representation of the variable).

For example, the line:

```
INC         Y
```

will be replaced by:

```
LDA         @Y
INC         A
STA         @Y
```

but

```
INC         DATA4
```

will be replaced by:

```
LDA         @DATA4
INC         A
STA         @DATA4
```

8.2 Strings, Constants, and Operators

Macro language literal **strings** are identical to the character strings used by TMS7000 assembly language. The strings contain one or more characters enclosed in single quotes.

Examples of valid strings are:

- 'ONE'
- ' ' (a blank)

Macro language **constants** are defined in the same manner as assembly language constants.

Examples of valid constants are:

- >9F3C
- $ (current PC value)

Arithmetic operators can be used in operands. Functions of +, –, * (multiply), and / (divide) can be used to generate operand values. Examples using arithmetic operators are:

```
LABEL   EQU   $+4  (current PC value + 4)
```

Relational operators can also be used. Relational operators compare the values of two variables or constants and return the answer of *true* or *false*. The relational operators are:

- = Equal
- > Greater than
- < Less than
- #= Not equal

Examples using relational operators are:

```
$IF   A.V>3       Process succeeding block if value
                  component of variable A is >3.
$IF   B.L#=A.L    Process succeeding block if length
                  component of variable B is not equal
                  to length component of variable A.
```

The macro assembler also allows the use of **Boolean operators**, which perform the desired operation and return either *true* or *false*. The Boolean operators are:

- & AND
- ++ OR
- – – NOT

An example using the Boolean operators is:

```
$IF - -((A.V>3) & (B.L#=A.L))
```

Macro symbol components can be concatenated with literal strings, model statement characters, and other macro variables. Concatenation is indicated by writing character strings side by side with string mode references.

8.3 Variables

Macro definitions can include **variables** which are represented in the same manner as symbols in the assembler symbol table (AST). Macro variables can have a maximum length of two characters. Examples of valid variables are:

- VA
- P4
- SC
- F2
- A

Note:

Macro variables are strictly local, available only to the macro which defines them. Symbols in the assembler symbol table can only be accessed through symbol components.

Macro variables can be defined in two ways:

1) As parameters defined by the $MACRO statement, and
2) In $ASG statements (see the $ASG verb).

The macro translator maintains a macro symbol table (MST) similar to the AST. Each MST entry contains the variable/parameter and its string, value, length, and attribute components. The macro expander module places parameters in the MST when macro calls are processed and places variables in the MST when it processes $ASG statements.

8.3.1 Parameters

Parameters are variables that are declared in the $MACRO definition statement. The parameter declaration sequence corresponds to the sequence of the operands in the macro call statement. During macro expansion, the parameters receive the values of the macro call operands. Examples of $MACRO statements with parameters are:

```
LABEL     $MACRO  A,B3
NAME      $MACRO  O,RC,AM
```

8.3.2 Macro Variable Components

There are four types of variable/parameter components:

1) The **string component** of an MST entry contains a character string assigned to the macro variable/parameter by the macro expander.
2) The **value component** of an MST entry contains:
   a) The *binary equivalent* of the string component, if the string component is an *integer.*

b) The *value of the symbol*, if the string component is a *symbol* in the AST.

c) The *length of the list*, if the parameter is an *operand list*.

3) The **length component** contains the number of characters in the string component.

4) The **attribute component** of the MST is a bit vector. The bits correspond to the attributes of the variable or parameter.

The following statement defines a macro with parameters X and NUM:

```
ADDK    $MACRO  X,NUM
```

The following statement calls the ADDK macro:

```
ADDK    VAR1,3
```

The MST now contains entries for parameters X and NUM and their associated components:

Parameter X:

String Component	Is the character string VAR1.
Attribute Component	Indicates that the parameter is supplied in a macro call (keyword $PCALL).
Length Component	Is 4.
Parameter NUM:	
String Component	Is the character 3.
Value Component	Is 3 also, expressed as a 16-bit binary number.
Length Component	Is 1.
Attribute Component	Indicates that the parameter is supplied in the macro call (keyword $PCALL).

Each component of a macro variable can be accessed individually in either **binary** or **string mode**:

- In *binary mode*, the referenced macro variable component is treated as a signed 16-bit integer. Binary mode is accessed by writing the variable name and component. A reference to the string component of a macro variable in binary mode is the 16-bit integer value of the ASCII representation of the first two characters of the string. For example, the binary mode value of the string component of X, in the preceding example, is >5641, which is the ASCII representation for VA.

- *String mode* access of macro variable components is signified by enclosing the variable in a pair of colon characters (:). For example,

```
:X:
```

Note:

Colons are always used in pairs to enclose a variable name. If a variable component qualifier is used, the pair of colons enclose the entire qualified name.

8.3.3 Variable Qualifiers

Table 8–1 lists the names used to indicate variable/parameter components. The variable name is followed by a period (.) and the single letter qualifier.

Table 8–1. Variable Qualifiers

Qualifier	Meaning
S	The string component of the variable
A	The attribute component of the variable
V	The value component of the variable
L	The length component of the variable

The following examples show qualified variables for the macro call:

```
ADDK      VAR1,3
```

which was defined by the following statement:

```
ADDK      $MACRO  X,NUM
```

X.S Is the string component (binary mode) of variable VAR1. X.S equals the binary equivalent for VA, or >5641. If string mode is indicated, as in :X.S:, the string component is the character string VAR1.

X.A Is the attribute component of variable VAR1. This component is accessed by using logical operators and keywords as described in Table 8–2, Table 8–3, and Table 8–4.

X.V Is the value component of variable VAR1.

X.L Is the length component of variable VAR1; in this case, it is equal to the character string 4.

Unqualified variables (except those in $ASG statements) refer to the variable's string component. These two strings are equivalent:

:CT.S: WAY Variable CT qualified; string component = WAY.
:CT: WAY Variable CT unqualified; string component = WAY.

Note:

Binary references to macro variables in model statements **must** be qualified.

8.3.4 Symbol Components

Entries in the assembler symbol table have symbol components. To access symbol components in a macro, the symbol must be assigned to the string component of a macro variable by an $ASG statement. The additional qualifiers shown in Table 8–2 are used with macro variables to access the AST symbol's components.

Table 8–2. Variable Qualifiers for Symbol Components

Qualifier	Meaning
SS	String component of a symbol that is the string component of a variable.
SV	Value component of a symbol that is the string component of a variable.
SA	Attribute component of a symbol that is the string component of a variable.
SL	Length component of a symbol that is the string component of a variable.

The following examples show qualified variables that specify symbol components of variable string components. Assume that the following statement appears in the source program:

```
MASK    EQU    >FF
```

This statement appears in a macro definition:

```
$ASG   V1.S TO MASK
```

V1.SS Is the string component of the symbol MASK. This is null unless a macro instruction has caused a string to be associated with it by using a $ASG statement.

V1.SV Is the value component of the symbol MASK (>FF). In the string mode, :V1.SV: equals the character string 255.

V1.SA Is the attribute component of the symbol MASK. This component may be accessed by using logical operators and keywords.

V1.SL Is the length component of the symbol MASK. If a string has been assigned to MASK, then V1.SL is the length of that string.

Concatenation is especially useful when a previously defined string is augmented with additional characters. Assume that CT.S represents the string ONE.

`:CT.S:' WAY'` produces the string `'ONE WAY'`

If CT.S represented the character string TWO, the result of the concatenation in the example would be TWO WAY. Strings and qualified variables can be concatenated as required. Components of variables that are represented by a binary value (for example, CT.V and CT.L) are converted to their ASCII decimal equivalent before concatenation. For example:

`:CT.S' WAY ':CT.L:` expands into `ONE WAY 3:`

since the length component of the variable CT is three.

8.4 Keywords

Keywords identify assembler symbol and macro parameter attribute components. Each keyword represents a bit position in a word that contains all of the symbol or parameter attribute components. Keywords can be used with logical operators and attribute components to test or set a specific attribute of a symbol or parameter. The following paragraphs describe how keywords are used with symbols and parameters.

8.4.1 Symbol Attribute Component Keywords

Table 8–3 lists keywords that are used with a logical operator and the symbol attribute component (.SA) to test or set the corresponding attribute component in the AST.

Table 8–3. Symbol Attribute Keywords

Keyword	Meaning
$REL	Symbol is relocatable
$REF	Symbol is an operand of an REF directive
$DEF	Symbol is an operand of a DEF directive
$STR	Symbol has been assigned a component string
$MAC	Symbol is defined as a macro name
$UNDF	Symbol is not defined

Note: Using these attributes in conditional assembly (with the $IF verb) may lead to pass conflict errors if the symbol is not defined before the macro is called.

Assume that the next statement is an assembler program source statement and the second statement appears in a macro definition:

```
MASK        EQU     >FF

            $ASG    V1.S TO MASK
```

The next line ANDs symbol MASK's attribute component with a flag corresponding to the keyword $STR.

```
V1.SA&$STR
```

This expression is *true* when MASK's contents are not null; otherwise, the expression is *false*.

The next example shows ORs symbol MASK's attribute component with the flag corresponding to the keyword $REL.

```
V1.SA++$REL
```

8.4.2 Parameter Attribute Keywords

Table 8–4 lists keywords that are used with a logical operator and the macro symbol attribute component to test or set the corresponding attribute in the

MST attribute component. Use these attribute keywords to test or set attribute components of all variables in the MST.

Table 8–4. Parameter Attribute Keywords

Keyword	Meaning
$PCALL	Parameter appears as a macro-instruction operand
$POPL	Parameter is an operand list; the value component contains the number of operands in the list
$PSYM	Parameter is a symbolic memory address †

† A symbolic memory address is recognized when the variable is preceded by an @ character.

The following expressions use parameter attribute component keywords:

P6.A&$PCALL AND variable P6's attribute component with the flag corresponding to keyword $PCALL. The expression is *true* when variable P6 is a parameter supplied in a macro call, otherwise the expression is *false*.

RA.A++$PSYM OR variable RA's attribute component with the flag corresponding to keyword $PSYM.

8.5 Assigning Values to Parameters

Macro definitions expand macro calls (statements that have the macro name as an opcode).

Macro definition syntax is:

```
<macro name> $MACRO [<parm>][,<parm>] [<comment>]
```

Macro call syntax is:

```
<macro name> [<operand/list>],[<operand/list>] [<comment>]
```

When a macro call is processed, the macro expander associates the first parameter in the $MACRO statement with the first operand or operand list in the macro call, the second parameter with the second operand or operand list, and so on.

Each operand may be any assembler expression or address type, or a quote-enclosed character string. An operand list is a group of operands enclosed in parentheses and separated by commas (when two or more operands are in list). An operand list is processed as a set, after the outer parentheses are removed, during macro expansion. Operands (or operand lists) may be nested in parentheses in the macro call for use within macro definitions.

The following $MACRO statement defines two parameters.

```
ONE         $MACRO    P1,P2
```

The corresponding macro call

```
ONE         PAR1,PAR2
```

associates PAR1 with P1 and PAR2 with P2. However, a call such as:

```
ONE         PAR1,(PAR21,PAR22)
```

associates PAR1 with P1 and the list PAR21,PAR22 with P2.

Now :P2: or :P2.S: can be used as a pair of operands in a model statement.

The $PCALL attribute is set for each parameter that receives a value. When the $MACRO statement defines more parameters than the number of operands in the macro call, the $PCALL attribute is not set for the excess parameters. The $PCALL attribute is also not set if an operand is "null"; i.e., the call line has two commas adjacent or an operand list of zero operands. Expansion of the macro can be controlled by the number of operands by using the $PCALL attribute and $IF statement. For example, the following macro definition and macro call

```
AMAC        $MACRO  P1,P2,P3
AMAC        AB1,AB2
```

sets $PCALL for parameters P1 and P2 but not for P3. Similarly,

```
AMAC    XY,,XY3
```

sets $PCALL for P1 and P3 but not for P2.

When the macro instruction has more operands than the number of parameters in the $MACRO statement, the excess operands are combined with the operand or operand list corresponding to the last parameter to form an operand list (or a longer operand list). In the macro statements below, the operands of the two macro calls would be assigned to the parameters in the same ways:

```
(1)
ONE     EQU     9
TWO     EQU     43
THREE   EQU     86
FIX     $MACRO  P1,P2              Define Macro FIX
          .
          .
          .
        FIX     ONE,TWO,THREE     Call Macro FIX
        FIX     ONE,(TWO,THREE)   Call Macro FIX
(2)
A       EQU     7
B       EQU     15
C       DATA    17
D       DATA    63
E       EQU     95
F       EQU     47
G       EQU     58
H       EQU     101
I       EQU     119
PARM    $MACRO  P1,P2,P3,P4,P5,P6,P7,P8,P9
          .
          .
          .
        PARM    @A,,B,(),C,(D),E,(G,(H,I))
```

Parameter assignments:

P1.S = A
P1.A = $PCALL
P1.L = 1
P1.V = 7

P2.S = (no string)
P2.A = (all false)
P2.L = 0
P2.V = 0

P3.S = B
P3.A = $PCALL
P3.L = 1
P3.V = 15

P4.S = (no string)
P4.A = $POPL
P4.L = 0
P4.V = 0

P5.S = C
P5.A = $PCALL
P5.L = 1
P5.V = 17

P6.S = D
P6.A = $PCALL,$POPL
P6.L = 1
P6.V = 1

P7.S = E
P7.A = $PCALL
P7.L = 1
P7.V = 95

P8.S = G,(H,I)
P8.A = $PCALL,$POPL
P8.L = 7
P8.V = 2

P9.S = (no string)
P9.A = 0 (all false)
P9.L = 0
P9.V = 0

8.6 Verbs

The macro language supports seven verbs that are used in macro language statements. Table 8–5 lists the seven verbs. Any statement in a macro definition that does not contain a macro language verb in the operation field is processed as a model statement.

Table 8–5. Macro Language Verb Summary

Verb	Description
$ASG	Assigns values to variable components
$ELSE	Begins an alternate block in a conditional process
$END	Marks the end of a macro definition
$ENDIF	Terminates conditional processing
$IF	Provides conditional processing
$MACRO	Marks beginning of macro definition
$VAR	Declares variables for macro definitions

Syntax

$ASG <expression/string> TO <var> [<comment>]

Description

The $ASG statement assigns values to variable components. Variables that are not parameters do not have values for any components until values are assigned using $ASG statements. Variable components with previously assigned values may be assigned new values with $ASG statements.

The expression operand may be any expression valid to the assembler and may contain binary mode variable references and the keywords in Table 8–3. and Table 8–4.

Note:

The binary mode value of a string component or symbol string component used in an expression is the binary value of the first two characters of the string. Thus, if GP.S has the string LAST, the value used for GP.S is an expression in the <string> hexadecimal number >4C41 which is the ASCII representation for LA.

A string may be one or more characters enclosed in single quotes, or the concatenation of such a literal string with the string mode value of a qualified variable. The <var> may be either an unqualified variable or a qualified variable.

When the operands are both unqualified variables, all components are transferred to target variables. When the destination variable is qualified, only the specified component receives the corresponding component of the expression or string. An exception to this is when a string is assigned to the string component of a variable or symbol, the length component of that variable or symbol is set to the number of characters in the assigned string. If the attribute component of the destination variable is to be changed, only those attributes which can be tested using keywords are changed. Other attributes maintained by the macro assembler may or may not be changed as appropriate.

Note:

A qualified variable that specifies the length component is illegal as a destination in a $ASG statement and will **not** set the length component.

Examples

Assume that variables P3, V3,and CT were previously declared as parameters ($MACRO statement) or variables ($VAR statement).

```
*    Assign all the components of variable P3 to
*    variable V3.
   $ASG   P3 TO V3
*
*    Concatenate string 'ES' to the string com-
*    ponent of variable P3, and set the string
*    component to the result. Also, add 2 to
*    the value of the new length component.
   $ASG  :P3.S:'ES' TO P3.S
*
*    Set the flag in the attribute component
*    of variable CT to indicate the symbolic
*    address attribute.
   $ASG A++PSYM TO CT.A
```

The $ASG statement may be used to modify symbol components as shown in the following examples. Assume that P3.V = 6 and P3.S = SUB.

```
*    Assign 'TEN' as the string component of
*    variable G. When 'TEN' is a symbol in the
*    AST, this statement allows the use of in-
*    direct component qualifiers to modify the
*    components of symbol TEN.
   $ASG 'TEN' TO G.S
*
*    Set the value component of the symbol in
*    the string component of variable G to the
*    value component of variable P3. In this
*    case, the value component of TEN is set to 6.
   $ASG P3.V TO G.SV
*
*    Concatenate string 'A', the string compo-
*    nent of variable P3, and string 'S' and
*    place the result in the indirect string
*    component of the same symbol. Thus, the
*    string component of TEN is ASUBS and the
*    length component is 5.
   $ASG 'A':P3.S:'S' TO G.SS
```

Note:

Keywords in an $ASG statement **must** be used with a Boolean operator and an attribute component of a variable in the source field. The attribute component must come first.

Syntax

$ELSE [<comment>]

Description

The $ELSE statement begins an alternate block to be processed if the preceding $IF expression was false.

Syntax $END [<macro name>][<comment>]

Description The $END statement ends a macro definition. When executed, the $END statement terminates the processing of the macro definition. The <macro name> parameter is optional.

Example `$END FIX` Terminates the definition of macro FIX.

Syntax

$ENDIF [<comment>]

Description

The $ENDIF statement terminates the conditional processing initiated by an $IF statement in a macro definition.

Syntax

$IF <expression> [<comment>]

Description

The $IF statement provides conditional processing in a macro definition.

An $IF statement is followed by a block of macro language statements terminated by an $ELSE statement or an $ENDIF statement. When the $ELSE statement is used, it is followed by another block of macro language statements terminated by an $ENDIF statement. When the expression in the $IF statement has a nonzero value (or evaluated as *true*), the block of statements following the $IF statement is processed. When the expression in the $IF statement has a zero value (or evaluated as *false*), the block of statements following the $IF statement is skipped. When the $ELSE statement is used and the expression in the $IF statement has a nonzero value, the block of statements following the $ELSE statement and terminated by the $ENDIF statement is skipped. Thus, the condition of the $IF statement may determine whether or not a block of statements is processed, or which of two blocks of statements is processed. A block may consist of zero or more statements. The <expression> may be any expression as defined for the $ASG statement and may include qualified variables and keywords. The expression defines the condition for the $IF statement.

Note:

The $IF expression is always evaluated in binary mode. Specifically, the relational operations (<,>,=,#=) operate only on the binary mode values of macro variables. Boolean operators may be nested. In addition, $IF blocks may be nested, at most, 44 levels deep.

Example

These examples show conditional processing in macro definitions:

```
 .
 .
$IF     KY.SV       Process the statements of BLOCK
 .                   A when the indirect value com-
 .                   ponent of the variable KY con-
 .                   tains a non-zero value.
 .      BLOCK A      Process the statements of BLOCK
 .                   B when the component contains
$ELSE                zero after processing either
 .                   block of statements. Continue
 .      BLOCK B      processing the statement fol-
 .                   lowing the $ENDIF statement.
$ENDIF
 .
```

```
  .
$IF --(T.A&$PCALL)
  .                    Process the statements of BLOCK
  .                    A when the attribute component
  .                    of parameter T indicates that
  .    BLOCK A         parameter T was not supplied in
  .                    the macro instruction. If para-
  .                    meter T was supplied, do not
  .                    process the statements of BLOCK
$ENDIF                 A. Continue processing at the
  .                    statement following the $ENDIF
  .                    statements in either case.
  .
$IF     T.L=5          Process the statements of BLOCK
  .                    A when the length component of
  .                    variable T is equal to 5, do not
  .                    process the statements of BLOCK
  .    BLOCK A         A. Continue processing at the
$ENDIF                 statement following the $ENDIF.
```

Syntax

<macro name> $MACRO [<parm>][,<parm>] [<comment>]

Description

The $MACRO verb begins a macro definition. It must be the first statement in the definition. $MACRO assigns a name to the macro and declares the macro parameters.

The **macro name** contains one to six alphanumeric characters; the first must be a letter. Each **<parm>** is a parameter for the definition as described in subsection 8.3.1. The operand field may contain as many parameters as the size of the field allows and must contain all parameters used in the macro definition. The **comment field** can only be used if there are parameters.

The macro definition is used to expand macro calls (statements that have the macro name as an opcode). The macro name specifies the macro definition to be used. When a macro call is processed, the macro expander associates the first parameter in the $MACRO statement with the first operand or operand list in the macro call, the second parameter with the second operand or operand list, and so on.

Example

```
ONE     $MACRO  P1,P2
```

specifies two parameters. A call such as

```
ONE     PAR1,PAR2
```

associates PAR1 with P1 and PAR2 with P2.

Note:

A macro definition supercedes previous macro definitions and opcodes with the same name. Symbolic operands which appear in a macro call are treated as symbolic operands in opcodes; if they are not defined with the program in which they appear, they will be listed as undefined symbols.

Syntax

$VAR <var>[,<var>] [<comment>]

Description

The $VAR statement declares the variables for a macro definition. $VAR is required only if the macro definition contains one or more variables that are not parameters. More than one $VAR statement may be included; each $VAR statement may declare more than one variable. Each <var> in the operand is a variable as previously described (see Section 8.3).

The $VAR statement does not assign values to any components of the variables. $VAR statements may appear anywhere in the macro definition to which they apply, provided each variable is declared before the first statement that uses the variable. Placing $VAR statements immediately following the $MACRO statement is recommended.

Example

```
$VAR  A,CT,V3  Three variables for a macro
```

This example declares variables A, CT, and V3; A, CT, and V3 must not have been declared as parameters.

8.7 Model Statements

Most macro definitions contain **model statements**. A model statement is, or produces, an assembly language statement. Model statements are composed of the usual assembly language statement elements and can include qualified variable components (string mode only). The source statement produced must be a legal assembly language statement.

The following examples show model statements:

```
MOV         %6,R12
```

This model statement is itself an assembly language source statement that contains a machine instruction.

```
:P7.S:      MPY    :P2.S:,R8 :V4.S:
```

This model statement begins with the string component of variable P7. Three blanks, MPY, and three more blanks are concatenated to the string. The string component of variable P2 is concatenated to the result, to which R8 and three blanks are concatenated. A final concatenation places the string component of variable V4 in the model statement. This produces an assembly language instruction in which the label, comment and part of the operand fields are supplied as string components.

```
:MS.S:
```

This model statement is the string component of variable MS. Preceding statements in the macro definition must place a valid assembly language source statement in the string component to prevent assembly errors.

Note:

Conditional assembly directives may not appear as operations in a model statement. Comments supplied in model statements may not contain periods (.) since the macro assembler scans comments in the same way as model statements and improper use of punctuation may cause syntax errors.

8.8 Macro Examples

Macros may simply substitute a machine instruction for a macro instruction, or they may include conditional processing, access the assembler symbol table, and employ recursion. Several examples of macro definitions are described in the following paragraphs.

8.8.1 Macro ID

Example macro ID is a macro with a default value. The macro supplies two DATA directives to the source program. It consists of nine macro language statements, four of which are model statements.

```
ID      $MACRO WS,PC          Defines ID with parameters WS and PC
        DATA   :WS.S:         Model statement - places a DATA directive
                              with the string of the first parameter as
                              the operand in the source program.
        $IF    PC.A&$PCALL    Tests for presence of parameter PC
        DATA   :PC.S:,15      Model statement - places a DATA directive in
                              the source program.  The first operand is the
                              string of the second parameter, and the second
                              operand is 15. This statement is processed if
                              the second parameter is present.
        $ELSE                 Start of alternate portion of definition.
        DATA   START,15       Model statement - places a DATA directive in the
                              source program.  The first operand is label
                              START, and the second operand is 15.
                              This statement is processed if the second
                              parameter is omitted.
START   EQU    $              Model statement - places a label START in the
                              source program.  This statement is processed
                              if the second parameter is omitted.
        $ENDIF                End of conditional processing.
        $END                  End of macro.
```

The macro call syntax is:

```
[<LABEL>] ID <address>[,<address>] [<comment>]
```

The addresses may be expressions or symbols.

A sample ID call would be:

```
ID      WORK1,BEGIN
```

This would be replaced with the following source code:

```
DATA    WORK1
DATA    BEGIN,15
```

If only one operand is supplied, the macro instruction could be coded as follows:

```
ID      WORK2
```

This would produce the following source code:

```
        DATA   WORK2
        DATA   START,15
START   EQU    $
```

This form of the macro instruction imposes two restrictions on the source program:

1) The source program may not use the label START and

2) May not call macro ID more than once.

Problems with labels supplied in macros may be prevented by reserving certain characters for use in macro-generated labels. A macro definition may maintain a count of the number of times it is called and use this count in each label generated by the macro.

8.8.2 Macro GENCMT

This example shows how to implement both those comments which appear in the macro definition only and those which appear in the macro expansion. When this macro is called, the statement in line six generates a comment.

```
0001                        IDT    'GENCMT'
0002              GENCMT    $MACRO
0003                        $VAR V
0004              * This is a macro definition comment     *
0005                        $ASG '*' TO V.S
0006              :V.S: This is a macro expansion comment *
0007                        $END
0008                        GENCMT
0001              * This is a macro expansion comment      *
0009 0000 0000              DATA 0,1
     0002 0001
0010                        GENCMT
0001              * This is a macro expansion comment      *
0011                        GENCMT
0001              * This is a macro expansion comment      *
0012 0004 0004              DATA 4
0013                        END
NO ERRORS, NO WARNINGS
```

8.8.3 Macro FACT

This example shows the recursive use of macros. FACT produces the assembly code necessary to calculate the factorial of N, and store that value at data memory address LOC. Macro FACT accomplishes this by calling FACT1, which calls itself recursively.

```
FACT   $MACRO  N,LOC
       $IF     N.V<2
       MOV     %1,A              * 1% = 0% =1
       STA     @:LOC:
       $ELSE
       MOV     %:N.V:,A          * N greater than/equal 2,
       STA     @:LOC:            * so store N at LOC
       $ASG    N.V-1 TO N.V      * Decrement N
       FACT1   :N.V:,:LOC:       * Do Factorial of N-1
       $ENDIF
       $END
*
FACT1  $MACRO  M,AREA
       $IF     M.V>1
       LDA     @:AREA:           * Multiply factorial so far
       MPY     %:M.V:,A          * by current position
       MOV     B,A
       STA     @:AREA:           * Save result
       $ASG    M.V-1 TO M.V      * Decrement position
       FACT1   :M.V:,:AREA:      * Recursively calls itself
       $ENDIF
       $END
```

8.8.4 Macro PULSE

This is a set of macros in which the name describes an addressing mode expected by the macro. The example assigns register A to a port, register B to a port, and an immediate value to a port. These macros can be useful in programming I/O routines.

```
PULSEA $MACRO PX
       ORP    A,:PX.S:
       $END
*
PULSEB $MACRO PX
       ORP    B,:PX.S:
       $END
*
PULSEI $MACRO I,PX
       ORP    %:I.S:,:PX.S:
       $END
```

8.9 Macro Error Messages

Table 8–6 lists and defines the macro error messages which may be generated.

Table 8–6. Macro Error Messages

Macro Error Message	Description
MACRO LINE TOO LONG	In a macro definition, macro directive lines may only be 58 characters long, and model statements, when fully expanded, may only be 60 characters long.
LONG MACRO VARIABLE QUALIFIER	Macro variable qualifiers may only be one or two characters in length.
TOO MANY MANY VARIABLES	The total number of macro parameters, variables and labels in one macro definition may not exceed 128.
INVALID MACRO QUALIFIER	The only valid macro qualifiers are: S,V, L, A, SS, SV, SL and SA.
VARIABLE ALREADY DEFINED	A macro variable cannot be redefined within a macro.
IF LEVEL EXCEEDED	The maximum nesting level of $IF directives is 44.
MACRO ASSEMBLER	The macro assembler has detected an internal PROGRAM ERROR error. These can be caused by incorrect syntax.

Chapter 9

Design Aids

This chapter contains sample TMS7000 applications to aid you in system development.

Section		Page
9.1	Microprocessor Interface Example	9-2
9.2	Programming the TMS77C82	9-6
9.3	Serial Communication with the TMS7000 Family	9-12
9.4	The Status Register	9-24
9.5	Stack Operations	9-27
9.6	Subroutine Instructions	9-28
9.7	Multiplication and Shifting	9-30
9.8	The Branch Instruction	9-31
9.9	Interrupts	9-32
9.10	Write-Only Registers	9-34
9.11	Sample Routines	9-35

9.1 Microprocessor Interface Example

Figure 9–1 illustrates a method for interfacing a TMS7xCx2 microcomputer to external memory devices such as EPROM and RAM. This interface is designed to operate at the TMS7xCx2's maximum operating frequency (8 MHz). Any combination of ROM, RAM or other peripheral devices could be added into the circuit and enabled by the other $\overline{SEL}$ pins, provided that their timing requirements allow them to be interfaced to the TMS7xCx2.

In this circuit, the mode control pin (MC) is tied to V_{CC}, placing the TMS7xCx2 in microprocessor mode. All 16 addressing bits on ports C and D are available in microprocessor mode. The on-chip ROM is disabled in this mode, and its address space is available externally. For more information on port and mode operation see Chapter 3.

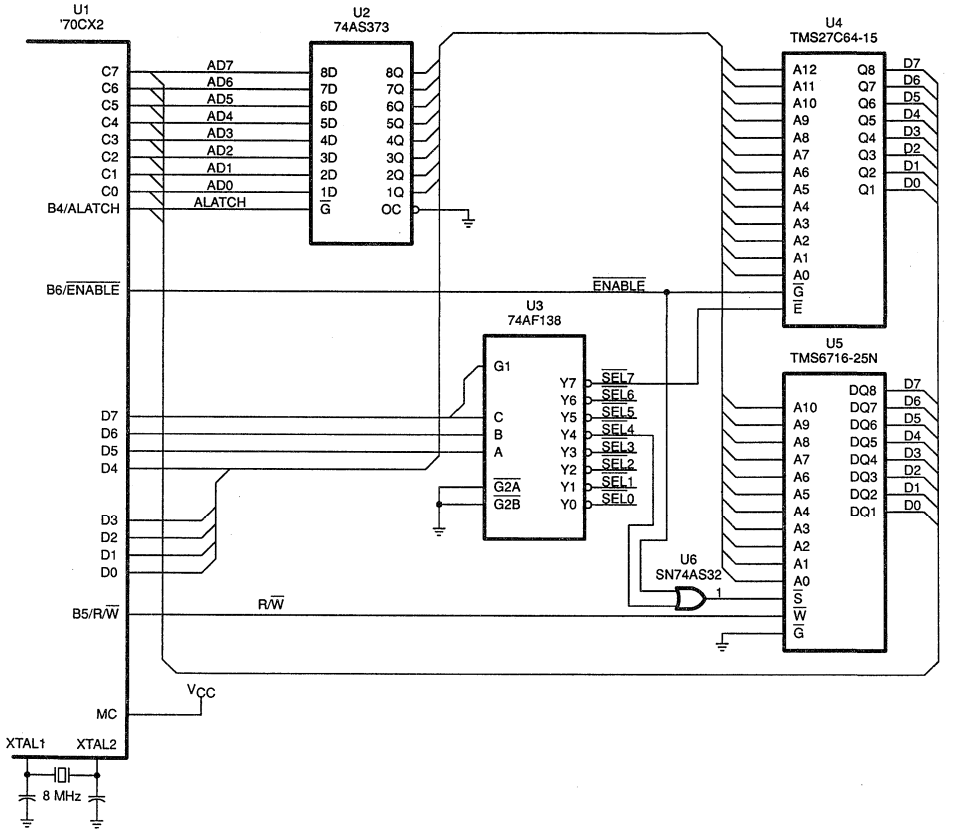
Note the following features in this sample circuit:

- Port A and the lower nibble of port B operate the same as in the single-chip mode.
- The memory control signals are brought out on the upper nibble of port B.
- Port C becomes the multiplexed least significant 8-bit address bus (A7–A0) and full 8-bit data bus.
- Port D becomes the most significant 8-bit address bus (A15–A8).
- The least significant 8 bits of the 16-bit address bus (A7–A0) are latched into the SN74AS373 (U2) by the ALATCH signal during read/write memory cycles.
- A full address decode is accomplished with the SN74AS138 (U3). Eight memory select lines ($\overline{SEL7}$–$\overline{SEL0}$) are generated by U3 and are each individually activated on an 8K-byte address block. Table 9–1 lists the address range decoded by each select pin.

Table 9–1. Memory Address Decode

Pin	Address Range
$\overline{SEL7}$	>E000 to >FFFF
$\overline{SEL6}$	>C000 to >DFFF
$\overline{SEL5}$	>A000 to >BFFF
$\overline{SEL4}$	>8000 to >9FFF
$\overline{SEL3}$	>6000 to >7FFF
$\overline{SEL2}$	>4000 to >5FFF
$\overline{SEL1}$	>2000 to >3FFF
$\overline{SEL0}$	>0000 to >1FFF

Figure 9–1. TMS7xCx2 Microprocessor Interface Sample Circuit



The devices used in this circuit are:

U1: **TMS70Cx2** – 8-bit microcomputer with UART.

U3: **SN74AF138** – Like U2, the AF version of the 138 allows use of less expensive EPROMs.

U4: **TMS27C64-15** – This EPROM chip is the slowest device that can be used in this circuit because the timing requirement [$T_{a(A–D)}$] for the TMS7xCx2 is 175 ns at 8 MHz. The propagation delay through U2 is 6 ns, so only 169 ns remain for the EPROM chip to use. Therefore the TMS27C64-15 with its 150 ns access time [$T_{a(A)}$] was selected.

U5: **TMS6716-25N** – This RAM chip is the slowest device that can be used in this circuit because the timing requirement [$T_{a(EL–D)}$] for the

TMS70Cx2 is 30 ns at 8 MHz. The propagation delay through U6 is 5.8 ns, so only 24.2 ns remain for the RAM chip to use. Therefore the TMS6716-25N with its 25 ns delay time was selected.

9.1.1 Read Cycle Timing

The TMS70Cx2 requires a minimum **address-to-data access time** [$T_{a(A-D)}$] of 175 ns at 8 MHz. $T_{a(A-D)}$ for the TMS27C64-15 in this circuit is:

$$\begin{aligned} \text{Access time (175 ns)} &\geq t_{ph[U2]} + t_{a(A)[U4]} \\ &\geq 6 + 150 \\ 175 \text{ ns} &\geq 156 \text{ ns} \end{aligned}$$

$T_{a(A-D)}$ for the TMS6716-25N in this circuit is:

$$\begin{aligned} \text{Access time (175 ns)} &\geq t_{ph[U2]} + t_{a(A)[U5]} \\ &\geq 6 + 25 \\ 175 \text{ ns} &\geq 31 \text{ ns} \end{aligned}$$

The TMS70Cx2 parameter used to calculate $T_{a(A-D)}$ will also be used to calculate **chip-select-to-data access time**. $T_{a(E)}$ for the TMS27C64-15 in this circuit is:

$$\begin{aligned} \text{Access time (175 ns)} &\geq t_{ph[U3]} + t_{a(E)[U4]} \\ &\geq 6 + 150 \\ 175 \text{ ns} &\geq 156 \text{ ns} \end{aligned}$$

Since the chip-select to the TMS6716-25N is gated with the ENABLE signal, use the access time $T_{a(EL-D)}$ to calculate the **chip-select-to-data time**. $T_{a(E)}$ for the TMS6716-25N in this circuit is:

$$\begin{aligned} \text{Access time (31 ns)} &\geq t_{plh[U6]} + t_{a(E)[U5]} \\ &\geq 5.8 + 25 \\ 31 \text{ ns} &\geq 30.8 \text{ ns} \end{aligned}$$

The TMS70Cx2 requires a minimum $\overline{\text{ENABLE}}$/**rise-to-data-disable time** of 75 ns at 8 MHz. The minimum requirement for the TMS27C64-15 in this circuit is:

$$\begin{aligned} \text{Disable time (75 ns)} &\geq t_{dis(G)[U4]} \\ &\geq 60 \\ 75 \text{ ns} &\geq 60 \text{ ns} \end{aligned}$$

The requirement for the TMS6716-25N in this circuit is:

$$\begin{aligned} \text{Disable time (75 ns)} &\geq t_{dis(E)[U5]} + t_{phl[U6]} \\ &\geq 10 + 5.8 \\ 75 \text{ ns} &\geq 15.8 \text{ ns} \end{aligned}$$

9.1.2 Write Cycle Timing for Microprocessor Mode

The TMS70Cx2 requires a minimum data-output-valid time ($T_{d(EH-AL)}$ of 25 ns at 8 MHz. Since $\overline{E}$ is gated to the $\overline{ENABLE}$ line, the $\overline{ENABLE}$ signal can be used to calculate the data-output requirement for the TMS6716-25N.

$$\begin{aligned} \text{Output valid (25 ns)} &\geq t_{phl[U6]} + t_{h(D)[U5]} \\ &\geq 5.8 + 5 \\ 25 \text{ ns} &\geq 10.8 \text{ ns} \end{aligned}$$

Table 9–2. TMS6716-25N Timing Characteristics

	Parameter	Min	Max	Unit
$T_{a(A)}$	Access time from address		25	ns
$T_{a(E)}$	Access time from chip enable		25	ns
$T_{dis(\overline{E})}$	Output disable time from $\overline{E}$	0	10	ns
$T_{su(A)}$	Address set up time	0		ns
$T_{su(D)}$	Data set up time before write high	15		ns
$T_{h(D)}$	Data hold time after write high	5		ns

Table 9–3. TMS27C64-15 Timing Characteristics

	Parameter	Min	Max	Unit
$T_{a(A)}$	Access time from address		150	ns
$T_{a(E)}$	Access time from chip enable		150	ns
$T_{en(G)}$	Output enable time from $\overline{G}$		75	ns
$T_{dis(G)}$	Output disable from $\overline{G}$	0	60	ns

Table 9–4. SN74AS363, SN74AF138, and SN74AS32 Propagation Delay Times

	Parameter	Min	Max	Unit
T_{pd}	Propagation delay, SN74AS373		6	ns
T_{pd}	Propagation delay, SN74AF138		6	ns
T_{pd}	Propagation delay, SN74AS32		5.8	ns

9.2 Programming the TMS77C82

The TMS77C82 is an EPROM version of the TMS70C82. It has 8K bytes of EPROM in place of the ROM. It has the same instruction set as all TMS7000 devices, and has the same peripheral map as all TMS7xCx2 devices.

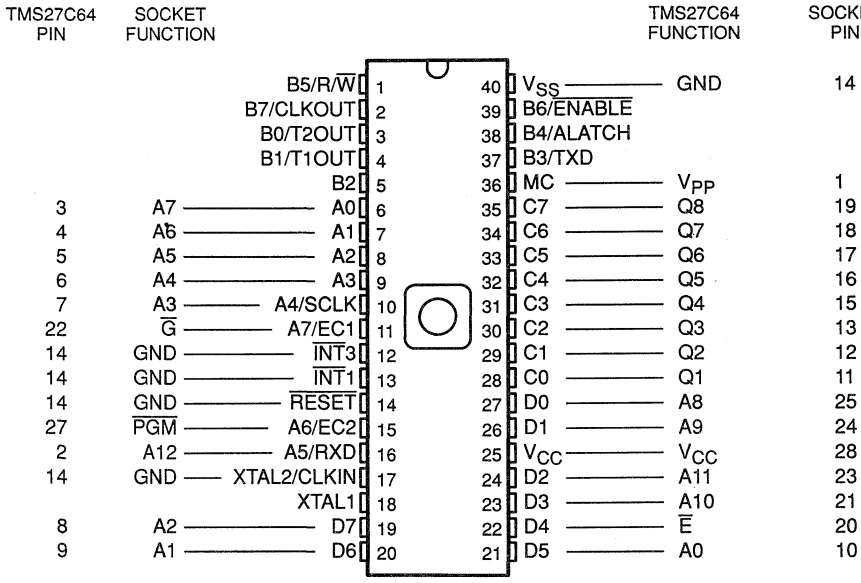
The TMS77C82 can be programmed with the following devices:

- Standard EPROM programmer
- TMS7000 Evaluation Module

9.2.1 Programming the TMS77C82 Using an EPROM Programmer

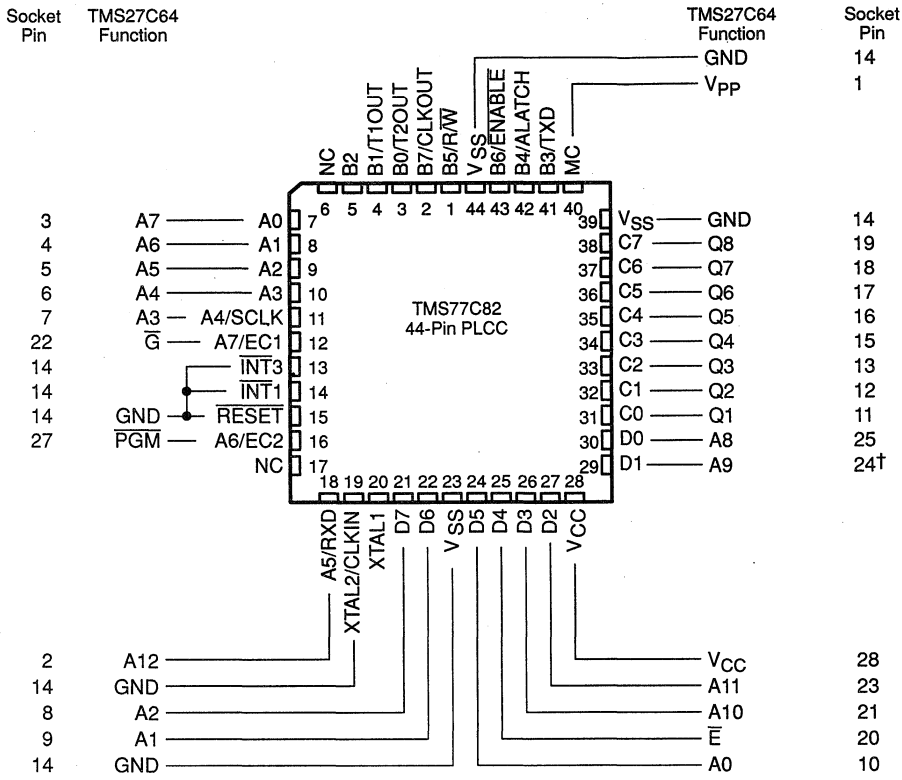
The TMS77C82 can be programmed like any Texas Instruments TMS27C64 on a wide variety of PROM programmers. Programming the TMS77C82 requires a 40-to-28 pin adapter socket with the $\overline{\text{RESET}}$, XTAL2, and $\overline{\text{INT}}$1 pins grounded. Figure 9–2 shows the connections needed to be made for the 40-to-28 pin programming socket. This programming adapter is available from any Texas Instruments systems distributor (Part Number RTC/PGMC82A-06).

Figure 9–2. EPROM Programmer 40-to-28-Pin Conversion Socket



† For signature mode, insert a 3.9K ohms resistor between the pin #21 of the socket and the pin #29 of the TMS77C82 NL

Figure 9–3. 44-Pin PLCC to 44-Pin Socket



† For signature mode, insert a 3.9K ohms resistor between the pin #24 of the socket and the pin #29 of the TMS77C82 FNL (PLCC Package).

Use the following sample procedure to program the TMS77C82 on a EPROM programmer:

1) Insert the TMS77C82 into the conversion socket.

2) Place the conversion socket (with the TMS77C82) into the 28-pin socket on the PROM programmer.

3) Program and verify the contents of the TMS77C82 in the same manner as any standard TMS27C64 EPROM.

Some standard programmer units perform a signature check automatically which results in programming voltage being applied to the A9 address line of the device. To avoid damage to the device, install a 3.9K ohm resistor between pin 24 of the socket and pin 26 of the TMS77C82 (DIL package).

9.2.2 EPROM Integrity Protection Using the R Bit

Once the TMS77C82 has been programmed with the desired code, the contents of the EPROM may be protected with the use of the R bit integrity feature. The function of the R-bit is to disable all external accesses to the on-chip EPROM while in the EPROM mode, which will prevent a protected code from being modified of read externally. The only way to "unprotect" the TMS77C82 after the R-bit has been programmed is by erasing the EPROM, thereby destroying protected code.

The following truth table demonstrates how to program and verify the EPROM and the R-bit, and the state of the data bus in each configuration:

Table 9–5. Truth Table for EPROM and R Bit

E Pin22	G Pin11	PGM Pin15	INT3 Pin12	MC Pin36	State of Data Bus	
0 V	0V	+5	+5/0V	+12.5	Fast EPROM program	EPROM Program Mode
0 V	+5	+5	+5/0V	+12.5	Fast EPROM verify	
+5	+5	+5	+12.5	+12.5	R-bit program	
+5	0V	+5	+12.5	+12.5	Output = FF hex	R-Bit Program Mode
0 V	0V	+5	+12.5	+12.5	EPROM	
+5	+5	+5	+12.5	+5	EPROM	
+5	+5	0V	+12.5	+5	EPROM	
+5	0V	+5	+12.5	+5	EPROM	R-Bit Verify Mode
+5	0V	0V	+12.5	+5	High-Z	
0 V	+5	+5	+12.5	+5	Output = FF hex	
0 V	+5	0V	+12.5	+5	Output = FF hex	
0 V	0V	+5	+12.5	+5	EPROM	
0 V	0V	0V	+12.5	+5	R-bit on C7	<< True R-bit Status
0 V	0V	+5	+12.5	+5	R-bit on C7, if it has blown. Or EPROM if not	<< False R-bit Status

For all the above states, $\overline{\text{RESET}}$ and XTAL/CLKIN are kept at V_{SS}.

R-Bit Programming Procedure:

1) Configure all referenced pins for the R-Bit program mode.
2) Power up the device.
3) Toggle C7 from the logical high (1), to a logical low (0), and back to a logical high (1).
4) Power down the device.

R-Bit Verify Procedure:

1) Configure all referenced pins for the R-Bit verify mode, called True R-Bit status.
2) Power up the device.
3) Read C7. Zero (0) is programmed, one (1) is not programmed.
4) Power down the device.

9.2.3 Programming the TMS77C82 Using the TMS7000 Evaluation Module

The RTC/EVM7000C (TMS7000 CMOS Evaluation Module) can be used to program the TMS77C82. A 40-to-28 pin conversion socket is required and $\overline{\text{RESET}}$ and XTAL2 must be grounded. Figure 9–2 shows the required connections for the 40-to-28 pin socket, and the socket is also available through Texas Instruments (Part Number RTC/PGMC82A-06).

Use the following procedure to program the TMS77C82 on an RTC/EVM7000C:

1) Verify that the TMS77C82 is erased (all >FFs).

   a) Enter: `?VE 0 1FFF C <CR>`

Note:

If an error statement appears at this point concerning the C character, a software patch may be required in the Debug Monitor EPROM U43 to program 12.5 volt V_{PP} EPROMs. If so, the procedure in subsection 9.2.4 will enable the EVM to program 12.5 volt V_{PP} EPROMs.

2) Program the TMS77C82. Note that the program to be loaded into the TMS77C82 must reside in EVM memory beginning at address >F006 or above. (If the code is planned to be put into a ROM coded device, it is suggested to start the program at >F006 or above for a 4K device or >F806 or above for a 2K device. See Section 11.1.1.)

   a) Enter: `?PE 6 1FFF F006 C <CR>`

3) Compare the TMS77C82 EPROM to the EVM memory to verify that they are identical.

   a) Enter: `?CE 6 1FFF F006 C <CR>`

9.2.4 Modify the RTC/EVM7000C Debug Monitor to Enable 12.5 Volt V_{PP} Programming

This step is required only if the EVM in use contains software revision 1.4 and errors are occuring whenever you are trying to program an EPROM with the C (12.5 volt V_{PP} EPROM) identifier.

(Example: ?PE 6 1FFF F006 C)

The following steps will modify the code in the present U43 TMS2764 EPROM. The new code will then be programmed into a new TMS2764 EPROM which will be used to replace to original U43 EPROM.

1) Move the present contents of U43 into user RAM.

   a) ?$MV E000 FFFF 4000

2) Modify three bytes using the $MM command.

   a) ?$MM 5EED (Change contents from 40 to 80)
      40 80 <CR>

   b) ?$MM 5EF5 (Change contents from 0F to 0C)
      0F 0C <CR>

   c) ?$MM 5F04 (Change contents from FF to 3F)
      FF 3F <CR>

3) Program a blank TMS2764 to replace the present U43.

   a) ?$PE 0 1FFF 4000 <CR>

   b) Power-down the EVM and replace U43 with new EPROM.

9.2.5 TMS77C82JDL Erasure

The TMS77C82JDL can be erased by exposing the chip to shortwave ultraviolet light that has a wavelength of 253.7 nanometers (2537 angstroms). The recommended minimum exposure dose (UV intensity × exposure time) is 15 watt-seconds per square centimeter. The lamp should be located about 2.5 centimeters (1 inch) above the chip during erasure. After erasure, all bits are at a high level. Note that normal ambient light contains the correct wavelength for erasure; therefore, when using the TMS77C82JDL the window should be covered with an opaque label.

9.3 Serial Communication with the TMS7000 Family

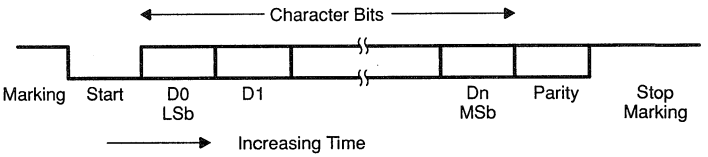
This section discusses using the TMS7000 for serial communication with a UART (Universal Asynchronous Receiver Transmitter). It describes implementing the UART function in software using any TMS7000 device and with the on-chip serial port using the TMS7xCx2.

9.3.1 Communication Formats

The TMS7000 family handles three basic formats of serial communication — asynchronous, isosynchronous and serial I/O. The first two require framing bits to be added to the data, allowing the receiver to properly detect incoming data. The last two require an addition serial clock to synchronize the data. This UART routine uses asynchronous communications; all the formats are discussed in detail in Chapter 3.

In **asynchronous format**, as shown in Figure 9–4, each character to be transmitted is preceded by a start framing bit and followed by a parity bit (if parity is enabled), then one or more stop framing bits.

Figure 9–4. Asynchronous Communication Format



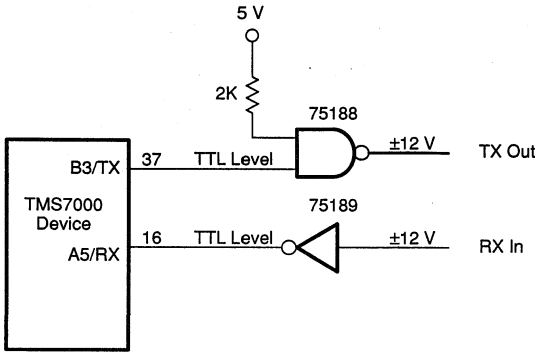
The **start** bit is a logical 0, or **space.** It notifies the receiver to start assembling a character and allows the receiver to synchronize itself with the transmitter.

A **parity** bit is an additional bit added to a character for error checking. The parity bit is set to 0 or 1 in order to make the number of 1s in the character (including the parity bit) even or odd depending on whether even or odd parity is selected.

The **stop** bit is a logical 1 or **mark**. One or more **stop** bits are added to the end of the character to ensure that the **start** bit of the next character will cause a transition on the communication line.

The connections for both the software and on-chip hardware UARTs are identical. Both use A5/RX for the incoming data and B3/TX for outgoing data. The connections are shown in Figure 9–5. The TMS7000 outputs a TTL-level signal which must be converted to ±12 volts for RS-232-C compatibility. The 75188 and 75189 devices are used for this purpose.

Figure 9–5. I/O Interface

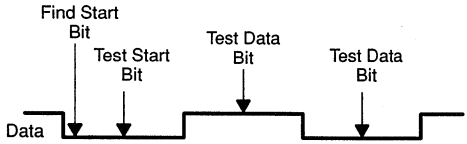


9.3.2 Software UART (All TMS7000 Devices)

This software UART routine will run on any TMS7000 family microcomputer. It requires the use of one timer to produce a consistent baud rate without requiring full use of the program's time. This UART will run mainly in the Interrupt-2 routine, allowing the main program to run independently of the UART.

The timer is configured so that the interrupts arrive every half bit. This is because the receiver section must find the start bit as soon as possible, but it must also test the following bits at the middle of the bit. Testing at the edge of a bit time would produce data errors. Figure 9–6 shows the start bit detection.

Figure 9–6. Start Bit Detection



The software, which consists of a receiver routine and a transmitter routine, runs mainly during the Interrupt-2 routine. Both routines maintain a progressive state counter, which will have one of the following values to indicate its condition:

State 0 The receive portion is in this state until a low start bit is detected.

State 1 This state begins a half bit later and tests for a valid start bit.

State 2 and State 3 The 8 character bits are built in states 2 and 3.

State 4 and State 5 The parity bit is received in states 4 and 5. If the parity does not agree with the parity of the input byte then a bit is set to indicate a parity error.

State 6 and State 7 These states look for the stop bit. If the stop bit is not found, then another bit is set to indicate a framing error. The complete character is then placed in the RXSTOR register and a bit is set to indicate to the main program that a character is ready to be read. The main program must clear the parity and framing error bits.

The transmitter routine operates similarly to the receiver routine, using a separate state counter to record its condition. The transmitter routine skips every other interrupt because the routine can be entered every full bit instead of every half bit, as in the receiver routine. The transmitter sends out bytes stored in a table. This table can be in either ROM or RAM and the table ends with a >FF to signify the end of string. Parity is calculated for both the receiver and transmitter by exclusive ORing the data bits together to produce even parity for the string.

9.3.2.1 Software UART Enhancements

If it is not necessary for the transmitter and receiver to run simultaneously, then several improvements can be implemented.

- The transmitter's baud rate can easily be doubled by interrupting every full bit instead of every half bit.
- The receiver can be improved by connecting the RX-in line to RX and to an Interrupt pin ($\overline{INT}1$ or $\overline{INT}3$). When the start bit is detected, the program enters the external interrupt routine. This interrupt routine must start the timer to count out one-half bit and also disable the interrupt. When the half-bit interrupt occurs, the timer must be reset and restarted to produce a full-bit interrupt; this would occur in the middle of the data bits.
- The parity can be selected by testing an even/odd bit and setting the initial parity register (TXPAR, RXPAR) to the correct value. Currently, the registers are cleared for every new byte, producing even parity.
- An extra stop bit could be added by using a test bit and repeating states 6 and 7 if the bit is set.
- Additional RS-232-C signals could be added to the program to interface to more complex equipment.

9.3.2.2 Software SWUART.ASM Routines

```
*
*  PROGRAMS TRANSMITS AND RECEIVES SERIAL DATA
*  Simultanously transmitts and receives RS-232-C format
*  data.
*
*  Transmitt pin out = B3
*  Receiver  pin in  = A5
*
***** REGISTER   FILE ************************************
*
*       U A R T      R E G I S T E R S
*
STATER  EQU     R2    The state of the current receive data
STATET  EQU     R3    The state of the current transmitt data
RXBUF   EQU     R4    Build the input byte here
RXCNT   EQU     R5    The number of bits left to receive
RXSTOR  EQU     R6    Pick up the finished input word here.
RXPAR   EQU     R7    Bit 0=parity (7 other bits free)
TXCNT   EQU     R8    The number of bits left to transmitt
TXTABL  EQU     R9    Address offset from String beginning
TXBUF   EQU     R10   Shift the out word from here.
TXPAR   EQU     R11   Bit 0 = parity ( 7 other bits free)
BITS    EQU     R12   Bit0= Transmit routine now or next int.
*                     Bit1= Transmitter active now
*                     Bit2= Receiver contains word now
*                     Bit3= Framing error ( bad stop bit)
*                     Bit4= Finished with the string output.
*                     Bit5= Parity error
ENDFIL  EQU    R13  ! Last received character +1 (>FF eof)
*
***** PERIPHERAL FILE ************************************
*                          PERIPHERAL PORTS AND REGISTERS
IOCTL0 EQU    P0      Interrupt control 0
T1MSB  EQU    P12     Timer 1 MSDATA          (not on TMS70Cx0)
T1LSB  EQU    P13     Timer 1 LSDATA          (P2 on  TMS70Cx0)
T1CTL0 EQU    P15     Timer 1 Control 0       (P3 on  TMS70Cx0)
T1CTL1 EQU    P14     Timer 1 Control 1       (not on TMS70Cx0)
PORTA  EQU    P4      Port A data
PORTB  EQU    P6      Port B data
ADDR   EQU    P5      Port A data direction register
*
***********  CONTROL CONSTANTS  FOR PORT A  ***************
*
*                              BAUD RATE
* CRYSTAL              300  600  1200  2400  4800  9600
* 5 MHz     Latch      129   63   129    64    32    15
* TMS7XCX2 Prescale     15   15     3     3     3     3
* TMS70CX0 Prescale      3    3     0     0     0     0
*                          CRYSTAL FREQ
*         (L+1)*(PS+1)= ------------------
*                       (BAUDRATE * 2) * 4
*
BAUD1  EQU    129                 Value for the timer latch
BAUD2  EQU    >80+3               Value for the timer control register
*
*TRING EQU    0
*
BIT0   EQU    1                   Various bit constants to make code more
BIT1   EQU    2                   readable
```

```
BIT2    EQU     4
BIT3    EQU     8
BIT4    EQU     16
BIT5    EQU     32
BIT6    EQU     64
BIT7    EQU     128
*
************************************************************
*
        AORG    >F806
*
START   DINT                        Disable all interrupts
        MOVP    %>00,T1CTL1         B1 will not toggle when T1 decrements through 0
        MOV     %>FD,B              Set index to clear out
        CLR     A                   all of RAM
CLEAR   STA     @1(B)               Store 0s into all of ram
        DJNZ    B,CLEAR             Loop until RAM is all 0s
        MOV     %>60,B              Set stack pointer
        LDSP
*       MOV     %BIT1,BITS          Active transmittter and initialize
        MOV     %>FF,R128           !End of string for test
        CLR     ENDFIL              !location of the end of string
*                                   receiver
        MOVP    %?00101110,IOCTL0   Enable Timer interrupt
        MOVP    %?00000000,PORTA    Clear Port A
        MOVP    %?00000000,ADDR     Initialize A5 for input
        MOVP    %?00001000,PORTB    Initialize Port B
        MOVP    %BAUD1,T1LSB        Put the baud rate into the
        MOVP    %>00,T1MSB
        MOVP    %BAUD2,T1CTL0       timer latch and timer control
        EINT                        Start looking for interrupts
*
LOOP    IDLE                        Wait for timer interrupt or
        BTJZ    %BIT2,BITS,LOOP     New word waiting?
        AND     %#BIT2,BITS         Clear out bits
        MOV     ENDFIL,B            Go to the last byte written
        INC     B
        OR      %BIT7,B             Limit to upper half of RAM
        MOV     %>FF,A              end of file character
        STA     @0(B)               store end of file character
        DEC     B                   back up one
        OR      %BIT7,B             Limit to upper half of RAM
        MOV     RXSTOR,A            store received character in RAM
        STA     @0(B)
        OR      %BIT1,BITS          Turn on transmitter
        INC     ENDFIL
        JMP     LOOP                execute main program here
*
*****************************
*
*       T I M E R   1   I N T E R R U P T
*
INTER2  EQU     $                   Start of timer interrupt
        PUSH    A                   Store registers
        PUSH    B
        MOV     STATER,B            Get current receiver state
        RLC     B                   Double in preparation for jump
        CALL    @JUMPR(B)           Go do receiver things.
*
        BTJZ    %BIT1,BITS,OUT      Is a word being transmitted now?
        XOR     %BIT0,BITS          Do only every other interrupt.
```

```
        BTJZ   %BIT0,BITS,OUT      Transmitt one out of two interrupts.
        MOV    STATET,B            Move transmitter state to index
        RLC    B
        CALL   @JUMPT(B)           Go to the proper state of routine.
*
OUT     POP    B
        POP    A                   Restore the registers
        RETI                       Exit the routine
*
*       R E C E I V E R   J U M P   T A B L E
*
JUMPR   JMP    STATE0              Check for start bit
        JMP    STATE1              Check for Half a start bit
        JMP    STATE2              Bit boundry, wait for 1/2 bit
        JMP    STATE3              Test input for Data
        JMP    STATE2              Parity bit boundry
        JMP    STATE5              Check parity bit
        JMP    STATE2              Stop bit boundry
        JMP    STATE7              Check middle of the stop bit
*
STATE0  BTJOP  %BIT5,PORTA,ISPACE     Is the Receive line low,
        INC    STATER              if so new start bit, go to
ISPACE  RETS                       next state, if not do nothing.
*
STATE1  BTJZP  %BIT5,PORTA,ISTART  Check for false starts.
        CLR    STATER              Clear state if false start.
        RETS
ISTART  MOV    %8,RXCNT            Number of bits to receive
        AND    %#BIT0,RXPAR        Initialize parity
        INC    STATER              Go to state 2
        RETS
*
STATE2  INC    STATER              Half bit, go to next state
        RETS
*
STATE3  BTJZP  %BIT5,PORTA,BITLOW  Input new bit
        SETC                       A one was found
BITLOW  RRC    RXBUF               Build the input word here
        XOR    RXBUF,RXPAR         Build up even parity
        DEC    STATER              Goto half state
        DJNZ   RXCNT,OUTP3         Is entire byte in?
        MOV    RXBUF,RXSTOR        Store byte in storage register
        MOV    %4,STATER           Goto state 4
OUTP3   RETS
*
STATE5  BTJO   %BIT0,RXPAR,IS1     Check for even parity (JZ for odd)
IS0     BTJZP  %BIT5,PORTA,OUTPAR  Out if both parities 0?
BADPAR  OR     %BIT5,BITS          Bit 5= Parity error
        JMP    OUTPAR
IS1     BTJZP  %BIT5,PORTA,BADPAR  Continue if both parities =1
OUTPAR  INC    STATER              Reset state counter
        OR     %BIT2,BITS          Set 'Word ready' bit.
        RETS
*
STATE7  BTJOP  %BIT5,PORTA,ISSTOP  Look at the stop bit, =1?
        OR     %BIT3,BITS          Bit 3= Framing error
ISSTOP  CLR    STATER              Reset state counter
        RETS
*
* T R A N S M T T I N G   S E C T I O N
*
```

```
*
*       T R A N S M I T T E R    J U M P     T A B L E
*
JUMPT   JMP   STATEA              Start outputting string
        JMP   STATEB              Output start  bit
        JMP   STATEC              Output data   bits
        JMP   STATED              Output parity bit
        JMP   STATEE              Output stop   bit
*
STATEA  CLR   TXTABL              Initialize table pointer
        CALL  @FIRST              Load the first byte into buffer
STATEB  ANDP  %#BIT3,PORTB        Send out a start bit
        MOV   %8,TXCNT            8 bits per character
        AND   %#BIT0,TXPAR        Initialize parity to 0
        INC   STATET              Go to the next state
        RETS
*
STATEC  XOR   TXBUF,TXPAR         Build up parity bit
        BTJZ  %BIT0,TXBUF,TRANS0  Send a 1 or a 0?
        ORP   %BIT3,PORTB         Output a 1 bit.
        JMP   NXTBIT
TRANS0  ANDP  %#BIT3,PORTB        Output a 0 bit.
NXTBIT  RR    TXBUF               Point to the next bit.
        DJNZ  TXCNT,OUTC          Are all 8 bits done yet?
        INC   STATET              Output stop bits next
OUTC    RETS
*
STATED  BTJZ  %BIT0,TXPAR,PARTY0  Output Even Parity (JO for odd)
        ORP   %BIT3,PORTB         Output a 1 bit.
        JMP   OUTD
PARTY0  ANDP  %#BIT3,PORTB        Output a 0 bit.
OUTD    INC   STATET              Output stop bit next
        RETS
*
STATEE  ORP   %BIT3,PORTB         Send out a stop bit
        MOV   %1,STATET           Send out start bit next
        OR    %BIT0,BITS          Enter TX routine every other int.
*
        INC   TXTABL              Point to next byte from table
FIRST   MOV   TXTABL,B            Setup output table pointer.
        LDA   @STRING(B)          Get value from table.
        MOV   %1,STATET           Output Start bit next
        CMP   %>FF,A              FF = end of string
        JNE   NEWTX               Jump if not end of string
        OR    %BIT4,BITS          End of text string, Set bit
        AND   %#BIT1,BITS         Turn off transmitter
        DEC   STATET              Start at beginning next time.
NEWTX   MOV   A,TXBUF             Store new byte
        RETS
*
*******************************************************
*
*             This text string could be in RAM or ROM.
STRING  TEXT  'ABCDEFGHIJKLMNOPQRSTVUWXYZ1234567890'
        BYTE  >FF                 end of string byte
*
INTER1  EQU   $
INTER3  RETI
*
        AORG  >FFF8
        DATA  INTER3,INTER2,INTER1,START
```

9.3.3 Hardware UART (TMS7xCx2)

The main portions of the serial port are the receiver (RX), transmitter (TX), and timer (T3). The complete functional definition of the serial port is configured by the user program. A set of control words must first be sent out to configure the serial port. For more information about the serial port, see Chapter 3.

The serial port is controlled and accessed through registers in the peripheral file. The registers associated with the serial port are shown in Table 9–6.

Table 9–6. Serial Port Control Registers

Register TMS70Cx2	Name	Type	Function
P20	SMODE	FIRST WRITE	Serial port mode
P21	SCTL0	READ/WRITE	Serial port control 0
P22	SSTAT	READ	Serial port status
P23	T3DATA	READ/WRITE	Timer 3 data
P24	SCTL1	READ/WRITE	Serial port control 1
P25	RXBUF	READ	Receiver buffer
P26	TXBUF	WRITE	Transmission buffer

The hardware serial port program is divided into three sections:

1) The initialization section
2) The transmitter section
3) The receiver section

The transmitter and the receiver sections are in the serial-port interrupt service routine. The main body of the program follows the initialization section and runs between interrupts.

9.3.3.1 Initialization

The program first initializes all registers, starting with the interrupt control registers IOCNT0 and IOCNT1. The stack pointer is set and output ports A and B are initialized.

Next, the serial port registers are set up.

Finally, the serial port timer is started and the interrupts are enabled. The processor then waits for the timer interrupt to service the serial port. Faster baud rates allow less time for the main program to run, since it only runs between the interrupts.

INT4 is dedicated to the serial port. Three sources can generate an interrupt through INT4: the transmitter (TX), the receiver (RX), and Timer 3 (T3). The serial port can be driven by Timer 3 or external baud rate generator. The Timer 3 interrupt function is usually disabled when using the UART because the timer will interrupt 16 times for every bit or about 160 times per byte. In this HWUART

program, the T3 interrupt is disabled and the internal Timer 3 is chosen as the serial clock.

9.3.3.2 Transmitter

When the program enters the serial port interrupt routine, it determines if the transmitter or receiver caused the interrupt. If the interrupt occurred because the transmitter is empty, then the program takes the next byte in the transmitter table and places it in the transmitter buffer. The first byte of the transmitter data contains the total number of bytes in the string. If the index is zero, the program places this byte count into the index register instead of transmitting it. This is an alternate method to the software UART's example of ending the string with a unique character.

9.3.3.3 Receiver

If the receiver causes an interrupt and no errors exist, then the program takes the value in the receiver buffer and places it into a receiver table. After the character is placed into the table, the character counter at the beginning of the table is updated. The main program must take this data and reset the character count before the RAM buffer becomes full. This is an alternate method to the software UART's example of putting the value in a register and setting a flag for the main program.

9.3.3.4 Error Conditions

If the program detects an error condition in the serial port status register, then the program sets a bit in RAM for the main program body to detect. When the main program detects this error bit, it looks at SSTAT to determine the cause of the error and takes action (if necessary). The main program may cause the byte to be retransmitted, if necessary.

9.3.3.5 Baud Rates

The baud rate generated by Timer 3 is user-programmable and is determined by the value of the 2-bit prescaler and the 8-bit timer reload register. The serial port discussion in Chapter 3 provides a table of common baud-rate values.

9.3.3.6 RS-232-C Interface

The RS-232-C interface consists of SN75188 line drivers and SN75189A line receivers as shown Figure 9–5. This is the same interface circuit used in the software example. Port A5 (input) of the TMS7xCx2 is used for all data receptions, and port B3 (output) is used for all data transmissions.

9.3.3.7 Hardware UART Routines

```
        TITL   'HARDWARE SERIAL PORT EXAMPLE'
        IDT    'HWUART'
*       This program uses the onboard UART to simultanously
*       transmitt and receive characters.  Characters for
*       transmitting are placed starting at TTABLE with
*       the first byte equal to the string byte count.  The
*       received bytes are stored in the table RTABLE with
*       the beginning byte equal to the characters received.
*--------------------------------------------------------------
* Peripheral Register Definition  TMS7XCX2
*--------------------------------------------------------------
IOCNT0  EQU   P0        Interrupts and mode control
PORTA   EQU   P4        Port A-UART input
ADDR    EQU   P5        Port A direction
PORTB   EQU   P6        Port B-UART output
IOCNT1  EQU   P2        Interrupt 4,5 control
SMODE   EQU   P20       Serial port mode
SCTL0   EQU   P21       Serial port control-0
SSTAT   EQU   P22       Serial port control status
T3DATA  EQU   P23       Timer 3 data
SCTL1   EQU   P24       Serial port control-1
RXBUF   EQU   P25       Receiver buffer
TXBUF   EQU   P26       Transmitter buffer
*--------------------------------------------------------------
* Register Definition
*--------------------------------------------------------------
POINTR  EQU   R5        Pointer into receiver table.
POINTT  EQU   R6        Number of bytes ready to send.
POINTC  EQU   R7        Transmitter chars send so far.
BITS    EQU   R8        Store random conditional bits here
RTABLE  EQU   030       Beginning of receiver table.
*
BIT0    EQU   1         Bit constants to make code more
BIT1    EQU   2         readable.
BIT2    EQU   4
BIT3    EQU   8
BIT4    EQU   16
BIT5    EQU   32
BIT6    EQU   64
BIT7    EQU   128
*--------------------------------------------------------------
*
        AORG  >F006
START   DINT                          Disable interrupts, (precaution)
        MOVP  %>2A,IOCNT0             Single chip, clear INT flags
*                                     Disable I1, I2, I3 MOV   %>60,B
        MOVP  %>03,IOCNT1             Clear INT4 flag and enable INT4
        MOV   %>60,B
        LDSP                          Initialize stack pointer
        MOVP  %#BIT2,ADDR             Set A2 = input others are output
        MOVP  %BIT3,PORTB             Enable TX by setting B3=1

        MOVP  %BIT6,SCTL0             Reset the UART via the UR bit
        MOVP  %?01111110,SMODE        One stop bit, communications
*                                     Mode, even parity, 8 bits,
*                                     Asynchronous mode, Motorola
        MOVP  %>15,SCTL0              Clear the serial port reset bit
*                                     Clear all error flags and
*                                     Enable transmitter, receiver
```

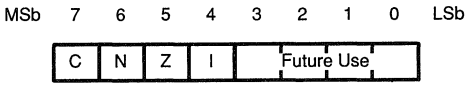
```
       MOVP   %65,T3DATA          Set timer at 1200 baud (5.0688MHZ)
       MOVP   %>00,SCTL1          Make sure the start bit is off.
       MOVP   %>C0,SCTL1          Use internal CLK, reset T3FLAG
*                                 Disable T3 interrupt & set PS=0
       EINT                       Enable maskable interrupt
*
SETUP  CLR    POINTC              Clear bytes transmitted count.
       MOV    %28,POINTT          INIT bytes to transmit.
       CLR    POINTR              Clear bytes received count.

*--------------------------------------------------------
*** Main body of program goes here
*
*
*** Main body finds and corrects serial port error
*   conditions by checking Bit 0 of 'BITS' and SSTAT.
*--------------------------------------------------------
* INTERRUPT 4 SERVICE ROUTINE
*--------------------------------------------------------
INTER4 BTJOP %>38,SSTAT,ERROR  Was there an error?
       JMP    SAVEIT
ERROR  OR     %BIT0,BITS          Set an error bit for the main
*                                 program to find and continue.
SAVEIT PUSH   A                   Save register A
       PUSH   B
       BTJZP  %BIT1,SSTAT,TXOUT Did receiver cause interrupt?
*
RXCV   INC    POINTR              Get receiver table pointer
       CMP    %30,POINTR          Is receiver table full yet?
       JHS    TXOUT               get out of routine if so
       MOV    POINTR,B            Get index value
SKIP1  MOVP   RXBUF,A             Put received character to A
       STA    @RTABLE(B)          Put value into table
       MOV    B,A                 Store the new character count
       STA    @RTABLE             Put count at location 0 in
*                                 table and exit.
TXOUT  BTJZP  %BIT0,SSTAT,OUTI4  Did XMIT cause interrupt?
*
XMIT   CMP    POINTT,POINTC       Is the table finished?
       JHS    OUTI5               Jump if finished.
VV     INC    POINTC              Point to the next index
       MOV    POINTC,B            Get transmit table pointer
SKIP0  LDA    @TTABLE(B)          Load value from TX table
       CMP    %0,B                Is this the byte count?
       JNE    OUTPUT              If not, output the byte
       MOV    A,POINTT            If so, put into pointer
       JMP    OUTI4
OUTPUT NOP                        Put data into transmitter
*
OUTI4  MOVP   A,TXBUF             Restore registers

       POP    B
       POP    A
```

```
        RETI                                        Return to main program
OUTI5   MOV   %>00,POINTC
        MOV   %28,POINTT
        JMP   VV
*
TTABLE  BYTE  26                                    Text can be either in ROM or
        TEXT  'ABCDEFGHIJKLMNOPQRSTUVWXYZ'          RAM registers.
        BYTE  >0D          CARRAGE RETURN
        BYTE  >0A          LINE FEED
*
        AORG  -(4+1)*2                              Set up 4 vectors
        DATA  INTER4,START,START,START,START        interrupt vectors
```

9.4 The Status Register

The status register contains four status bits that provide conditional execution for a variety of arithmetic and logical tasks. The carry (C), negative (N), zero (Z), and interrupt enable (I) flags occupy bits 7–4 of the status register. The C, N, and Z bits are affected by most instructions. The global interrupt enable (I) bit is affected by the EINT, DINT, and POPST instructions.

Figure 9–7. Status Register



Section 9.4.1 describes the way in which the compare instructions can be used to create the necessary status conditions for either a logical-type (unsigned) or arithmetic-type (signed) jump instruction. Subsection 9.4.2 describes the effects of addition and subtraction on the status register for both signed and unsigned systems. Finally, subsection 9.4.3 describes how SWAP and the rotation instructions (RR, RRC, RL, and RLC) can be used to clear, set, shift, or test the various status bits as required.

9.4.1 Compare and Jump Instructions

The compare instructions, CMP and CMPA, affect the C, N, and Z bits in the status register by subtracting a source operand (S) from a destination operand (d). Destination and source may be misnomers in this case, because the result of (d) – (s) is not stored; however, the status bits are set according to the result of the subtraction.

C Serves as a "no-borrow" bit. If (d) is greater than or equal to (s), then there is no borrow and C is set to 1. C is set to 0 if (d) is less than (s).

N Is set to the same value as the MSb of the result. For 2's complement (signed) systems, N = 1 indicates a negative number, and N = 0 indicates a positive number.

Z Is set to 1 if the source is equal to the destination [(d) = (s)].

The CMP instruction uses the contents of a register (Rn) as the destination operand, and either an immediate operand or the contents of another Rn as the source operand. The CMPA instruction uses the contents of register A as the destination operand and one of the extended addressing modes (direct, register file indirect, or indexed) generates the source operand. Table 9–7 illustrates the limits of both signed and unsigned systems by listing the status bits affected for various source and destination operands substituted into the (d) – (s) expression.

Table 9–7. Compare Instruction Examples: Status Bit Values

Source	Destination	D–S	C	N	Z	Instructions That Will Jump					
FF	00	01	0	0	0	JL	JNC	JNE	JNZ	JP	JPZ
00	FF	FF	1	1	0	JHS	JC	JNE	JNZ	JN	
00	7F	7F	1	0	0	JHS	JC	JNE	JNZ	JP	JPZ
81	00	7F	0	0	0	JL	JNC	JNE	JNZ	JP	JPZ
00	81	81	1	1	0	JHS	JC	JNE	JNZ	JN	
80	00	80	0	1	0	JL	JNC	JNE	JNZ	JN	
00	80	80	1	1	0	JHS	JC	JNE	JNZ	JN	
7F	80	01	1	0	0	JHS	JC	JNE	JNZ	JP	JPZ
80	7F	FF	0	1	0	JL	JNC	JNE	JNZ	JN	
7F	7F	00	1	0	1	JHS	JC	JEQ	JZ	JPZ	
7F	00	81	0	1	0	JL	JNC	JNE	JNZ	JN	

Since the compare instructions do not alter the source and destination operands, these instructions can be executed before a conditional jump instruction to test for a particular relationship between the source and destination operands. Table 9–8 lists the necessary status bit conditions for each of the conditional jump instructions.

Table 9–8. Status Bit Values for Conditional Jump Instructions

Mnemonic	Instruction	Condition on Which Jump Is Taken	Status Bit Values For Jump:		
			C	N	Z
JC/JHS	Jump if carry/jump if higher or same	(d) unsigned >= (s)	1	X	X
JNC/JL	Jump if no carry/jump if lower	(d)unsigned<(s)	0	X	X
JZ/JEQ	Jump if zero/jump if equal	(d) = (s)	X	X	1
JNZ/JNE	Jump if non-zero/jump if not equal	(d) ≠ (s)	X	X	0
JP	Jump if positive	(d) – (s) = pos #	X	0	0
JN	Jump if negative	(d) – (s) = neg #	X	1	X
JPZ	Jump if positive or zero	(d) – (s) = pos # or 0	X	0	X

Note: X = Don't Care

9.4.2 Addition and Subtraction Instructions

The TMS7000 instruction set supports both single and multiprecision addition and subtraction for either binary or BCD, signed (2's complement) or unsigned data.

The following example illustrates 32-bit addition with the ADD and ADC instructions:

```
ADD  R30,R120
ADC  R29,R119
ADC  R28,R118
ADC  R27,R117
```

Since no initial carry-in is desired, the first instruction is ADD. The ADC instruction is then executed three times in succession to transfer the carry through all 32 bits.

The following example illustrates 24-bit subtraction with the SUB and SBB instructions:

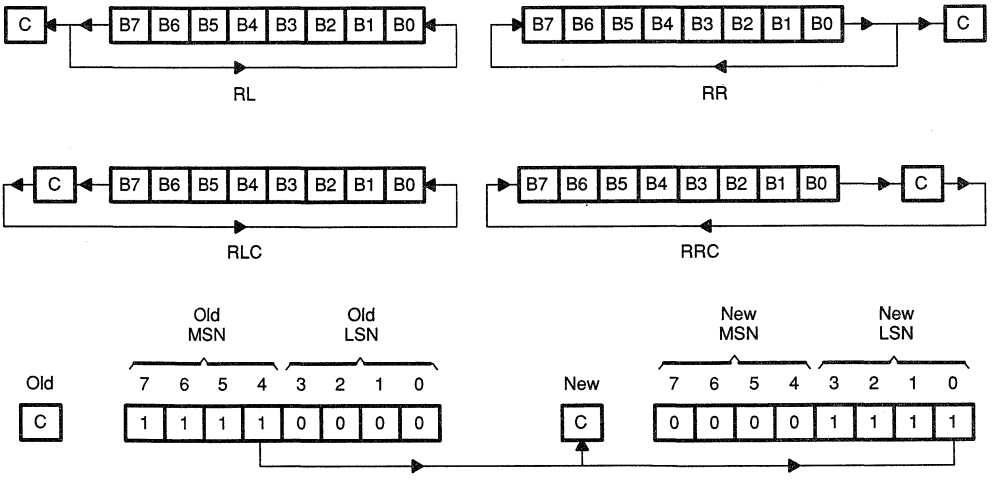
```
SUB  R4,R127
SBB  R3,R126
SBB  R2,R125
```

Since no initial borrow-in is desired, the first instruction is SUB. The SBB instruction is then executed twice in succession to achieve the 24-bit result.

9.4.3 Swap and Rotation Instructions

Figure 9–8 illustrates the rotation operations performed by the four rotation instructions rotate right (RR), rotate right through carry (RRC), rotate left (RL), and rotate left through carry (RLC), and the SWAP instruction. SWAP executes the equivalent of four consecutive RL instructions, setting the C bit in the status register equal to bit 4 of the original operand or bit 0 (LSb) of the result.

Figure 9–8. Swap and Rotation Operations



9.5 Stack Operations

The stack is located in RAM and can be tailored to your needs. One powerful application of the stack is the establishment of tables. For example, Figure 9–9 illustrates a dispatch table with an interpretive program counter (IPC). An IPC is used in some high level languages, such as Pascal, to give the proper execution sequence. The IPC can be contained in any register; it points to an interpretive pseudo code (pcode) byte that in turn specifies one of 256 dispatch routines. The overall effect of this function is that a program can execute one of a large number of different routines depending on a single value stored in a register.

Figure 9–9. A Dispatch Table with an Interpretive Program Counter (IPC)

```
IPC     EQU   R3                    Interpretive Program Counter
        LDA   *IPC                  Get the input code range=0-127
        DECD  IPC                   Point to next input code
        RL    A                     Double pointer for word table
        MOV   A,B                   Move to index register
        LDA   @DTABLE(B)            Lookup MSB address of
        PUSH  A                     Put MSB on stack
        LDA   @DTABLE+1(B)          Lookup LSB address of
        PUSH  A                     Put LSB on stack
        RETS                        Jump to address on stack

DTABLE  DATA  ROV0,ROV1,ROV2,ROV3   address of routines
        DATA  ROV4,ROV5,ROV6,ROV7   address of routines
```

Note that the assembler expressions have 16-bit values. For those instructions requiring an 8-bit operand, the expression is truncated to the least significant 8 bits. This may produce a warning message, but the value will be correct. Thus, the following instructions place byte values >AA, >55, and >55 at memory locations >8000, >8001, and >8002, respectively:

```
       AA55  LABEL  EQU   >AA55
8000                AORG  >8000
8000   AA55         DATA  LABEL     16-bit word
              *                    LSB only
8002   55           BYTE  LABEL-(LABEL/256*256)
```

The most significant byte (MSB) of an expression can be obtained by dividing the value by 256 (2^8) as shown below:

```
       AA55  LABEL  EQU   >AA55
8000                AORG  >8000
8000   AA55         DATA  LABEL
8002   AA           BYTE  LABEL/256   MSB only
```

9.6 Subroutine Instructions

Two instructions, CALL and TRAP, can invoke subroutines. TRAP is a one byte subroutine call. Both instructions save the current value of the program counter (PC) on the stack before transferring control to the subroutine. Since the return address is stored on the stack, subroutines can be easily nested. The two instructions differ only in the way in which the subroutine address is determined and in the amount of program memory required for execution.

The CALL instruction uses the extended addressing modes (direct, register file indirect, and indexed) to specify the subroutine address. This permits simple calls with a fully specified address as well as more complex calls with a calculated address. Of the two types of instructions, the CALL instruction requires more program memory than the TRAP instructions. For example:

```
CALL  @BITTEST
```

requires three bytes of memory — one byte for the opcode and two bytes for the subroutine address. If the subroutine is called six times, 18 bytes are necessary to implement the CALLs. The equivalent task for the TRAP instruction requires only 8 bytes for six successive uses of the same TRAP, since only the opcode byte is necessary after the first use. Six of these 8 bytes are the TRAP opcodes and the other two bytes are the trap vector. The first use of the TRAP instruction requires one opcode byte plus the two bytes of the subroutine address which are located in the Trap Table. Each subsequent use requires only one more byte, compared to three bytes for each CALL. All the trap vectors are stored at the end of memory with the most significant byte of the trap subroutine stored in the lower numbered location. The exact address where the trap vector (which is the trap subroutine address) is stored is derived from the following formula.

LSB of Address which contains the TRAP subroutine address = >FFFF – 2 x N where N is the TRAP number.

```
MSB of address  = LSB - 1
```

The TRAP instructions (TRAPs 4–23) provide the most efficient means of invoking subroutines. Figure 9–10 shows a subroutine call generated by a TRAP instruction.

Figure 9–10. Example of a Subroutine Call by Means of a TRAP Instruction

```
                .
                .               (Main Program)
            TRAP  4
                .               (More Main Program)
                .
            BR    MAINPR
                .
                .
BITTEST     EQU   $
                .
                .               (Subroutine Body)
            RETS
                .
                .
            AORG  >FFF6 Trap 4 vector
            DATA  BITTEST
                .
                .
```

The return from subroutine (RETS) instruction should be executed to pop the PC from the stack and restore program control to the instruction immediately following the CALL or TRAP instruction.

9.7 Multiplication and Shifting

The MPY instruction performs an 8-bit by 8-bit multiply and stores the 16-bit result in registers A and B. The most significant byte (MSB) of the result is in register A, and the least significant byte (LSB) is in register B. The MPY instruction can also be used to perform multi-bit right or left shifts by using an immediate operand as the multiplier. For example:

```
MPY  %8,B
```

The preceding example multiplies the value of register B by 8. After the instruction executes, register B contains the previous value left-shifted three bits ($2^3 = 8$) with no fill bits. Register A contains the previous value's most significant three bits which produces a value equivalent to shifting the previous value right five bits ($8 - 3 = 5$) with no fill bits. Using this method, it is possible to shift any 8-bit value left or right up to 8 bits. In many cases this is faster than the rotate instructions and almost always takes less program bytes. If the word only needs to be shifted one or two places then the rotate instructions may take less execution time. Table 9–9 lists the number of bits right- or left-shifted for a range of immediate multipliers.

Table 9–9. Multi-Bit Right or Left Shifts by Immediate Multiply

Immediate Multiplier	Bits Right Shifted	Bits Left Shifted
2	7	1
4	6	2
8	5	3
16	4	4
32	3	5
64	2	6
128	1	7

Multiprecision multiplications can be easily executed by breaking the multiplier and the multiplicand into scaled 8-bit quantities, as shown in the examples at the end of this chapter.

9.8 The Branch Instruction

The branch instruction (BR) unconditionally transfers program control to any desired location in the 64K byte memory space. BR supports direct, indexed, and indirect addressing:

- Direct addressing is used for simple GOTO programming.
- Indexed addressing allows table branches. This indexed branch technique is similar to the Pascal CASE statement. Program control is transferred to location CASE0 if the input is 0, to CASE1 if it is a 1, etc. This transferring method can implement up to 85 different cases. In the example below, indexed addressing is used to access a relative branch table:

```
JTABLE   MOVP P4,A        Get data from A port
*                         (value < 85)
         ADD  A,B         Add twice to triple value
         ADD  A,B         Multiply it by 3
*  (BR is 3 bytes long)
         BR   @CTABLE(B)  Branch according to the
*                         A port value * 2
         .
*
CTABLE   BR   @CASE0      If P4 = 0 do this branch
         BR   @CASE1      If P4 = 1 do this branch
         BR   @CASE2      If P4 = 2 do this branch
*
```

- The branch instruction can also be used with indirect addressing in order to branch to a computed address. For example, suppose that a computed branch address has been constructed in R19 and R20. The desired program control transfer is made by:

```
BR   *R20
```

9.9 Interrupts

The number of interrupts and the hardware configuration for a TMS7000 family device are specified in Chapters 2 and 3. The TMS7020, for example, has three interrupts in addition to RESET.

RESET and the interrupts are vectored through predetermined memory locations. RESET uses the TRAP 0 vector which is stored at memory locations >FFFE and >FFFF. The interrupts also use the TRAP vector table with $\overline{\text{INT}}$1 using the TRAP 1 vector, etc. Thus, the TRAP 2 instruction involves the same code as the interrupt INT2.

The interrupts differ from the TRAPs; they push the status register value on the stack, clear the interrupt enable bit in the status register, and reset the corresponding interrupt flag bit. Thus the EINT instruction must be used if nested interrupts are desired. The return from interrupt (RETI) instruction restores the status register and the program counter, re-enabling interrupts.

Many interrupt service routines alter the status of key registers such as registers A and B. These routines should use the stack to restore the machine state to the desired value. For example, the following interrupt routine performs an I/O driven table look-up. Registers A and B are used, but their values are saved and restored:

```
INT   PUSH  A           Store Registers A and B on stack
      PUSH  B
      MOVP  P4,B        Get input from Port A
      LDA   @LOOKUP(B)  Do a table lookup to get new value
      MOVP  A,P6        Output new value on Port B
      POP   B           Restore Registers A and B in the
      POP   A           reverse order that they were put
*                       on
      RETI              Back to main program
```

All interrupts are usually disabled during an interrupt service routine. If it is necessary for an interrupt to occur while the processor is servicing another interrupt, then the global interrupt enable bit should be set to 1 by the interrupt service routine. The number of interrupts that can be serviced at any one time is determined by the size of the stack, which is also the internal RAM size (the stack resides in the register file). Since other registers and data will most probably share the same space, the stack size is usually much less. When nesting interrupts, great care must be taken to avoid corrupting the data in the registers used by the most recent routine. If $\overline{\text{INT}}$1 interrupts an ongoing INT1 service routine, then the registers used by the INT1 routine are used in two different contexts. If provisions are not made for these situations, such as disabling all interrupts at critical times, then data errors will occur.

Sometimes a program contains distinct portions that require different responses to the same interrupt call. Since the interrupt vector is always set in nonchangeable ROM, another method must be used to change the vector for each part. One method for accomplishing this is to store a second vector in a

RAM register pair and allow the first instruction in the interrupt routine execute an indirect branch on that register.

```
* Program to demonstrate multiple interrupt service
* routine locations.
* Main Program
*
        MOVD  %SERVIC,R127   Put INT1 service routine
        EINT                 address in register
        IDLE                 Turn on and wait for
*                            interrupts
        MOVD  %SERVI2,R127   Change INT1 routine to
*                       SERVI2
         .
         .
*
* First Interrupt 1 Service Routine
SERVIC  PUSH  A              Beginning of the INT1
        PUSH  B              service routine for
*                            this part of the program
         .
         .
*
* Second Interrupt 1 Service Routine
SERVI2  PUSH  A              Start of another interrupt
        DEC   R4             1 service routine
         .
         .
*
INT1    BR    *R127          The entire INT1 service
*                            routine tranfers control
*                            to the address which is
*                            in R127 and R126
*
* Interrupt vector table at end of memory
        AORG  >FFFC
        DATA  INT1           Address of Interrupt 1
*                            service routine
        DATA  >F806          Reset vector start of
*                            program
```

9.10 Write-Only Registers

Certain TMS7xCx2 peripheral registers are write-only registers, which means that the program cannot directly ascertain the contents of the register. Table 9–10 lists write-only registers.

Table 9–10. Write-Only Registers

Register	Location	Function	Register	Location	Function
IOCNT0	P0	Interrupts	T1CTL1	P14	Timer 1 Control 1
IOCNT1	P2	Interrupts	T2CTL0	P19	Timer 2 Control 0
T1DATA	P12&P13	Timer 1 Reload	T2CTL1	P18	Timer 2 Control 1
T2DATA	P16&P17	Timer 2 Reload	SCTL0	P21	Serial Port
T3DATA	P23	Timer 3 Reload	TXBUF	P26	Transmit Buffer
T1CTL0	P15	Timer 1 Control 0			

Problems may arise using some instructions with these write-only registers because most have a separate read-only function at the same address. An error may occur when you execute an instruction that reads the register, modifies the value and then writes back to the register. These instructions are ANDP, ORP, XORP. For instance, the program cannot turn on the timer by ORing a 1 to the timer start bit, because the instruction will read the capture latch, set the MSb to 1, and then write this value to the timer control register. Unfortunately, this will change the prescaler and the timer may wait forever for a nonexistent external clock source.

The solution to this problem involves **image registers** which store the contents of a write-only register. An image register is a RAM register set aside to contain the value of a particular register. Whenever the write-only PF register must be changed, the program first fetches its image register, changes it, and then writes the image register to the peripheral register. This way, the image register always contains the value of the peripheral register. The following code using an image register could be used to turn on the timer start bit.

```
        OR    %>80,T1CTLI      Turn on start bit of
*                              timer control
        MOV   T1CTLI,A
        MOVP  A,T1CTL          Move the image register
*                              to the Peripheral File
```

9.11 Sample Routines

The following sections contain sample routines to show the various ways the TMS7000 handles common software tasks. Actual programs usually contain a combination of simple routines such as these along with custom routines tailored to the applications.

9.11.1 Clear RAM

This routine clears all the internal RAM registers. It can be used at the beginning of a program to initialize the RAM to a known value.

Register	Function
A	Holds the initialization value
B	Index into the RAM

```
        AORG   >F006
*
CLEAR   MOV    %254,B    Number of register to clear - 2
        CLR    A         Load the initialization value of
*                        zero
LOOP    STA    @2(B)     Clear the location indexed by
*                        B + 2
        DJNZ   B,LOOP    Loop until all RAM is cleared
```

9.11.2 RAM Self Test

This routine performs a simple alternating 0/1 test on the RAM. The RAM is tested by writing a >AA, >55 pattern to the entire RAM and then checking the RAM for this pattern. The inverted pattern is then written to RAM and rechecked. Finally, the entire RAM is cleared. If an error is found, a bit is set in a flag register.

		After	
Register	Before	No Error	After Error
A	XX	0	?
B	XX	0	?
Rn	XX	0	?
FLAG	XX	0	Bit 0 = 1

Passing data: None
Registers affected: All
Ending data: All registers = 0
Bit 0 in FLAG = 1 if error was found

```
*******************************************************
       MOV    %>55,A        Start RAM fill with >55
FILLR  MOV    %>FD,B        Set RAM start address - 2
*                           (don't change register A or B)
FILL1  STA    @2(B)         Fill RAM with AA 55 pattern
       RR     A             Change from 55 to AA to 55
       DJNZ   B,FILL1       Fill the entire RAM with this
*                           pattern
       RR     A             Change to beginning number
```

```
        MOV   %>FD,B      Refresh index
*
COMPAR  CMPA  @2(B)       Check for errors
        JNE   ERROR       Exit if the values don't match
        RR    A           Change from 55 to AA to 55
        DJNZ  B,COMPAR    Check the entire RAM
*
        TSTA              Is Reg A now 55, AA or 00?
        JN    FILLR       =AA, change to opposite pattern
        JZ    EXIT        =00, finished now get out
FILL0   CLR   A           =55, clear the RAM now
        JMP   FILLR       Repeat the fill and check
*                         routine
ERROR   OR    %1,FLAG     Set bit 0 in the flag
*                         register
EXIT    EQU   $           Continue program here
```

9.11.3 ROM Checksum

This routine checks the integrity of the ROM by performing a checksum on the entire ROM. All ROM bytes from >F008 to >FFFF are added together in a 16-bit word. This sum is checked against the value at the beginning of the ROM (>F006,>F007). If the values don't match, then an error has occurred and a bit is set in a register.

Register	Before	After No Error	After Error
A	XX	??	??
B	XX	??	??
R2	XX	CHKSUM MSB	CHKSUM MSB
R3	XX	CHKSUM LSB	CHKSUM LSB
R4	XX	>F0	>F0
R5	XX	>07	>07
R6	XX	>FF	>FF
R7	XX	>FF	>FF
FLAG	XX	Bit 1 = 0	Bit 1 = 1

```
*************************************************************
        AORG  >F006
        DATA  CHECKSUM    Put correct checksum into ROM
*       ...               Other initialization program
*                         here
ROMCHK  MOVD  %>FFFF,R5   Starting address (end of memory)
        MOVD  %>FF7,R7    Number of bytes to add + 1
        MOVD  %>0,R3      Reset summing register
*
ADDLOP  LDA   *R5         Get ROM byte
        ADD   A,R3        Add to 16-bit sum
        ADC   %0,R2
        DECD  R5          Point to next address
        DECD  R7          Decrement byte counter
        JC    ADDLOP      Continue until byte count goes
*                         past 0
        LDA   @>F007      Compare LSB stored to LSB sum
        CMP   A,R3
        JNE   ERROR       Set error bit if different
        LDA   @>F006      Compare MSB stored to MSB sum
```

```
        CMP   A,R2
        JEQ   EXIT          Set error bit if different
ERROR   OR    %2,FLAG       Set bit 1 in the Flag register
EXIT    EQU   $             Continue program here
*
```

9.11.4 Binary-to-BCD Conversion

This program converts a 16-bit binary word to a packed 6-nibble value.

Register	Before	After
A	XXXX	BCD MSB
B	XXXX	BCD
R2	XXXX	BCD LSB
R3	BINARY MSB	ZERO
R4	BINARY LSB	ZERO
R5	XXXX	ZERO

```
        AORG  >F006
*
BN2BCD  CLR   A             Prepare answer registers
        CLR   B
        CLR   R2
        MOV   %16,R5        Move loop count to register
LOOP    RLC   R4            Shift higher binary bit out
        RLC   R3            Carry contains higher bit
        DAC   R2,R2
        DAC   B,B           Double the number then add the
*                           binary bit
        DAC   A,A           Binary bit (a 1 in carry on 1st
*                           time is doubled 16 times).
        DJNZ  R5,LOOP       Do this 16 times, once for each
*                           bit
        RETS
```

9.11.5 BCD-to-Binary Conversion

Register	Before	After
A	BCD	Binary MSB
B	X	Binary LSB
R2	X	X

```
        AORG  >F006
*
BCD2BN  MOV   A,R2          Store word in R2
        AND   %>F0,A        Isolate MSB
        SWAP  A             Move to LSB position
        CMP   %10,A         Is it a valid BCD digit?
        JHS   ERROR         Goto error routine if not
        MPY   %10,A         Multiply MSB by 10, results
*                           at A,B in binary
        AND   %>0F,R2       Isolate LSB
        CMP   %10,A         Is it a valid BCD digit?
        JHS   ERROR
        ADD   R2,B          Add LSB to binary MSB to finish
*                           conversion
ERROR   RETS
        END
```

9.11.6 BCD String Addition

The following subroutine uses the addition instructions to add two multidigit numbers together. Each of the numbers is a packed BCD string of less than 256 bytes (512 digits) stored at memory locations STR1 and STR2. This routine adds the two strings together and places the result in STR2. The strings must be stored with the most significant byte in the lowest numbered register. The TMS7000 family instruction set favors storing all numbers and addresses with the most significant byte in the lower numbered location.

Register	Before	After	Function
A	XXXX	????	Accumulator
B	XXXX	0	Length of string
R2	XXXX	????	Temporary save register
STR1	XXXX	no change	BCD string
STR2	XXXX	STR1+STR2	Target string, 6 bytes max

```
*     Decimal Addition Subroutine
*     Stack must have 3 available bytes.
*     On output: STR2 = STR1 + STR2
*
*
ADDBCD  CLRC                  Clear carry bit
        PUSH   ST             Save status of stack
LOOP    LDA    @STR1-1(B)     Load current byte
        MOV    A,R2           Save it in R2
        LDA    @STR2-1(B)     Load next byte of STR2
        POP    ST             Restore carry from last add
        DAC    R2,A           Add decimal bytes
        PUSH   ST             Save the carry from this
*                             add
        STA    @STR2-1(B)     Store result
        DJNZ   B,LOOP         Loop until done
        POP    ST             Restore stack to starting
*                             position
        RETS                  Back to calling routine
*
```

Notice the use of the indexed addressing mode to reference the bytes of the decimal strings. Notice also the need to push the status register between decimal additions, to save the decimal carry bit. Register B is used to keep count of the number of bytes that have been added.

9.11.7 Fast Parity

This routine presents a quick way to determine the parity of a byte. By exclusive ORing all the bits of the byte together, a single bit will be derived which is the even parity of the word. When exclusive ORing, an even number of 1s will combine to form a 0, leaving either an odd 1 or 0 bit. This routine keeps splitting the byte in half and exclusive ORing the two halves.

Register	Before	After	Function
A	Target	????	Passing byte from program
B	XXXX	????	Length of string
Carry	XXXX	Parity	Status bit, result to calling routine

```
*********************************************************
* STEP 1                                        SUBROUTINE
* Byte bits   7654 3210                         TO FIND
*          XOR     7654 [MSN above]             EVEN PARITY
*            ==========
*             xxxx ABCD
* STEP 2                 ------>  AB CD
*                               XOR   AB [MS bits above]
*                                  =====
*                                  xx ab
* STEP 3                                ---->  a b
*                                          XOR  a   [MS bit]
*                                             ====
*                                             x P   {answer }
*
*********************************************************
*
PARITY MOV   A,B    Duplicate the target byte
       SWAP  A      Line up the MS nibble with the LS nibble
       XOR   B,A    Exclusive OR the nibbles to get a nibble
*                   answer
       MOV   A,B    Duplicate the nibble answer
       RR    A      Line up bits 0, 1 of the answer to bits
       RR    A      2, 3 of the answer
       XOR   B,A    XOR to get a new 2-bit answer
       MOV   A,B    Duplicate this 2-bit answer
       RR    A      Line up bit 0 with bit 1
       XOR   B,A    XOR to get final even parity answer
       RR    A      Rotate answer into the carry bit and bit 7
       RETS         Carry = 0 = even # of 1s
*                   Carry = 1 = odd # of 1s
*                   Use JC, JN or JNC JPZ in next executed
*                   instruction
```

9.11.8 Overflow and Underflow

An exclusive OR of the C and N bits ANDed with the exclusive OR of the MSbs of the operands can be used as a check for an overflow or underflow for **subtraction** in a signed system (if (C XOR N) AND (MSb1 XOR MSb2) = 1 then out of range).

When **adding** two signed numbers, the test for an out-of-range condition is similar to the subtraction method. When an exclusive OR of the C and N bits ANDed with the inverse of the exclusive OR of the MSbs of the two operands equals one then an overflow or underflow has occurred (if (C XOR N) AND (NOT(MSb1 XOR MSb2)) = 1 then out of range).

Register	Before	After	Function
A	XXXX	????	
OPRND1	XXXX	OPRND1	
OPRND2	XXXX	OPRD2–OPRD1	Subtraction results

```
* Routine to check for signed underflow or overflow
* If (C XOR N) AND (MSb1 XOR MSb2) = 1 then out of range
*
        MOV   OPRND1,A
        XOR   OPRND2,A          Get XOR of the MSbs
        SUB   OPRND1,OPRND2     Subtract 2 signed numbers
        JN    ISNEG
NOTNEG  JNC   NOERR             N = 0
        JMP   CXORN1            C XOR N = 1, First part of
*                               equation is true
ISNEG   JC    NOERR             N=1
CXORN1  TSTA                    C XOR N = 1; set flags for
*                              MSb1 XOR MSb2
        JPZ   NOERR               If (N XOR C) AND (MSb1 XOR
*                                 MSB2) = 1 then out of range.
*                                 For addition change this
*                                 instruction to JN NOERR
OUTRNG  ...                       Out of range; underflow or
*                                 overflow
*
NOERR   ...                     No underflow or overflow
```

9.11.9 Bubble Sort

This routine will sort up to 256 bytes using the bubble sort method. Longer tables could be sorted using the indirect addressing mode.

Register	Function
A	Temporary storage register
B	Index into the table
R2	Holds flag to indicate a byte swap has been made

```
        AORG  >F006
*
FLAG    EQU   R2              'Swap has been made' flag
*
SORT    CLR   FLAG            Reset swap flag
        MOV   %149,B          150 bytes to be sorted
LOOP1   LDA   @TABLE(B)       Look at entry in table
        CMPA  @TABLE-1(B)     Look at next lower byte
        JL    LOOP2           If lower skip to next value
        INC   FLAG            Entry is not lower, set swap
*                             flag
        PUSH  A               Store upper byte
        LDA   @TABLE-1(B)     Take lower byte
        STA   @TABLE(B)       Put where upper was
        POP   A               Get the old upper byte
        STA   @TABLE-1(B)     Put where the lower byte was
LOOP2   DJNZ  B,LOOP1         Loop until all the table is
*                             looked at
        BTJO  %>FF,FLAG,SORT
* If swap was made, then resweep table
* If no swap was made, then table is done
```

9.11.10 Table Search

Table searches are efficiently performed by using the CMPA (compare register A extended) instruction. In the following example, a 150 byte table is searched for a match with a 6-byte string:

Register	Before	After	Function
A	XXXX	????	
B	XXXX	????	
R2	XXXX	????	Table length
TABLE	XXXX	no change	Long string in table
STRING	XXXX	no change	Target string, 6 bytes max

```
*
SEARCH    MOV   %150+1,R2     Table length = 150 bytes
LOOP1     MOV   %6,B          String length = 6 bytes
LOOP2     XCHB  R2            Swap pointers, long string
*                             in B
          DEC   B             Table end? If so, no match
*                             found
          JZ    NOFIND
          LDA   @TABLE-1(B)   Load test character
          XCHB  R2            Swap pointers, string
*                             pointer in B
          CMPA  @STRING-1(B)  Match?
          JNE   LOOP1         If not, reset string
*                             pointer else test
          DJNZ  B,LOOP2       next character
MATCH     EQU   $             Match found
*
NOFIND    EQU   $             No match found
```

The indexed addressing mode is used in this example and has the capability to search a 256-byte string, if needed. Register B alternates between a pointer into the 6-byte test string and a pointer into the longer table string.

9.11.11 16-Bit Address Stack Operations

This routine performs 16-bit stack operations using the 1-byte TRAP instruction for pushing and popping. It uses macros to make code more readable. All values pass through register A.

Register	Function
A	Passing register for routines
R2	Indirect pointer MSB
R3	Indirect pointer LSB

```
*  Define Macro PUSH16 as a trap instruction
*
PUSH16    $MACRO
          TRAP   6
          $END
*
*  Define Trap 7 to be the POP16 operation
*
POP16     $MACRO
          TRAP   7
          $END
*
TRAP6     INC    R3          PUSH16
          ADC    %0,R2       Increment the indirect pointer
          STA    *R3         Push Register A
          RETS
```

```
*
TRAP7    LDA    *R3          POP16   Pop into Register A
         DECD   R3           decrement the indirect pointer
         RETS
*
*
         AORG   >FFF0        Set up Trap and Interrupt
*                            vectors
         DATA   TRAP7,TRAP6,INT5,INT4,INT3,INT2,INT1,RESET
         END
*
* Examples of use
*
         MOVD   %>1234,R3    Initialize the 16-bit stack
*                            pointer
         MOV    %DATA,A      Load Register A
         PUSH16              Use the macro to push A onto
*                            the stack
         POP16               Return a value from the stack.
         MOV    A,TEMP       Move the value to a temporary
*                            register
```

9.11.12 16-by-16 (32-Bit) Multiplication

This example multiplies the 16-bit value in register pair R2,R3 by the value in register pair R4,R5. The results are stored in R6, R7, R8, R9, and registers A and B are altered.

```
*
*
*        16-BIT MPY:              XH    XL       X VALUE
*                           X     YH    YL       Y VALUE
*                           ---------------
*                                 XLYLm XLYLl    l = LSB
*                           XHYLm XHYLl          m = MSB
*                           XLYHm XLYHl
*                  +  XHYHm XHYHl
*                  -------------------------
*                     RSLT3 RSLT2 RSLT1 RSLT0
*
XH       EQU    R2          Higher operand of X
XL       EQU    R3          Lower  operand of X
YH       EQU    R4          Higher operand of Y
YL       EQU    R5          Lower  operand of Y
RSLT3    EQU    R6          Msb of the final result
RSLT2    EQU    R7
RSLT1    EQU    R8
RSLT0    EQU    R9          LSB of the final result
*
MPY32    CLR    RSLT2       Clear the present value
         CLR    RSLT3
         MPY    XL,YL       Multiply LSBs
         MOV    B,RSLT0     Store LSB in result register 0
         MOV    A,RSLT1     Store MSB in result register 1
         MPY    XH,YL       Get XHYL
         ADD    R1,RSLT1    Add to existing result  XLYL
         ADC    R0,RSLT2    Add carry if present
         MPY    XL,YH       Multiply to get  XLYH
         ADD    R1,RSLT1    Add to existing result XLYL+XHYL
         ADC    R0,RSLT2    Add to existing results and carry
```

```
ADC     %0,RSLT3    Add if carry present
MPY     XH,YH       Multiply MSBs
ADD     R1,RSLT2    Add once again to the result reg
ADC     R0,RSLT3    Do the final add to the result reg
```

9.11.13 Binary Division, Example 1

This program divides a 16-bit dividend by an 8-bit divisor giving a 8-bit quotient and an 8-bit remainder. All numbers are unsigned positive numbers. The dividend's MSB must be less than the divisor to ensure an 8-bit quotient.

Dividend: 0–FFFF
Divisor: 1–255
Quotient: 0–255

Register	Before	After
A	DIVIDEND MSB	REMAINDER
B	DIVIDEND LSB	QUOTIENT
R2	DIVISOR	DIVISOR
R3	XXXX	ZERO

```
        AORG  >F006
*
BINDVD  MOV   %8,R3     Set loop counter to 8
DVDLP   RLC   B         Multiply dividend by 2
        RLC   A
        JNC   SKIP1     * These * steps are not needed
        SUB   R2,A      * if the divisor is limited
        SETC            * to 7 bits
        JMP   DIVEND    *
SKIP1   CMP   R2,A      Is MSB of dividend > divisor
        JNC   DIVEND
SUBIT   SUB   R2,A      If so dividend=dividend
*                       - divisor
*                       C=1 gets folded into next
*                       rotate
DIVEND  DJNZ  R3,DVDLP  Next bit, is the divide done.
        RLC   B         Finish the last rotate
```

9.11.14 Binary Division, Example 2

This program divides a 16-bit dividend by an 8-bit divisor, producing a 16-bit quotient and an 8-bit remainder. All numbers are unsigned positive numbers. The dividend's MSB can be larger than divisor.

Dividend: 0–FFFF
Divisor: 0–255
Quotient: 0–FFFF

$$8\overline{)16}\ \text{r}8$$

Register	Before	After
A	XXXX	REMAINDER
B	DIVISOR	DIVISOR
R2	DIVIDEND MSB	QUOTIENT MSB
R3	DIVIDEND LSB	QUOTIENT LSB
R4	XXXX	ZERO

```
        AORG  >F006
*
BINDVD  MOV   %16,R4    Set loop counter to 16 (8+8)
        CLR   A         Initialize result register
DVDLP   RLC   R3        Multiply dividend by 2
        RLC   R2
        RLC   A
        JNC   SKIP1     * These * steps are not needed
        SUB   B,A       * if the divisor is limited
        SETC            * to 7 bits
        JMP   DIVEND    *
SKIP1   CMP   B,A       Is MSB of dividend > divisor
        JNC   DIVEND
        SUB   B,A       If so dividend=dividend
*                       - divisor
*                       C=1 gets folded into next
*                       rotate
DIVEND  DJNZ  R4,DVDLP  Next bit, is the divide done?
        RLC   R3        Finish the last rotate
        RLC   R2
```

9.11.15 Binary Division, Example 3

This program divides a 16-bit dividend by an 16-bit divisor, producing a 16-bit quotient and a 16-bit remainder. All numbers are unsigned positive numbers. The dividend's MSB can be larger than divisor.

Dividend: 0–FFFF
Divisor: 0–FFFF
Quotient: 0–FFFF

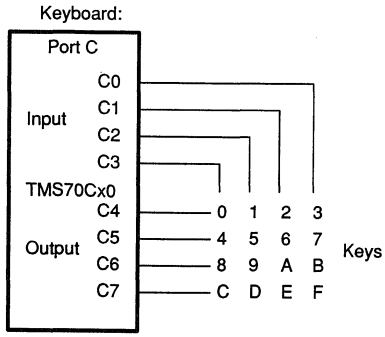
```
   16 r16
16)16
```

Register	Before	After
A	XXXX	REMAINDER MSB
B	XXXX	REMAINDER LSB
R2	DIVIDEND MSB	QUOTIENT MSB
R3	DIVIDEND LSB	QUOTIENT LSB
R4	DIVISOR MSB	DIVISOR MSB
R5	DIVISOR LSB	DIVISOR LSB
R6	XXXX	ZERO

```
        AORG  >F006
*
BINDVD  MOV   %16,R6    Set loop counter to 16
*                       (8 + 8)
        CLR   A         Initialize result register
        CLR   B
DIVLOP  RLC   R3        Multiply dividend by 2
        RLC   R2
        RLC   B
        RLC   A
        JNC   SKIP1     Check for possible error
        SUB   R5,B      condition that results
        SBB   R4,A      when a 1 is shifed past
*                       the most significant bit
        SETC            Correct by subtracting out
*                       the divisor
        JMP   DIVEND
*
SKIP1   CMP   R4,A      Is MSB+LSB of dividend >
*                       divisor
        JNC   DIVEND
        JNE   MSBNE     Are MSBs equal?
        CMP   R5,B      If so, compare LSBs
        JNC   DIVEND
*
MSBNE   SUB   R5,B      If borrow, dividend=divi-
*                       dend - divisor
        SBB   R4,A      C=1 get folded into next
*                       rotate
*                       Next bit, is the divide
*                       done?
DIVEND  DJNZ  R6,DIVLOP
        RLC   R3        Finish the last rotate
        RLC   R2
        RETS
```

9.11.16 Keyboard Scan

This routine reads a 16-key keyboard, returns the hex digit of the key, and debounces the key to avoid noise. A valid key flag is set when a new key is found.

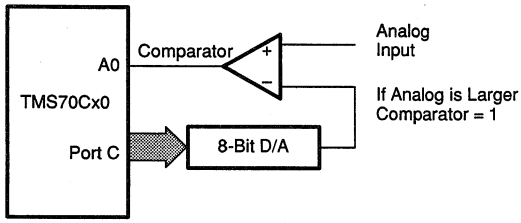


Register	Before	After No Key	After New Key	Function
A	XXXX	0	COLUMN	Temporary
B	XXXX	0	ROW	Temporary
R2	XXXX	16	KEY #	Temp store for Key value
R3	OLD KEY	>FF	KEY #	Holds Key pressed now
R4	DEBOUNCE	0	0	Debounce counter, old key or new
R5	GENERAL BITS	?xxxxxxx0	?xxxxxxx1	One bit of register is 1 if new key

```
        AORG   >F006
*
CDDR    EQU    P9
PORTC   EQU    P8
*
GETKEY  MOV    %8,B          Initialize row pointer
        CLR    R2
        MOVP   %>F0,CDDR     Set Data direction register 4 output,
*                            4 input
LOOP    RLC    B             Select next row
        JC     NOKEY         Last row ? if so no key was found
        ADD    %4,R2         Add number of keys/row to key
*                            accumulator
        MOVP   B,PORTC       Activate row
        MOVP   PORTC,A       Read columns
        MOVP   %0,PORTC      Clear row
        AND    %>F,A         Isolate column data
        JZ     LOOP          If no keys found then check next row
KEYLSB  DEC    R2            Decrement column offset
        RRC    A             Find  column
        JNC    KEYLSB        If not column then, try again
*
NEWKEY  CMP    R2,R3         Is the new key the same as the old key
        JEQ    DEBONS        If it is then debounce it
        MOV    R2,R3         Brand new key, Move it to current key
*                            value
        MOV    %16,R4        Set up debounce count


DEBONS  CMP    %2,R4         Is the debounce count 1 or 0 ?
        JL     GOODKY
        DJNZ   R4,GETKEY     If greater than 1 then debounce is
*                            not finished, go read key again
*
GOODKY  BTJZ #01,R4,NOTNEW   If debounce count=0 then key was here
*                            last time
        DEC    R4            If it was one this is a new valid key,
*                                make old key
        OR     %1,R5         Set new key flag in BIT register, the
NOTNEW  RETS                 calling routine uses this flag
*
NOKEY   MOV    %>FF,R3       No key was found, set key value to
*                            unique value
        RETS
```

9.11.17 8-Bit Analog-to-Digital Converter

This routine converts an analog signal to a digital value using a digital-to-analog converter and a comparator.

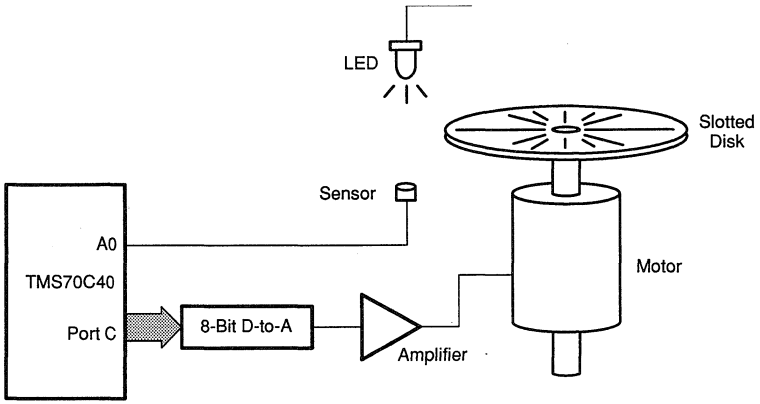


Register	Before	After	Function
A	XXXX	ANALOG VALUE	Final digital value
B	XXXX	ZERO	Trial and error test value

```
ATOD     MOV   %>80,B          Starting value for binary search
         CLR   A               Initialize  value
         MOVP  %>FF,P9         Port C is all outputs
*
LOOP     OR    B,A             Set the next bit in Test value
         MOVP  A,P8            Send it to the D-A converter
         BTJOP %1,P4,ABIGER    Is this value Less than the analog
*                              value?
ASMALL   XOR   B,A             If Analog value is smaller, decrease
*                              test value
ABIGER   RRC   B               If Bigger go to next bit in test value
         JNC   LOOP            If not at the end, then go test the
*                              next bit
*
FINISH   RETS
```

9.11.18 Motor Speed Controller

This routine keeps the speed of a motor constant. A pulse proportional to the speed of the motor comes from a sensor next to a slotted disk on the motor. The motor is controlled by a variable voltage generated by a D-A converter. Some mechanical considerations are necessary for an actual system.



Register	Before	After	Function
A	DATA	NO CHANGE	Temporary register
PULSE1	PULSE MSB	0	Holds MSB of pulse length value
SPEED	SPEED	NEW SPEED	Holds current voltage value for D–A
STEP	STEP SIZE	NEW SIZE	How much the voltage is changed per cycle
SPEED1	SPEED MSB	NO CHANGE	The desired time between the slots as measured by the
SPEED2	SPEED LSB	NO CHANGE	timer (1=MSB, 2=LSB)

```
        AORG   >F006
PULSE1  EQU    R4              MSB of 16-bit pulse length counter
SPEED   EQU    R5              Current voltage output to motor
STEP    EQU    R6              Change output voltage by this amount
SPEED1  EQU    R7              MSB of 16-bit speed reference
SPEED2  EQU    R8              LSB of 16-bit speed reference
BITS    EQU    R9              General purpose register for bits
INCR1   EQU    2               Step size for coarse adjustment
INCR2   EQU    4               Step size for fine adjustment
MCNTL   MOVP   %>1D,P12        Initialize the timer value
        MOVP   %>4C,P13        ""        ""
        MOVP   %>80+20,P15     Initialize the prescaler and start
*                              timer
        MOVP   %>3E,P0         Clear interrupts, enable I2, I3
        EINT                   The interrupts are now enabled
*
*       Main program body here
*
INT2    BTJZP  %>20,P0,OK      Interrupt 2 routine, check for pending
*                              INT3
        BTJOP  %>80,P15,OK     Check Capture Latch value for recent
*                              change
        JMP    INT3            If P3 is pending and CL just under-
*                              flowed then INT3 came first,
*                              go directly to INT3
OK      INC    PULSE1          Increment the MSB counter for the
*                              pulse length
        JNC    NOERR           If overflow there was an error
*                              (Motor too slow)
ERROR1  OR     %01,BITS        Set an error bit for the main
*                              routine to find
NOERR   RETI
*
INT3    MOVP   %>80+20,P15     Restart the timer at beginning
        PUSH   A               Save register
        MOV    %INCR1,STEP     Coarse adjustment step size for
*                              voltage change
        CMP    SPEED1,PULSE1
*                              Compare desired speed to measured
*                              speed (MSB)
        JEQ    TESTLS          If the same then compare LSBs
TESTSP  JL     GOSLOW          Does motor need to go faster or slower
GOFAST  ADD    STEP,SPEED      If faster, increase voltage to motor
OUTPUT  MOV    SPEED,A         Move new voltage value to D-A
        MOVP   A,P8
SAME    POP    A               Restore register
        CLR    PULSE1          Clear MSB of pulse length
        RETI   %
*
GOSLOW  SUB    STEP,SPEED      Decrease the motor voltage
        JMP    OUTPUT          Output voltage value
*
TESTLS  MOVP   P15,A           Get LSB of pulse length from capture
*                              Latch
        INV    A               Since it counts from FF to 00, invert
*                              value
```

```
        CMP     SPEED2,A        Compare desired speed to measured
*                               speed (LSB)
        JEQ     SAME            If the same do nothing
        MOV     %INCR2,STEP     Fine adjustment step size for voltage
*                               change
        JMP     TESTSP          Set new speed according to LSB values
```

Chapter 10

The TMS7000 8-Bit MCU Development Support

This chapter discusses key features of the hardware development tools.

Section		Page
10.1	The RTC/EVM7000 Evaluation Module	10-2
10.2	The Interactive Software for the EVM7000	10-4
10.3	The Extended Development Support	10-6
10.4	The Assembly Language	10-9
10.5	The Link Editor	10-11

10.1 The RTC/EVM7000 Evaluation Module

The RTC/EMV7000 is a TMS7000 Evaluation Module, referred to throughout this manual as the EVM. It is designed to emulate the single-chip mode of the TMS7000 CMOS families. It provides all the signals that would be available from masked ROM parts including the UART functions. The EVM provides the ability to develop, debug, and test programs prior to factory masking.

Note:

The EVM does not support the expansion modes of the TMS7000 family of processors. The EVM part number is: RTC/EVM7000C-1.

10.1.1 Functional Overview

The EVM is a single-board development system capable of emulating the single-chip of the TMS7000 family of microcomputers.

The EVM stands alone as a development system using its text editor for creation of TMS7000 assembly language text files with storage on cassette tape. The tape recorder has limited directory- and file-search capabilities. The EVM can also accept text files from a host CPU through either of the two EIA ports. In both situations, the resident assembler will convert the incoming text into executable code in the second pass after resolving labels from the first assembly pass.

The EVM firmware supports three ports in the operations of loading and dumping data (text and object code) for storage and/or display. Ports 1 and 2 conform to the EIA RS-232-C standard, and port 3 is the audio-tape connection. Details about connecting a device to these ports are contained in Chapter 2.

Port 1 User terminal/terminal emulator
Port 2 Uplink/downlink to/from host CPU, or line printer
Port 3 Audio tape

No UART is visible onboard; the EVM implements the UART function for both EIA ports in software and supports the following baud rates:
110, 150, 300, 600, 1200, 2400, 4800, 9600

The baud rate of port 1 (terminal) is determined automatically at powerup by striking the carriage return on the terminal (cabled to port 1) after hitting the RESET switch on the EVM. This feature is called autobaud and eliminates the need to select baud rates. The baud rate of port 2 defaults to 9600 baud at reset, and the baud rates of both ports can be changed with the monitor command BR.

The EVM firmware is contained in 24K bytes of EPROM. The unused portion of the U45 EPROM is accessed with the monitor commands U0 through U9.

The EVM requires 2K bytes of system RAM that is separate from the 32K bytes of user RAM. A wire-wrap development area, with all required signals provided and labeled, is available for additional logic.

Since the EVM is intended to be a development tool by using the emulation cable, the crystal frequency of the EVM can be altered to fit the needs of the target system.

10.1.2 Operating System

The EVM operating system firmware resides in 24K bytes of EPROM and can be divided into three major parts:

- Debug monitor and EPROM programmer
- Text editor
- Assembler

All the software is designed to interact with the user to provide a complete, powerful, and easy to use development tool. During assembly and debug operations, the EVM RAM can be configured to emulate all TMS7000 family members. For emulation of the TMS7000 devices, the EVM allows assembly of text files from RAM, leaving the text intact for immediate editing after execution. After assembly of the text editor output, breakpoints can be set based on either addresses or line numbers. During execution, several modes of fixed displays are available, providing a hex display of the entire register and peripheral files or a binary display of the peripheral ports. During a fixed display, subsequent execution to a breakpoint or execution of a single instruction step will overwrite the old data on the screen with new data. A programmable line of up to six register or peripheral locations is provided for display with breakpoints and instruction steps. The text editor is cursor- and line-number oriented with autoincrement-line numbers, resequence-line numbers, change-line number, duplicate line, and find string commands. The cursor-oriented edit capability simulates a screen editor by allowing editing of the previous or next line moving the cursor up or down.

10.2 The Interactive Software for the EVM7000

10.2.1 General Information

This menu driven software offers the advantage to exploit the features of the EVM7000 evaluation board with more efficiency. The goal is to optimize tremendously the man-machine interface, by offering a high level of communication. Therefore, the PC's resources are fully used: windows, hard disk, function keys, processor, etc. The use of the EVM7000 Board is supported by this software. The connection is realized by using a RS232 cable pin to pin wire. The whole handling of communication is done by this software. Also, the user can, at any time, go back to standard connection of the EVM7000 (terminal emulation).

Main features of this software are

- Windows used for general information
- Menu-driven instructions with macro instructions
- Call and editor commands
- Handling of instruction set
- Symbolic disassembler
- Symbolic access to all instructions
- Large choice of display (byte, words, long word, binary, hexa, decimal...)
- Windowing edition
- Step-by-step execution
- 10 breakpoints with floppy backup
- Terminal emulator mode for driving directly the EVM7000 Board
- Temporary return to DOS

As an illustration please find below a view of a PC screen in normal operation.

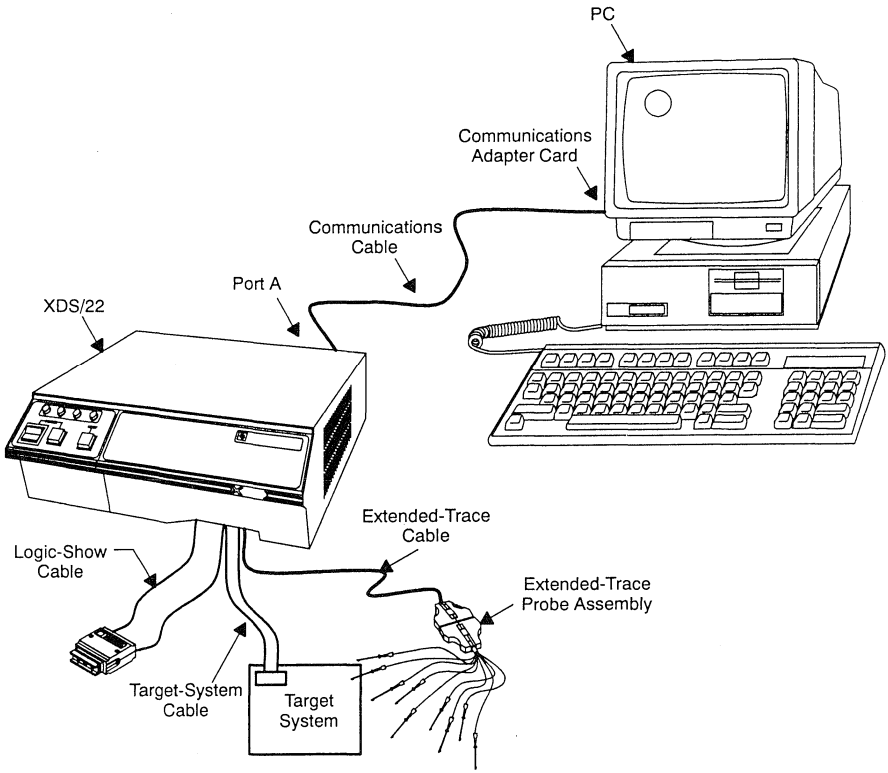
Display modify execute reset save load terminal quit command:

```
                    CODE                                  CPU REGISTERS
FC27  A3BF06        ANDP  %>BF, P006        PC FBF2        SP 05   ST CNZI
FC2A  A3EF06        ANDP  %>EF, P006        A  00          B  00   20 0010
IREC    8EF9A7      CALL  @IRECO            ======== R FILE==========  STACK=
LOOP    05          EINT                    R00  00H  R01  00H       SP (05) AA
FC31    321C        MOV   R01C,B            R02  FCH  R03  30H       - 1(04) F9
FC33    E2FB        JZ    LOOP              R04  F9H  R05  AAH       - 2(03) 30
FC35    8EF9C0      CALL  @RECO             R06  20H  R07  FFH       - 3(02) FC
FC38    C1          TSTB                    R08  00H  R09  FFH       - 4(01) 00
FC39    E2F5        JZ    LOOP              R0A  00H  R0B  00H       - 5(00) 00
FC3B    06          DINT                    R0C  00H  R0D  FFH       - 6(FF) 00
                                            R0E  80H  R0F  BFH       - 7(FE) 00
                                            R10  18H  R11  FFH       - 8(FD) 00
                                            R12  00H  R13  FFH       - 9(FC) 00
                  DISPLAY                     TMS7000 EVM INTERACTIVE DEBUGGER
                                              V 1.0
0000 00 00 FC 30 F9 AA 20 FF ...0.. .
0008 00 FF 00 00 00 FF 80 BF ........
0010 18 FF 00 FF 00 FF 00 2F ......./
0018 40 00 00 00 01 FF 60 DF @..... .
0020 00 DF 00 FF 40 FF 80 BF ....@...
0028 80 FF 00 FF 00 FF 04 3F .......?   0028
0038 00 F7 00 FF 00 FF 40 DF ......@.
0040 14 EF 00 FF 00 FF 10 EF ........
0048 00 BF 00 FF 00 FF 20 7F .......
```

10.3 The Extended Development Support

The Texas Instruments Extended Development Support (XDS) System, Model XDS/22 (Figure 10–1) is configured for the TMS7000 family members. It must be used in conjunction with the Communication Equipment Employing Serial Binary Data Interchange Standard (see *Related Information* in Preface).

Figure 10–1. Extended Development Support (XDS) System, Model XDS/22



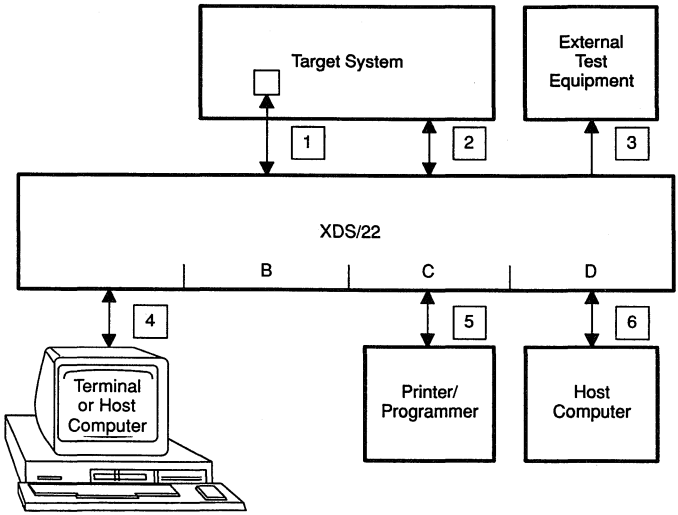
This XDS/22 system contains

- Communications card
- Breakpoint/Trace/Timing card
- TMS7000 Emulator card with firmware, revision level 2.0.0 or higher
- CMOS SE Emulator card

An XDS/22 executes emulator commands, device assembly language statements and directives, and machine language instructions to drive the hardware

in your design system (hereafter called the target system) just as if that target system contained a TMS7000 device. Figure 10–2 is a block diagram showing typical device emulation.

Figure 10–2. Emulation Block Diagram



The target cable (Figure 10–2, item 1) supplied with your XDS/22 has a connector that plugs into the exact system socket that will contain the TMS7000 device. You can enter device and emulation control commands or download them from a terminal keyboard through communications cables (items 4 or 6). Item 4 is provided with your XDS/22; item 6 is not. The entries and the target system or XDS responses can be seen on the terminal screen. The program can be written into an EPROM by connecting a PROM programmer to port C via item 5 or uploading it through the cables shown as items 4 and 6 to convert it into a masked ROM. However, if the application program doesn't work, the XDS/22 debugging and memory modification features are to be used. An XDS/22 supports a printer connected to port C for recording your input and XDS responses. The cable (item 5) may not be the same as for the EPROM programmer.

Included with your XDS/22 is an extended trace cable (item 2) and a logic analyzer (Texas Instruments Logic Show) cable (item 3). The extended trace feature is used to examine points in the target system circuitry that do not go directly to the emulated device. The Logic Show cable brings out buffered versions of the XDS address and data buses, as well as certain control signals for display on a logic analyzer or oscilloscope.

An XDS/22 can be used with any RS-232-C terminal as an excellent standalone development system, or with the XDS/22 and a personal computer as a standalone or a computer-assisted development system, or with the XDS/22 and a terminal or personal computer to tap the power and resources of a separate micro, mini, or mainframe host computer.

Up to nine XDS units can be connected together in either the standalone or computer-assisted mode to emulate multiprocessing designs.

10.3.1 XDS/22 Components

The XDS/22 contains three printed circuit cards:

- Communications card
- TMS7000 Emulator card
- Breakpoint/Trace/Timing card

10.4 The Assembly Language

10.4.1 General

Assembly language is a computer-oriented language for writing programs, consisting of mnemonic instructions and assembler directives. In assembly instructions, symbolic addresses are assigned to memory locations and specify instructions by means of symbolic operation codes called mnemonic operation codes. Instruction operands are specified by means of symbolic addresses, numbers, and expressions consisting of symbolic addresses and numbers.

Assembler directives control the process of converting an assembly language program into a machine language program, place data in the program, and assign symbols to values to be used in the program. Assembler directives that place data in memory locations allow the user to assign symbolic addresses to those locations.

Assembly language is computer-oriented in that the mnemonic operation corresponds directly with machine instructions. The chief advantage of an assembly language over machine language is that the mnemonic symbols are easier to use and easier to remember than the binary zeros and ones of machine language. Other advantages are the use of expressions as operands and the use of decimal numbers in expressions and as operands.

This manual describes the construction of assembly language programs for Texas Instrument's TMS 7000 family of 8-bit microcomputers. Topics covered include general programming information, discussion of addressing modes and instruction types, a definition of instructions, discussion of user application techniques, and descriptions of source- and cross-reference listings, object code, and normal and abnormal errors.

10.4.2 Assembly Language Application

An assembly language program (the source program) must be processed by an assembler to obtain a machine language program that can be executed by the computer. Changing a source program to object code is called assembling because the process converts the mnemonic instruction to binary values, then associates them with absolute or relocatable binary addresses to form a machine language instruction.

Steps in program development include

1) Define the problem.

2) Flowchart the solution to the problem.

3) Code the solution by writing assembly language statements (machine instructions and assembler directives) that correspond to the steps of the flowchart.

4) Prepare the source program by writing the statements on the medium appropriate to the installation; for example, enter a file on a disk, keypunch the statements, etc.

5) Execute the assembler to assemble the machine language object code corresponding to the source program.

6) Debug the resulting object code by loading and executing the object code and making corrections indicated by the results of executing the object code.

7) Repeat steps 5 and 6 until no further corrections are required.

The use of assembly language in program development relieves the programmer of the tedious task of writing machine language instructions and keeping track of binary machine addresses within the program.

10.5 The Link Editor

The link editor combines separately generated object modules with associated procedures and overlays to form a single, linked, relocatable object module that can be installed and executed on various computer systems. The object code is generated by assemblers supplied with the TMS7000 software development systems. The link editor is currently available for VAX (VMS and Berkeley UNIX 4.1 and 4.2), and TI/IBM PC (MS/PC-DOS) operating systems.

The link editor manual describes its files and control commands, and gives examples of various linking procedures. Included in this document are the following major topics:

- Introduction
  - Description
  - Program definition (phase and task)
- Link editor files
  - Link control file
  - Object modules
  - Libraries
  - Linked output file
  - Listing file
- Linker commands
  - Entering a command
  - Command set summary (listed according to function)
  - Individual command descriptions (alphabetized)
- Linking examples
  - Simple link
  - RAM/ROM partitioning
  - Partial link
  - Library creation
- Link editor error messages
- Glossary

Chapter 11

Customer Information

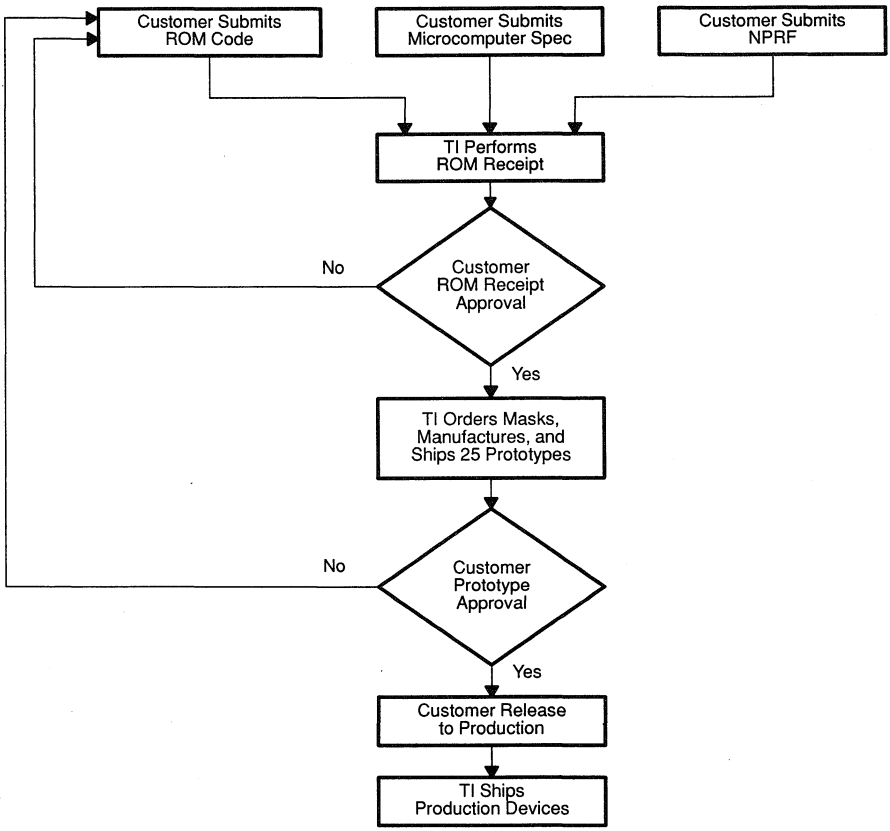
Topics covered in this chapter include:

Section		Page
11.1	Mask ROM Prototype and Production Flow	11-2
11.2	Mechanical Package Information	11-7
11.3	TMS7000 Family Numbering and Symbol Conventions	11-16
11.4	Development Support Tools Ordering Information	11-19

11.1 Mask ROM Prototype and Production Flow

The TMS7000 family of masked-ROM microcomputers are semi-custom devices. The ROM is tailored to the customer's application requirements. The semi-custom nature of these devices requires a standard, defined interface between the customer and the factory in the production of TMS7000 devices with on-chip ROM. Figure 11–1 shows this standard prototype/production flow for customer ROM receipt.

Figure 11–1. Prototype and Production Flow



1) Customer required information

For TI to accept the receipt of a customer ROM algorithm, each of the following three items must be received by the TI factory:

a) The customer completes and submits a New Product Release Form NPRF (available from TI Field Sales Office) describing the custom features of the device (for example, customer information, prototype and production quantities and dates, any exceptions to standard electrical specifications, customer part numbers and symbolization, package type, etc.). This form will bear the first of three signatures at this stage.

b) If non-standard specifications are requested on the NPRF, then the customer submits a copy of the specification for the microcomputer in their system, including the functional description and electrical specification (including absolute maximum ratings, recommended operating conditions, and timing values).

c) When the customer has completed code development and verified its operation with an emulation system and appropriate form factor emulation device such as a TMP77C82JDL, the code is submitted to Texas Instruments in the following form:

- Two EPROMs containing the final object code (TMS2764 or TMS27128 or CMOS equivalent)
- A floppy disc containing
  - The source code
  - The object code (TI tagged, Intel or Tek hex format)
- If a linker has been used to generate the final object file
  - All of the source code modules `.ASM` files
  - All of the object modules `.MPO` files
  - The final linked object `.LOD` file
  - The link control file `.CTL` file

The completed NPRF, customer specification (if required), and ROM code should be given to the Field Sales Office.

2) **TI performs ROM receipt**

Code review and ROM receipt is performed on the customer's code and a unique manufacturing ROM code number (such as C13827N) is assigned to the customer's algorithm. All future correspondence should indicate this number. The ROM receipt procedure reads the ROM code information, processes it, reproduces the customer's ROM object code on the same media on which it was received, and returns the processed and the original code to the customer for verification of correct ROM receipt.

3) **Customer ROM receipt approval**

The customer then verifies that the ROM code received and processed by TI is correct and that no information was misinterpreted in the transfer. The customer then returns a copy of the NPRF bearing a second signature. This written confirmation of verification constitutes the contractual agreement for creation of the custom mask and manufacture of ROM verification prototype units.

4) **TI orders masks, manufacturing, and ships 25 prototypes**

TI generates the prototype photomasks, processes, manufactures, and tests 25 microcomputer prototypes containing the customer's ROM pattern for shipment to the customer for *prototype* verification. These microcomputer devices have been made using the custom mask but are for the purposes of ROM verification only. **Texas Instruments recommends that prototype devices not be used in production systems.**

5) **Customer prototype approval**

The customer then returns a copy of the NPRF bearing a third signature. This written customer prototype approval constitutes the contractual agreement to initiate volume microcomputer production using the verified prototype ROM code.

6) **Customer release to production**

With customer algorithm approval, the ROM code is released to production and TI will begin shipment of production devices according to customer's final specification and order requirements.

Two lead times are quoted in reference to the preceding flow:

- Prototype lead time — elapsed time from the receipt of written ROM receipt verification to the delivery of 25 prototype devices.
- Production lead time — elapsed time from the receipt of written customer prototype approval to delivery of production devices.

For the latest TMS7000 family lead times, contact the nearest TI field sales office.

11.1.1 Reserved ROM Locations

All TMS7000 family devices with on-chip mask ROM reserve the first six bytes of the ROM space for TI use and therefore should not be used in the customer's software algorithm. For applications targeted for on-chip mask ROM production, the customer must remember to reserve this space during the development stage when using the XDS emulator, the EVM board, the TMP77C82JDL, piggyback emulators (SE70CP160 and SE70CP168), or a TMS7000 family member without on-chip ROM. Table 11–1 lists the valid ROM starting addresses for the mask-ROM devices.

Table 11–1. Valid ROM Start Addresses

Member	ROM Size	Start Address
TMS70C20, TMS70CT20	2K bytes	>F806
TMS70C42, TMS70C48, TMS70C40, TMS70CT40	4K bytes	>F006
TMS70C82	8K bytes	>E006

11.1.2 Manufacturing Mask Options

The TMS7000 family supports several mask options, depending on the base set. These options are selected at the time of mask manufacturing and therefore cannot be changed by software or hardware once the device has been manufactured. Selection of these mask options are designated by the customer New Product Release Form (NPRF) when ordering TMS7000 family members with on-chip mask ROM. TMS7000 family members without on-chip mask ROM have this designation as part of their standard part-number symbolization.

The oscillator input options RC, XTAL and, for the 70Cx8 devices, CER, define how the TMS7000 internal oscillator driver circuits operate. The internal schematics and the various option capabilities are described in the subsection 3.4.2. All three options can have a 50% duty cycle external clock as an external clock source. The XTAL input option allows the external clock source (a crystal or ceramic resonator) to oscillate continuously in all operational modes. The RC input option causes the clock source (an RC network only) to be effectively disables from driving the on-chip oscillator of the device when the halt-off mode is entered. This option significantly reduces the low power requirements for all devices. The CER input option is available for the 70Cx8 devices. It is especially recommended when using a ceramic resonator to obtain the oscillator stopped when the halt-off mode is entered; the device will be brought out of halt mode after a delay specified by the HALT/DELAY pin capacitor.

The 70Cx8 family devices offer other mask options:

- The multiplex option (MUX) allows the lines of port C to be multiplexed between 8 least significant bits of the address bus, and the system data bus. In the non-multiplex option (NMUX) the CPORT lines are the address bus 8 least significant bits, with DPORT lines being the MSB lines, and EPORT lines being the data bus.

- The chip select (CS) option affects the GPORT 4 most significant bits. In the IO option, these 4 bits are standard I/O lines. Thus the all G0 to G5 lines function as bit-programmable bidirectional I/O port. In the CS option, G2 to G5 pins are used as the four chip-select signal pins to produce a low active signal decoded most significant 4-bit address or peripheral file area (>0100–>01FF) by mask option using the on-chip fixed PLA (programmable logic array). For detailed informations on the PLA, you can contact your local TI office.

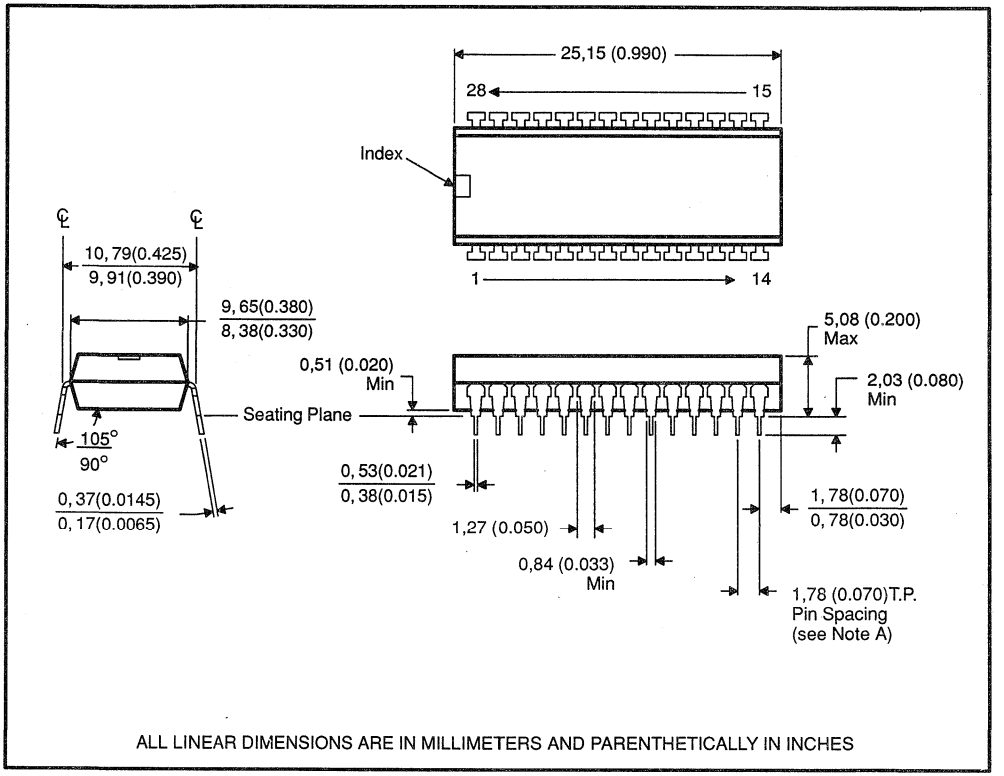
11.2 Mechanical Package Information

The TMS7000 microcomputer family devices are packaged in five package types according to the type of material and outline used for the package: plastic-dual-in-line (DIP), plastic-leaded-chip-carrier (PLCC), quad-flat-pack (QFP), ceramic-sidebraze package and ceramic-sidebraze-piggyback package. Package types are designated in the device symbolization by the suffix on the customer's ROM code number for device manufactured with on-chip ROM (for example, C55401N) and by the suffix of the standard device number for devices without on-chip ROM (for example, TMS70C00N). Table 11–2 indicates the package type, suffix indicator and family members supported on that package type.

Table 11–2. Package Types

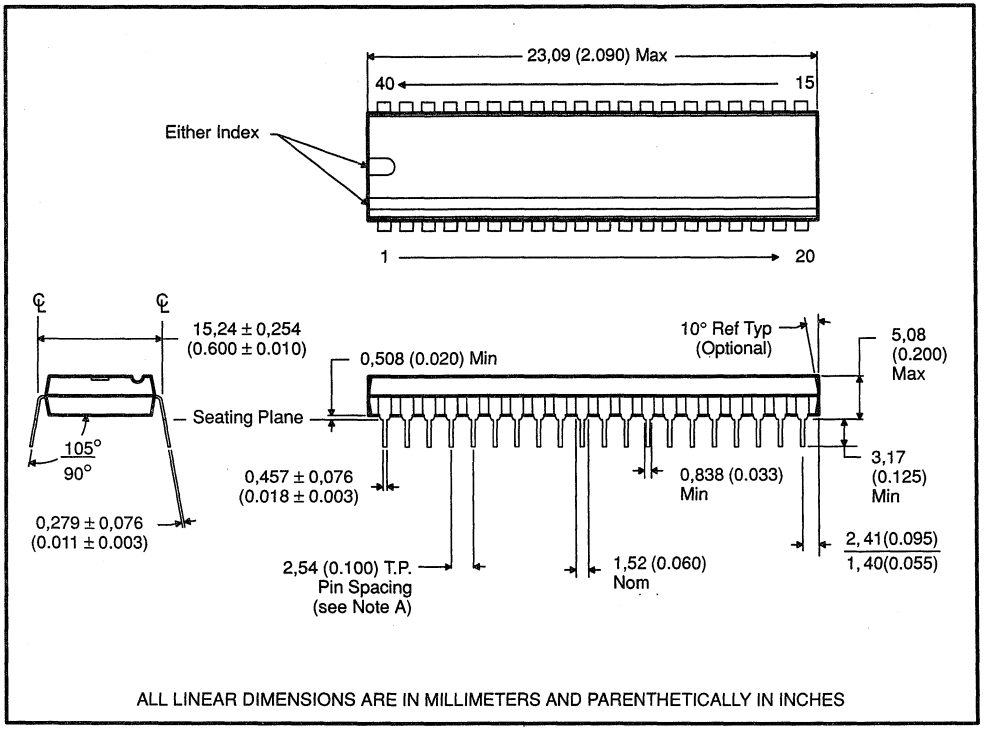
Package Type	Suffix	Family Members
28-pin DIP (70-mil pin spacing)	N2	TMS70CT20, TMS70CT40
40-pin plastic DIP (100-mil pin spacing)	N	TMS70C00, TMS70C20, TMS70C40, TMS70C02, TMS70C82, TMS77C82
40-pin plastic DIP (70-mil pin spacing)	N2	TMS70C00, TMS70C20, TMS70C40, TMS70C02, TMS70C82, TMS77C82
40-pin ceramic piggyback (100-mil pin spacing)	JD	SE70CP160
44-pin PLCC (50-mil pin spacing)	FN	TMS70C00, TMS70C20, TMS70C40, TMS70C02, TMS70C82, TMS77C82
68-pin PLCC (50-mil pin spacing)	FN	TMS70C08, TMS70C48
64-pin quad flat pack (100-mil pin spacing)	PG	TMS70C08, TMS70C48
64-pin ceramic piggyback (100-mil pin spacing)	JD	SE70CP168

Figure 11–2. 28-Pin Plastic Package, 70-MIL Pin Spacing (Type N2 Package Suffix)



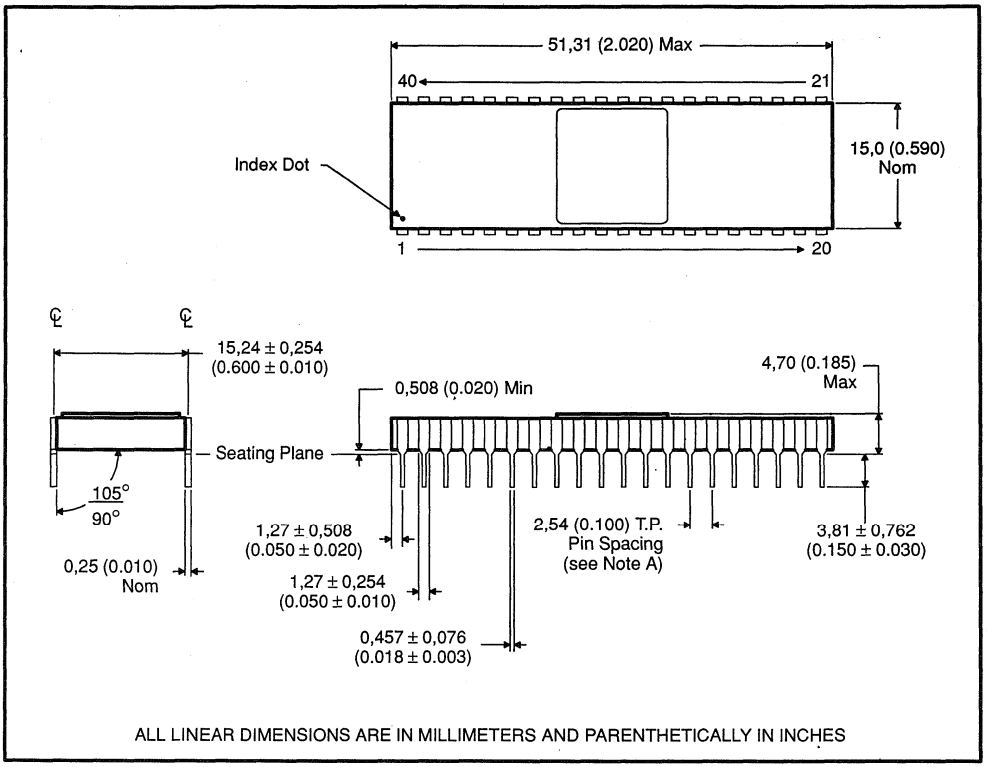
NOTES: A. Each pin centerline is located within 0,254 (0.010) of its true longitudinal position.
B. This dimension does not apply for solder-dipped leads.
C. When solder-dipped leads are specified, dipped area of the lead extends from the lead tip to at least 0,51 (0.020) above seating plane.

Figure 11–3. 40-Pin Plastic Package, 100-MIL Pin Spacing (Type N Package Suffix)



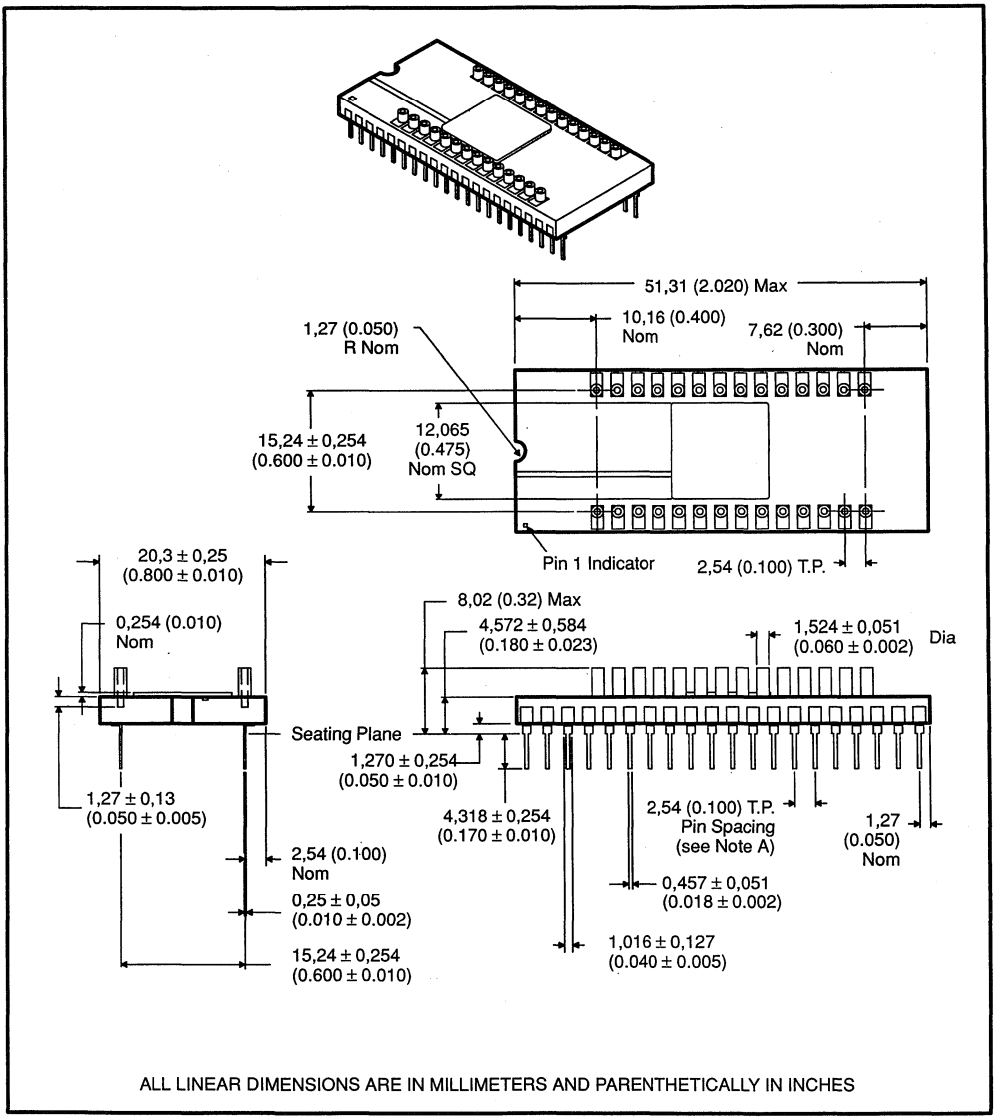
NOTE: A. Each pin centerline is located within 0,254 (0.010) of its true longitudinal position.

Figure 11–4. 40-Pin Ceramic Package, 100-MIL Pin Spacing (Type JD Package Suffix)



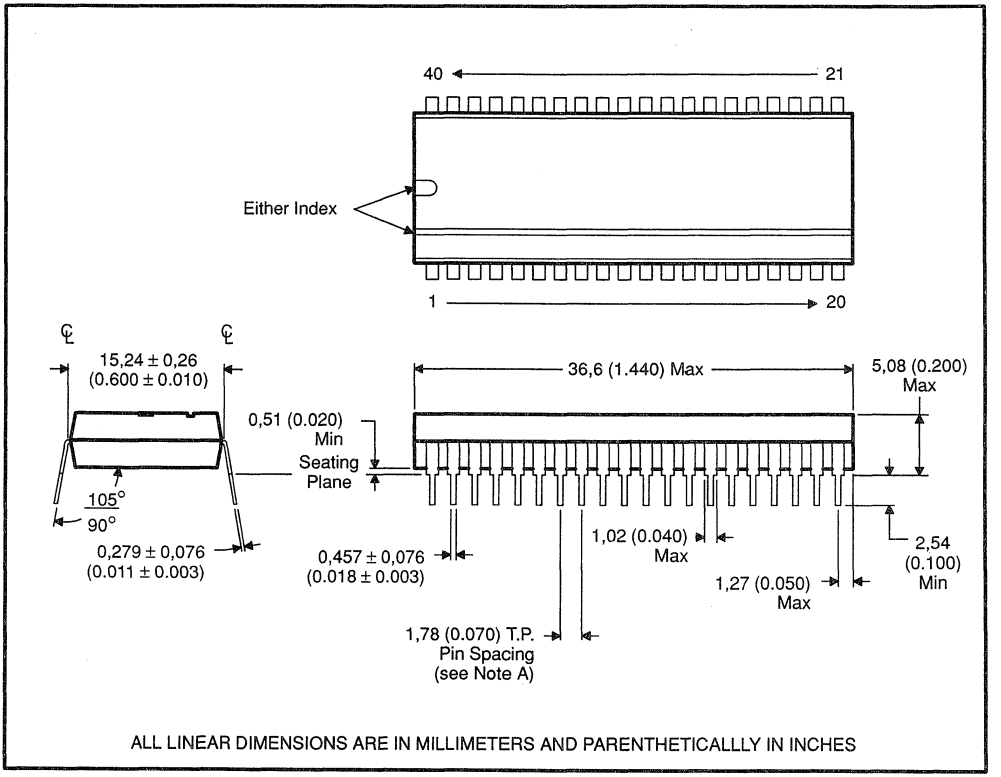
NOTE: A. Each pin centerline is located within 0,254 (0.010) of its true longitudinal position.

Figure 11–5. 40-Pin Ceramic Piggyback Package, 100-MIL Pin Spacing (Type JD Package Suffix)



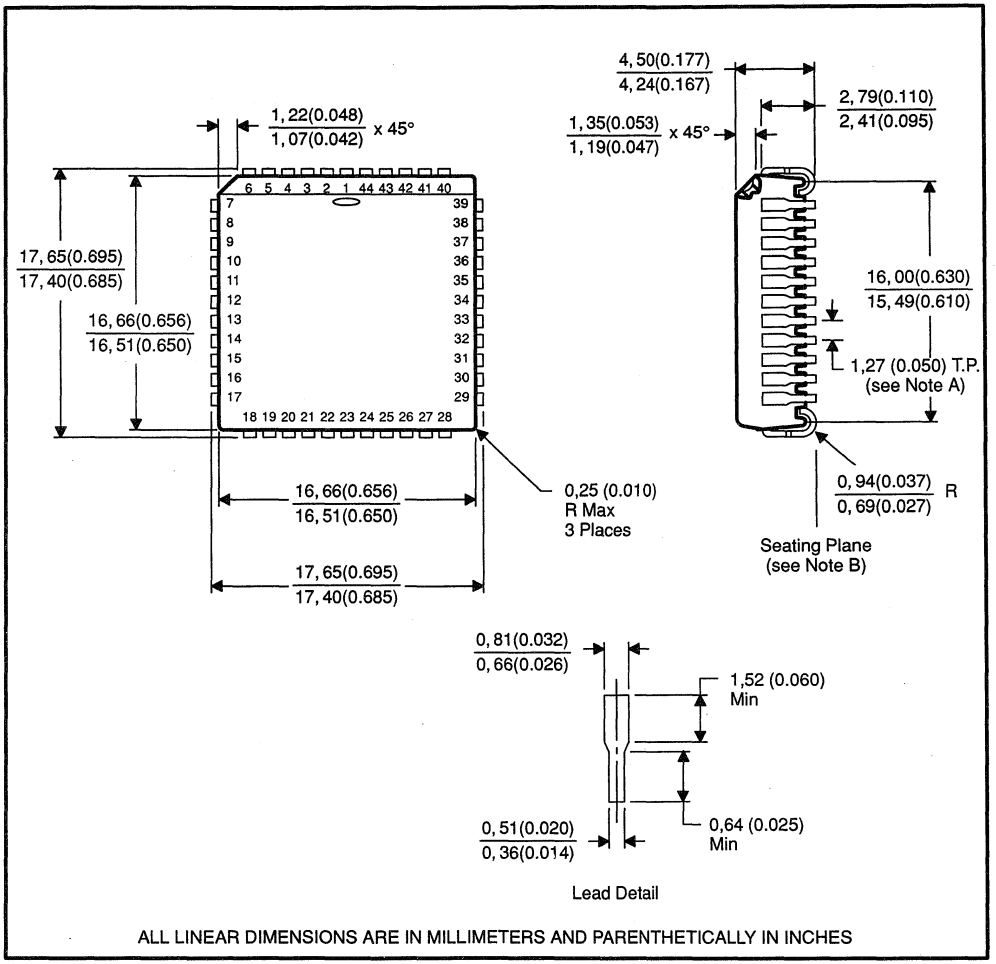
NOTE: A. Each pin centerline is located within 0,25 (0.010) of its true longitudinal position.

Figure 11–6. 40-Pin N2 Plastic Package, 0.070" Pin Center Spacing 0.600" Pin Row Spacing



NOTE: A. Each pin centerline is located within 0,26 (0.010) of its true longitudinal position.

Figure 11–7. 44-Pin Plastic-Leaded Chip Carrier FN Package



NOTES: A. Location of each pin is within 0,127 (0.005) of true position with respect to center pin on each side.
B. The lead contact points are planar within 0,10 (0.004).

Figure 11–8. 68-Pin Plastic-Leaded Chip Carrier FN Package

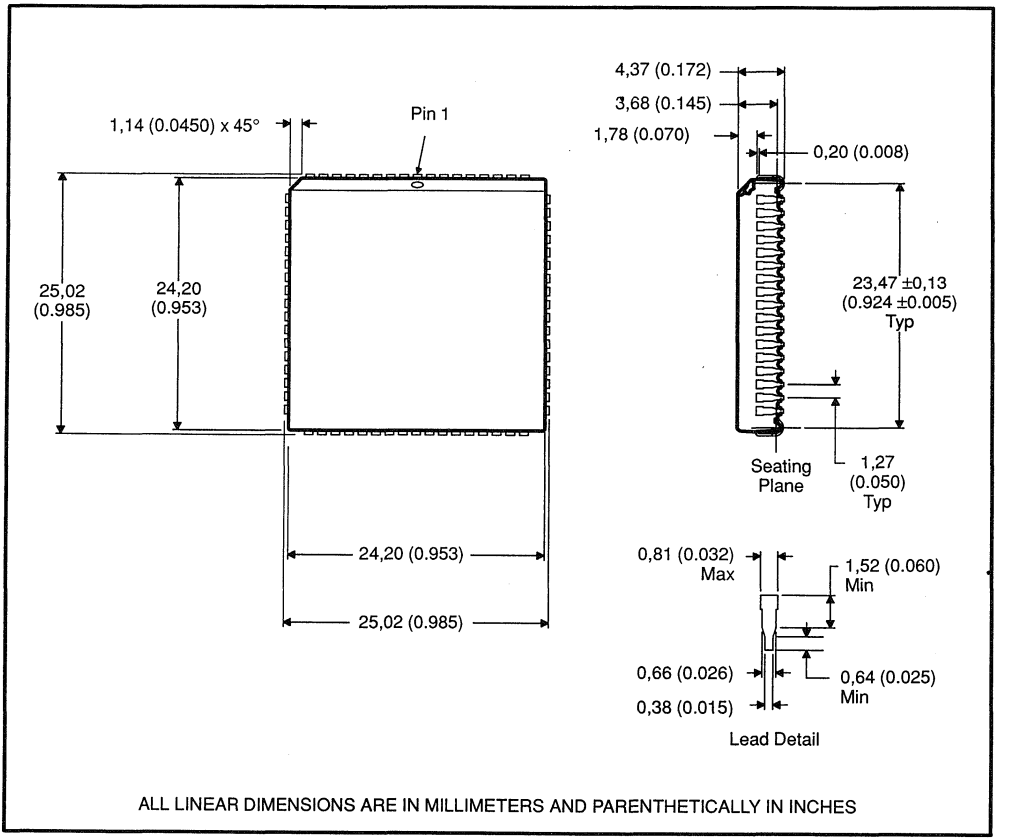
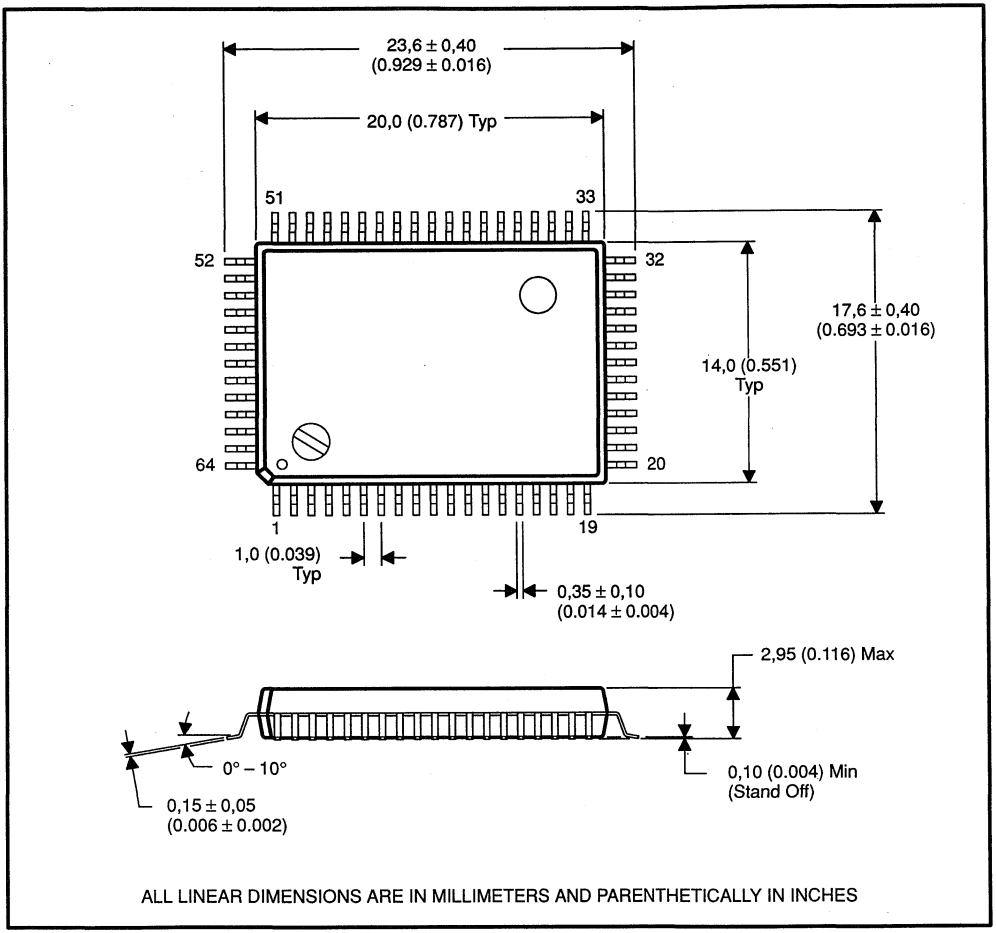


Figure 11–9. 64-Pin Flat Package PG Package



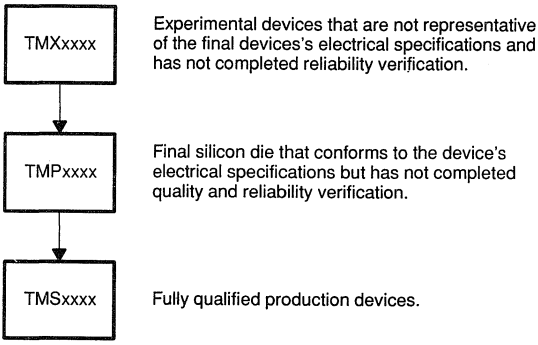
11.3 TMS7000 Family Numbering and Symbol Conventions

11.3.1 Device Prefix Designators

To provide expeditious system evaluations by customers during the product development cycle, Texas Instruments assigns a prefix designator with four options: TMS, TMP, TMX, and SE.

TMX, TMP, and TMS are representative of the evolutionary stages of product development from engineering prototypes through fully qualified production devices. Figure 11–10 depicts this evolutionary development flowchart. Production devices shipped by Texas Instruments have the TMS designator signifying that they have demonstrated the high standards of Texas Instruments quality and reliability.

Figure 11–10. Development Flowchart



TMX devices are shipped against the following disclaimer:

1) Experimental product and its reliability has not been characterized.
2) Product is sold "as is".
3) Product is not warranted to be exemplary of final production version if or when released by Texas Instruments.

TMP devices are shipped against the following disclaimer:

1) Customer understands that the product purchased hereunder has not been fully characterized and the expectation of reliability cannot be defined; therefore, Texas Instruments standard warranty refers only to the device's specifications.
2) No warranty of merchantability or fitness is expressed or implied.

TMS devices have been fully characterized and the quality and reliability of the device has been fully demonstrated. Texas Instruments' standard warranty applies.

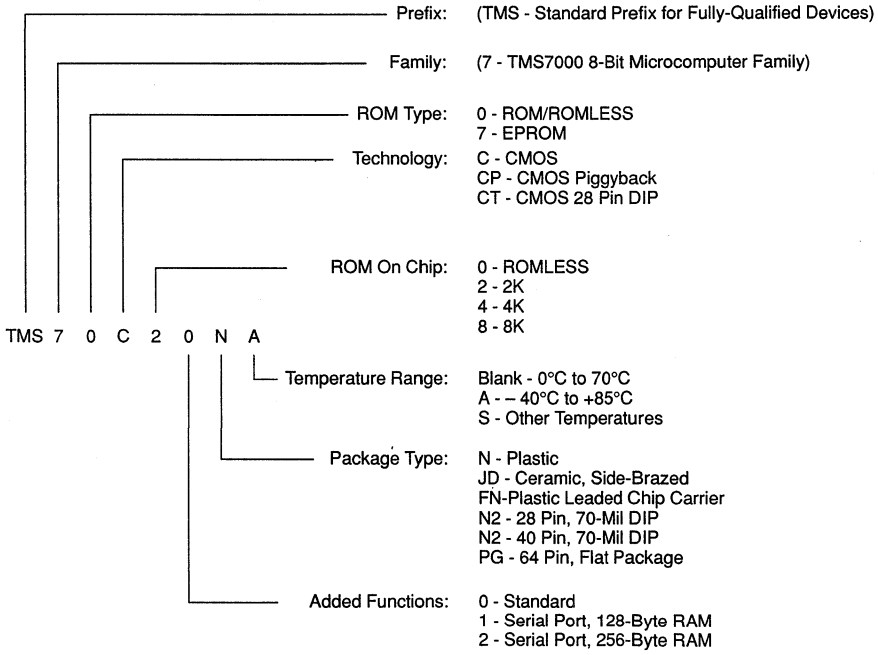
The SE prefix designation is given to the system evaluator devices used for prototyping purposes. This designation applies only to the prototype members of the TMS7000 family (the CMOS SE70CP160 and TMP77C82JDL devices). SE devices are shipped against the following disclaimer:

System evaluators and development tools are for use only in a prototype environment and their reliability has not been characterized.

11.3.2 Device Numbering Convention

Figure 11–11 illustrates the numbering and symbol nomenclature for the TMS7000 family.

Figure 11–11. TMS7000 Family Nomenclature



11.3.3 Device Symbols

The TMS7000 family members can be divided into two categories for description of symbols, with the distinction being made on the presence (or absence) of on-chip ROM.

11.3.3.1 TMS7000 Family Members with On-Chip ROM

TMS7000 family members with on-chip ROM are semicustom devices where the ROM is mask programmed according to the customer's requirements.

These devices follow the prototyping and production flow outlined in Section 11.3. Since they are semicustom devices, they receive a unique identification.

There are two types of symbolization for TMS7000 family members with on-chip ROM:

1) TI standard symbolization and
2) TI standard symbolization with customer part number.

Figure 11–12. TI Standard Symbolization

Line 1:	(a)	(b) C12345N	(c) DBUA8327	Key: (a) Texas Instruments Trademark (b) Customer's ROM code and package type
Line 2:	(d) ©1981TI	(f) ©1983TI		(c) Tracking mark and date code (d) TI microcode copyright
Line 3:	(e) 24655	(g) Philippines		(e) Lot code (f) Copyright of ROM code (g) Assembly Site

Figure 11–13. TI Standard Symbolization with Customer Part Number

Line 1:	(a)	(b) 123456789012		Key: (a) Texas Instruments Trademark (b) Customer Part Number
Line 2:		(c) C12345N	(d) DBUA8327	(c) Customer's ROM code and package type (d) Tracking mark and date code
Line 3:	(e) ©1981TI	(f) ©1983TI		(e) TI microcode copyright (f) Copyright of ROM code
Line 4:	(g) 24655	(h) Philippines		(g) Lot code (h) Assembly Site

11.3.3.2 TMS7000 Family Members Without On-Chip ROM

TMS7000 family members without on-chip ROM are standard device types, and therefore have a standard identification. Examples of TMS7000 family members without on-chip ROM include:

TMS70C00N2 TMS77C82FN
TMS70C08PG TMS70C02N2A

Figure 11–14. TI Standard Symbolization for Devices without On-Chip ROM

Line 1:	(a)	(b) TMS70C02NA	Key: (a) Texas Instruments Trademark (b) Standard Device Number
Line 2:	(d) ©1981TI	(c) DBUA8327	(c) Tracking mark and date code (d) TI microcode copyright
Line 3:	(e) 24655	(f) Philippines	(e) Lot code (f) Assembly Site

11.4 Development Support Tools Ordering Information

11.4.1 TMS7000 Macro Assembler/Linker

Part Number	Description	Operating System	Medium
TMDS7040810–02	TI/IBM PC	PC/MS-DOS	5 1/4" floppy
TMDS7040210–08	DEC VAX	VMS	1600 BPI mag tape
TMDS7040310–08	IBM Mainframe	MVS	1600 BPI mag tape
TMDS7040320–08	IBM Mainframe	CMS	1600 BPI mag tape

11.4.2 TMS7000 XDS Emulators

Part Number	XDS Model #
TMDS70622A1	CMOS Model 22 (220 V) for Europe
TMDS7062220	CMOS Model 22 (110 V) for U.S.

11.4.3 TMS7000 Evaluation Modules

Part Number	Description
RTC/EVM7000-1	TMS7000 CMOS Evaluation Module
KIT77C82PC	TMS77C82 Starter Kit

11.4.4 Adaptors and Hardware

Part Number	Description
RTC/PGMC82–06	TMS77C82 Programming Adaptor
RTC/ADP748A–06	TMS70C48 EVM Adaptor
ATC70CT40	TMS70CTXX 28-Pin Adaptor

Chapter A

TMS7000 Bus Activity Tables

This chapter describes the internal and external bus activity during each instruction execution and hardware operation (for example, interrupts). The **external bus** activity is the information seen on the *expansion bus*. The **internal bus** refers to the *address and data buses* that are part of the TMS7000 internal architecture. The information on the address and data buses, as well as the control pins, can be monitored externally when the device operates in any mode but single-chip. The internal and external buses' activity is documented on a cycle-by-cycle basis. The information in this section is useful to:

- Understand the external expansion bus for the purpose of designing an interface
- Calculate instruction execution times
- Gain a better understanding of microcomputer operation

The information on the bus activity tables is the same for NMOS and CMOS devices except for the IDLE instruction. This difference is noted in Table A–8.

Topics covered in this appendix include:

Section		Page
A.1	TMS7000 Operating Modes	A-2
A.2	TMS7000 Addressing Modes	A-3
A.3	Instruction Execution	A-5

Table A–1 contains an alphabetical listing of the TMS7000 instructions and indexes into the appropriate bus activity tables.

A.1 TMS7000 Operating Modes

The TMS7000 is a microcoded microcomputer with four operating modes:

- In the **single-chip mode**, there are four 8-bit I/O ports (ports A, B, C, and D) that provide 32 general purpose I/O lines.
- In **peripheral-expansion mode**, one 8-bit port (port C) becomes a multiplexed address and data bus and four output lines (the four most significant bits of port B) become the bus control signals. This is called the *external expansion bus*. The 8-bit address/data bus allows the TMS7000 to access up to 256 bytes of externally memory-mapped peripherals (excluding the dedicated on-chip peripheral-file locations).
- **Full-expansion mode** is similar to peripheral-expansion mode, except that another port D becomes the MSB of a 16-bit address (port C supplies the LSB). This means that the TMS7000 can access up to 64K bytes externally minus the number of bytes of on-chip ROM.
- **Microprocessor mode** is the same as full-expansion mode, except that the on-chip ROM (if any) is ignored and the entire 64K bytes are mapped off chip.

A.2 TMS7000 Addressing Modes

Because the TMS7000 implements a microcoded architecture, the microcode that fetches the instructions and their operands can be shared by many instructions. The instruction can be grouped according to the types of operands the instructions require and how the instructions are fetched. Each instruction group is based on one of the addressing modes supported by the TMS7000:

- **Double Operand Functions (DOPFUN)**

  ADD, ADC, AND, BTJO, BTJZ, CMP, DAC, DSB, MOV, MPY, OR, SBB, SUB, XOR

  These instructions require two operands for execution.

- **Miscellaneous Functions (MISCFUN)**

  DINT, EINT, IDLE, LDSP, NOP, POP ST, PUSH ST, RETI, RETS, SETC, STSP

  These instructions need no operands because the instruction function is implied in the opcode.

- **Long Addressing Functions (LAFUN)**

  BR, CALL, CMPA, LDA, STA

  These instructions require a 16-bit address which is used to address the entire 64K-byte address range of the TMS7000.

- **Single Operand Functions — Special (SOPFUNS)**

  CLR, DEC, INC, INV, MOV A B, MOV A RN, MOV B RN, SWAP, TSTA/CLRC, TSTB, XCHB

  These instructions need one operand for execution.

- **Single Operand Functions — Normal (SOPFUNN)**

  DECD, DJNZ, POP, PUSH, RL, RLC, RR, RRC

  These instructions need one operand for execution. Two groups of single operand instructions are needed because of the way CPU control is implemented and the number of supported single operand instructions.

- **Double Operand Functions — Peripheral (DOPFUNP)**

  ANDP, BTJOP, BTJZP, MOVP, ORP, and XORP.

  These instructions require two operands and interact with the TMS7000 peripheral file registers.

- **Move Double (MOVD)**

  MOVD

  Moves a register pair to a register pair and is the only instruction in this group.

- **Relative Jumps (RJMP)**

  JMP, JN/JLT, JZ/JEQ, JC/JHS, JP/JGT, JPZ/JGE, JNZ/JNE, JNC, JL

  These conditional and unconditional jumps alter program flow by adding or subtracting an 8-bit value with the program counter.

- **Traps (TRAP)**

  Trap 0 through Trap 23.

  These instructions are used to perform subroutine calls.

A.3 Instruction Execution

There are three phases of instruction execution:

1) Opcode fetch (instruction acquisition mode)
2) Operand addressing (addressing mode)
3) Functional operation on the operands (functional mode)

The bus activity tables, which list the number of cycles executed in each phase, are grouped according to these three phases:

- The **instruction acquisition sequence** is common to all instructions, so they are presented separately:

Table		Page
A-2	Instruction Acquisition Mode — Operation Code Fetch	A-11
A-3	Instruction Acquisition Mode — Interrupt Handling	A-11
A-4	Instruction Acquisition Mode — Reset	A-12

- To determine the number of **addressing mode** and **functional mode** cycles, locate the instruction's functional group (Table A–1) and reference the appropriate table. Table A–1 lists the TMS7000 instructions in alphabetical order with the corresponding addressing mode.

Table		Page
A–5	Double Operand Functions — Addressing Modes	A-13
A–6	Double Operand Functions — Functional Modes	A-14
A–7	Miscellaneous Functions — Addressing Modes	A-15
A–8	Miscellaneous Functions — Functional Modes	A-15
A–9	Long Addressing Functions — Addressing Modes	A-16
A–10	Long Addressing Functions — Functional Modes	A-16
A–11	Single Operand Functions, Special — Addressing Modes	A-17
A–12	Single Operand Functions, Special — Functional Modes	A-17
A–13	Single Operand Functions, Normal — Addressing Modes	A-18
A–14	Single Operand Functions, Normal — Functional Modes	A-18
A–15	Double Operand Functions, Peripheral — Addressing Modes	A-19
A–16	Double Operand Functions, Peripheral — Functional Modes	A-20
A–17	Move Double — Addressing Modes	A-21
A–18	Move Double — Functional Modes	A-21
A–19	Relative Jumps — Addressing and Functional Modes	A-22
A–20	Traps — Addressing and Functional Modes	A-22

Add all these cycles together to obtain the bus activity present during that instruction's execution.

Each table indicates whether a read or a write is performed during that cycle. The R/$\overline{W}$ signal is high for reads and low (logic zero) for writes. The memory control signals, ALATCH and $\overline{ENABLE}$, are asserted during both reads and writes. Note that the $\overline{ENABLE}$ signal is asserted only during external reads and writes.

Accesses other than internal RAM are long memory cycle (two-cycle) accesses. The timing of these accesses devices is specified in the memory interface timing specifications in Chapter 4. These long memory cycle accesses have been indicated by their grouping within the tables (two-cycle accesses are not separated by a horizontal line). For these cycle pairs, the first cycle uses the C and D ports for the address bus (C only for peripheral-expansion mode). In the second cycle, port C becomes a data bus. Figure A–1 illustrates the read/write information.

Although short memory cycles (RAM cycles) influence the external bus activity, no valid information is seen and the timing cannot be specified.

The following terms are used throughout this appendix:

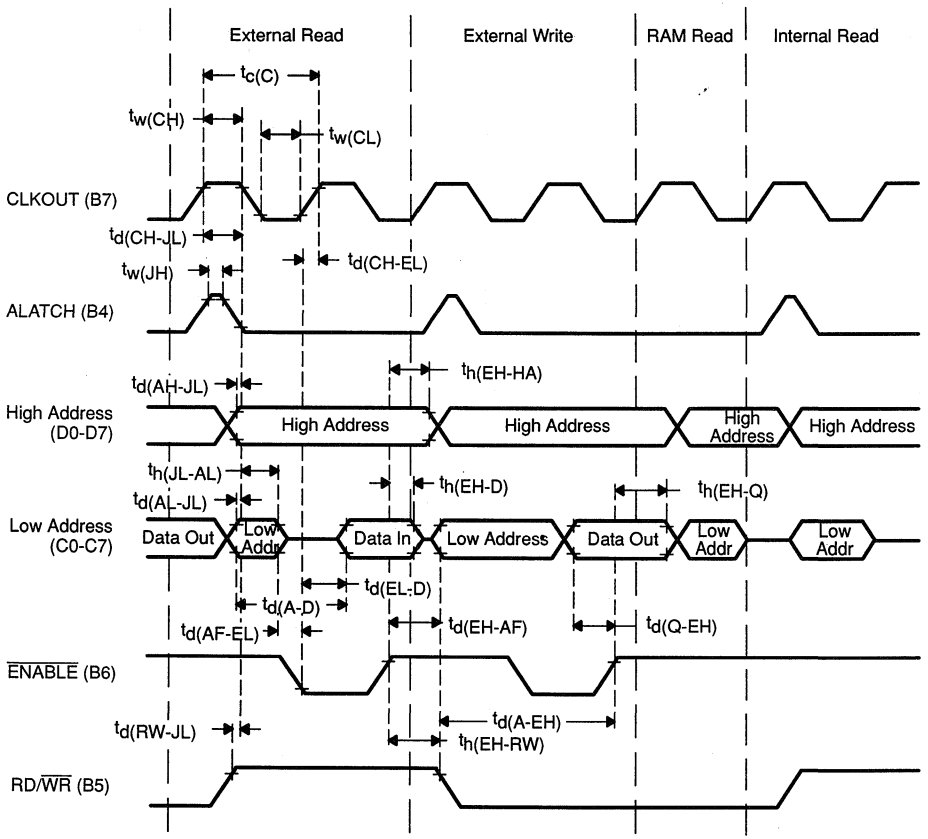
LSB least significant byte of a 16-bit value

MSB most significant byte of a 16-bit value

Rs (Rn source) the first operand listed

Rd (Rn destination) the second operand listed. The resulting value is stored at the Rd address.

Figure A–1. Read and Write Timing Diagram



A.3.1 An Example Using the Bus Activity Tables

Example A–1 illustrates the execution steps produced by the instruction

```
ADD     R5, R6
```

To construct the cycles required to execute the instruction, begin with the op-code fetch as shown in Example A–1. These three cycles:

1) Fetch the instruction opcode,
2) Increment the program counter, and
3) Prefetch register B.

Example A–1. Execution Steps for ADD (Instruction Acquisition)

Addressing Mode	Cycle	Address Bus	Data Bus	R/$\overline{W}$
All Instructions	1 2	Opcode address Opcode address	Irrelevant data Instruction opcode	R R
	3 †	Register B address	Register B contents	R

† The first two cycles fetch the ADD instruction's opcode and increment the program counter. The third state prefetches register B to speed up instructions that reference register B.

Note: This information is from Table A–2.

Example A–2. Execution Steps for ADD (Addressing Modes)

Addressing Mode	Cycle	Address Bus	Data Bus	R/$\overline{W}$
Rn, Rn	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data Rs address	R R
	3	Rs address	Rs data	R
	4 5	Opcode address + 2 Opcode address + 2	Irrelevant data Rd address	R R
	6	Rd address	Operand data	R

Note: The addressing mode is entered next and is found in Table A–5.

The ADD instruction is a double operand function, requiring two operands. Double operand functions are described in Table A–5 and Table A–6. Cycles 1 and 2 of this mode read the R5 operand address. Cycle 3 reads the register contents.

Note:

The internal register read (or write) is a one cycle operation. All other reads/writes are two cycles long, requiring that the address bus be held stable for two complete machine cycles.

Each machine cycle corresponds to one clock period of the CLKOUT signal (pin 2), starting with the rising edge of this signal. Cycles 4 and 5 read the Rd address, (R6) where the resultant value is placed. Cycle 6 reads the contents of register R6. Now, both operands are inside the CPU and the indicated function can be performed as shown in Example A–3 for functional modes (excerpted from Table A–6).

Example A–3. Execution Steps for ADD (Functional Modes)

Instruction Mode	Cycle	Address Bus	Data Bus	R/$\overline{W}$
ADD	1	Register address	Register data	$\overline{W}$

Once both operands are inside the CPU, only one cycle is needed to perform the add operation. The result is written back to register R6 during this cycle. A total of 10 cycles is required to perform an `ADD R5, R6`.

Table A–1. Alphabetical Index of Instruction Groups

Instruction	Address Mode	Table Number	Function
ADC	DOPFUN	Table A–5	Add with carry
ADD	DOPFUN	Table A–5	Add
AND	DOPFUN	Table A–5	And
ANDP	DOPFUNP	Table A–15	And value with peripheral port
BTJO	DOPFUN	Table A–5	Test bit and jump if one
BTJOP	DOPFUNP	Table A–15	Test peripheral bit and jump if one
BTJZ	DOPFUN	Table A–5	Test bit and jump if zero
BTJZP	DOPFUNP	Table A–15	Test peripheral bit and jump if zero
BR	LAFUN	Table A–9	Long branch
CALL	LAFUN	Table A–9	Subroutine call
CLR	SOPFUNS	Table A–11	Clear
CLRC	SOPFUNS	Table A–11	Clear status carry bit
CMP	DOPFUN	Table A–5	Compare value
CMPA	LAFUN	Table A–9	Compare value with register A
DAC	DOPFUN	Table A–5	Decimal add with carry
DEC	SOPFUNS	Table A–11	Decrement value
DECD	SOPFUNN	Table A–13	Decrement double register pair
DINT	MISCFUN	Table A–7	Disable interrupts
DJNZ	SOPFUNN	Table A–13	Decrement and jump if not zero
DSB	DOPFUN	Table A–5	Decimal subtract
EINT	MISCFUN	Table A–7	Enable interrupts
IDLE	MISCFUN	Table A–7	Idle (PC is held unchanged)
INC	SOPFUNS	Table A–11	Increment
INV	SOPFUNS	Table A–11	Invert
JMP	REL JUMPS	Table A–19	Unconditional relative jump
J<cnd>	REL JUMPS	Table A–19	Conditional relative jumps (JN/JLT, JZ/JEQ, JL, JC/JHS, JP/JGT, JPZ/JGE, JNZ/JNE, JNC)
LDA	LAFUN	Table A–9	Load register A from long address
LDSP	MICSFUN	Table A–7	Load stack pointer
MOV	DOFUN	Table A–5	Move a data value
MOV	SOPFUNS	Table A–11	Move with implied operand
MOVD	MOVD	Table A–17	Move a 16-bit value to register pair

Table A–1. Alphabetical Index of Instruction Groups (Concluded)

Instruction	Address Mode	Table Number	Function
MOVP	DOPFUNP	Table A–15	Move a data value to/from port
MPY	DOPFUN	Table A–5	Multiply two 8-bit values
NOP	MISCFUN	Table A–7	No operation
OR	DOPFUN	Table A–5	OR two values together
ORP	DOPFUNP	Table A–15	OR port value with another value
POP	SOPFUNN	Table A–13	POP a value off the stack
POPST	MISCFUN	Table A–7	POP stack value into status register
PUSH	SOPFUNN	Table A–13	PUSH a value onto the stack
PUSHST	MISCFUN	Table A–7	PUSH status register onto stack
RETI	MISCFUN	Table A–7	Return from interrupt
RETS	MISCFUN	Table A–7	Return from subroutine
RL	SOPFUNN	Table A–13	Rotate left
RLC	SOPFUNN	Table A–13	Rotate left through carry bit
RR	SOPFUNN	Table A–13	Rotate right
RRC	SOPFUNN	Table A–13	Rotate right through carry bit
SBB	DOPFUN	Table A–5	Subtract with borrow
SETC	MISCFUN	Table A–7	Set carry bit
STA	LAFUN	Table A–9	Store register A to long address
STSP	MISCFUN	Table A–7	Store stack pointer to register B
SUB	DOPFUN	Table A–5	Subtract
SWAP	SOPFUNS	Table A–11	Swap nibbles of an 8-bit value
TSTA	SOPFUNS	Table A–11	Test register A and set status
TSTB	SOPFUNS	Table A–11	Test register B and set status
TRAP n	TRAP	Table A–20	Trap to subroutine
XCHB	SOPFUNS	Table A–11	Exchange value with register B
XOR	DOPFUN	Table A–5	Exclusive OR
XORP	DOPFUNP	Table A–15	Exclusive OR with peripheral port

Table A–2. Instruction Acquisition Mode — Opcode Fetch

Addressing Mode	Cycle	Address Bus	Data Bus	R/$\overline{W}$
All Instructions	1†	Opcode address	Irrelevant data	R
	2	Opcode address	Instruction opcode	R
	3‡	Register B address	Register B contents	R

† Go to interrupt code listed for cycle 3 if an interrupt is pending.

‡ Go to addressing modes (Table A–5 through Table A–20).

Notes: 1) This mode is executed for all instructions to fetch the instruction's opcode.
2) Register B is prefetched to speed up the execution of instructions that reference register B.
3) The program counter is incremented during cycles 1 and 2 of this mode.
4) An interrupt check is performed during cycle 2. If an interrupt is detected, cycle 3 is not executed. Control is passed immediately to the interrupt handling code shown next.

Table A–3. Instruction Acquisition Mode — Interrupt Handling

Function	Cycle	Address Bus	Data Bus	R/$\overline{W}$
Interrupts	1†	Irrelevant data	Irrelevant data	–
	2	Irrelevant data	Irrelevant data	–
	3	Irrelevant data	Irrelevant data	–
	4	Irrelevant data	Irrelevant data	–
	5	SP register	Status register	$\overline{W}$
	6	Irrelevant data	Irrelevant data	–
(Reset entry)	7	Irrelevant data	Irrelevant data	–
	8	Irrelevant data	Irrelevant data	–
	9	Address >FF00 + vector	Irrelevant data	R
	10	Address >FF00 + vector	LSB INT vector	R
	11	Address >FF00 + vector	Irrelevant data	R
	12	Address >FF00 + vector	MSB INT vector	R
	13	SP contents	PCH contents	$\overline{W}$
	14	Irrelevant data	Irrelevant data	–
	15	SP + 1 contents	PCL contents	$\overline{W}$
	16	Irrelevant data	Irrelevant data	–
	17	Irrelevant data	Irrelevant data	–

† Jump to cycle number 5 if opcode was IDLE (>01). If it was an IDLE instruction, do not decrement PC because desired return is past the IDLE instruction.

Notes: 1) The program counter is decremented during cycles number 3 and 4. This is done because the instruction that the PC had pointed at has not been executed.
2) The status register is saved on the stack during cycle 5. The program counter is saved during cycles 13 and 15.
3) The vector is selected by hardware depending upon which interrupt was asserted.

Table A–4. Instruction Acquisition Mode — Reset

Function	Cycle	Address Bus	Data Bus	R/$\overline{W}$
Reset	1	Irrelevant data	Irrelevant data	R
	2	Irrelevant data	Zeroes	–
	3†	Address >0100	Zeroes	$\overline{W}$
	4	Address >0100	Zeroes	$\overline{W}$

† Jump to interrupt cycle 7 (see Reset Entry).

Notes: 1) A read operation is done the first cycle even though the address and data buses contain irrelevant data. This read is done to protect memory in case a long write was in progress when the reset action occurred.

2) The write to address >0100 is done to disable all interrupts.

3) The stack pointer is initialized to >01.

4) The program counter is stored in the register pairs A and B.

5) The reset function is initiated when the $\overline{RESET}$ line of the TMS7000 device is held at a logic zero level for at least five clock cycles. When an active signal is detected on $\overline{RESET}$, the sequence shown above is entered immediately after the current machine cycle is done.

Table A–5. Double Operand Functions — Addressing Modes (ADD, ADC, AND, BTJO, BTJZ, CMP, DAC, DSB, MOV, MPY, OR, SBB, SUB, XOR)

Function †	Cycle	Address Bus	Data Bus	R/$\overline{W}$
Rn, A	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data Rn address	R R
	3	Rn address	Rn data	R
	4	Register A address	Register A data	R
%n, A	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data Immediate value (%n)	R R
	3	Register A address	Register A data	R
Rn, B	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data Rn address	R R
	3	Rn address	Rn data	R
	4	Register B address	Operand data	R
Rn, Rn	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data R Rs address	R
	3	Rs address	Rs data	R
	4 5	Opcode address + 2 Opcode address + 2	Irrelevant data Rd address	R
	6	Rd address	Rd data	R
%n, B	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data Immediate data	R
	3	Register B address	Register B data	R
B, A	1	Register A address	Register A data	R
%n, Rn	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data Immediate data	R
	3 4	Opcode address + 2 Opcode address + 2	Irrelevant data Rn address	R
	5	Rn address	Rn data	R

† See functional modes in Table A–6.

Table A–6. Double Operand Functions — Functional Modes (ADD, ADC, AND, BTJO, BTJZ, CMP, DAC, DSB, MOV, MPY, OR, SBB, SUB, XOR)

Instructions †	Cycle	Address Bus	Data Bus	R/$\overline{W}$
MOV	1	Register address	Register data	W
AND	1	Register address	Register data	$\overline{W}$
OR	1	Register address	Register data	$\overline{W}$
XOR	1	Register address	Register data	$\overline{W}$
ADD	1	Register address	Register data	$\overline{W}$
ADC	1	Register address	Register data	$\overline{W}$
SUB	1	Register address	Register data	$\overline{W}$
SBB	1	Register address	Register data	$\overline{W}$
CMP	1	Irrelevant data	Irrelevant data	–
DAC	1 2 3 †	Register address Register address Register address	Register data Register data Register data	$\overline{W}$ R $\overline{W}$
DSB	1	Register address	Register data	$\overline{W}$
	2	Register address	Register data	R
	3	Register address	Register data	$\overline{W}$
MPY	1	Register B address	Register B data	$\overline{W}$
(Note 1)	2	Irrelevant data	Irrelevant data	–
	3	Irrelevant data	Irrelevant data	–
(9 iterations)	4	Register B address	Register B data	R
	5	Register B address	Register B data	$\overline{W}$
	6	Irrelevant data	Irrelevant data	–
	7	Irrelevant data	Irrelevant data	–
	8	Register A address	MSB mult. product	$\overline{W}$
	9	Irrelevant data	Irrelevant data	–
BTJO,BTJZ	1	Irrelevant data	Irrelevant data	–
(Note 2)	2	Opcode address + 1	Irrelevant data	R
	3	Opcode address + 1	Jump PC offset	R
	4	Opcode address + 1	Jump PC offset	R
	5	Irrelevant data	Irrelevant data	–
	6	Irrelevant data	Irrelevant data	–
	7	Irrelevant data	Irrelevant data	–

† Jump to instruction acquisition sequence.

Notes: 1) MPY — This microcode iterates to perform the multiply. The functional portion of the MPY instruction requires 40 states for execution.

2) BTJO, BTJOP — Not all states are executed. Either state 2 or state 3 is executed, but not both. The same applies to states 6 and 7.

Table A–7. Miscellaneous Functions — Addressing Modes (DINT, EINT, IDLE, LDSP, NOP, POP ST, PUSH ST, RETI, RETS, SETC, STSP)

Addressing Mode	Cycle†	Address Bus	Data Bus	R/$\overline{W}$
	1	SP contents	Stack value	R

† See functional modes in Table A–8.

Table A–8. Miscellaneous Functions — Functional Modes (DINT,EINT,IDLE,LDSP,NOP,POP ST,PUSH ST, RETI,RETS,SETC,STSP)

Addressing Mode	Cycle	Address Bus	Data Bus	R/$\overline{W}$
EINT	1	Irrelevant data	Irrelevant data	–
DINT	1	Irrelevant data	Irrelevant data	–
SETC	1 †	Irrelevant data	Irrelevant data	–
POP ST	1	SP contents	Stack data	R
	2 †	Irrelevant data	Irrelevant data	–
STSP	1	Irrelevant data	Irrelevant data	–
	2 †	Register B address	SP contents	$\overline{W}$
RETS	1	Irrelevant data	Irrelevant data	–
	2	Register address	Register data	R
	3 †	Irrelevant data	Irrelevant data	–
RETI	1	Irrelevant data	Irrelevant data	–
	2	Register address	Register data	R
	3	Irrelevant data	Irrelevant data	–
	4	SP contents	Register data	R
	5 †	Irrelevant data	Irrelevant data	–
LDSP	1 †	Irrelevant data	Irrelevant data	–
PUSH ST	1	Irrelevant data	Irrelevant data	–
	2 †	SP contents	Status register	$\overline{W}$
IDLE	1	Irrelevant data	Irrelevant data	–
	2 †	Irrelevant data	Irrelevant data	–

† Jump to instruction acquisition sequence.

Notes: 1) NOP does not have an execution state. From the addressing mode control is passed back to the instruction acquisition microcode.
2) An IDLE corresponds to a microcoded halt.

Table A–9. Long Addressing Functions — Addressing Modes (BR, CALL, CMPA, LDA, STA)

Addressing Mode	Cycle	Address Bus	Data Bus	R/$\overline{W}$
@n	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data MSB of long address	R R
	3 4	Opcode address + 2 Opcode address + 2	Irrelevant data LSB of long address	R R
	5 †	Irrelevant data	Irrelevant data	–
*Rn	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data Rn address	R R
	3	Rn address	LSB of long address	R
	4 †	Rn - 1 address	MSB of long address	R
@n(B)	1	Irrelevant data	Irrelevant data	–
	2 3	Opcode address + 1 Opcode address + 1	Irrelevant data MSB of long address	R R
	4 5	Opcode address + 2 Opcode address + 2	Irrelevant data LSB of long address	R R
	6	Irrelevant data	Irrelevant data	–
	7 †	Irrelevant data	Irrelevant data	–

† See functional modes in Table A–10.

Table A–10. Long Addressing Functions — Functional Modes (BR, CALL, CMPA, LDA, STA)

Addressing Mode	Cycle	Address Bus	Data Bus	R/$\overline{W}$
LDA	1 2	Operand address Operand address	Irrelevant data Operand data	R R
	3 †	Register A address	Operand data	$\overline{W}$
STA	1	Register A address	Register A contents	R
	2 3 †	Operand address Operand address	Register A contents Register A contents	$\overline{W}$ $\overline{W}$
BR	1	Irrelevant data	Irrelevant data	–
	2 †	Irrelevant data	Irrelevant data	–
CMPA	1 2	Operand address Operand address	Irrelevant data Operand data	R R
	3	Register A address	Register A contents	R
	4 †	Irrelevant data	Irrelevant data	–
CALL	1	Irrelevant data	Irrelevant data	–
	2	SP contents	PCH contents	$\overline{W}$
	3	Irrelevant data	Irrelevant data	–
	4	SP + 1$^-$	PCL	$\overline{W}$
	5	Irrelevant data	Irrelevant data	–
	6 †	Irrelevant data	Irrelevant data	–

† Jump to instruction acquisition sequence.

Table A–11. Single Operand Functions, Special — Addressing Modes (CLR, DEC, INC, INV, MOV A B, MOV A RN, MOV B RN, SWAP, TSTA/CLRC, TSTB, XCHB)

Addressing Mode	Cycle	Address Bus	Data Bus	R/$\overline{W}$
A	1 †	Register A address	Register A contents	R
B	1 †	Register B address	Register B contents	R
Rn	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data Rn address	R
	3 †	Rn address	Rn data	R R

† See functional modes in Table A–12.

Table A–12. Single Operand Functions, Special — Functional Modes (CLR, DEC, INC, INV, MOV A B, MOV A RN, MOV B RN, SWAP, TSTA/CLRC, TSTB, XCHB)

Addressing Mode	Cycle	Address Bus	Data Bus	R/$\overline{W}$
DEC	1	Register address	Register data	$\overline{W}$
INC	1	Register address	Register data	$\overline{W}$
INV	1	Register address	Register data	$\overline{W}$
CLR	1 †	Register address	Register data	$\overline{W}$
XCHB	1	Register B address	Register data	$\overline{W}$
	2 †	Register address	Register data	$\overline{W}$
SWAP	1	Irrelevant data	Irrelevant data	–
	2	Irrelevant data	Irrelevant data	–
	3	Irrelevant data	Irrelevant data	–
	4 †	Register address	Register data	$\overline{W}$
MOV A,B	1	Register A address	Register A data	R
	2 †	Register B address	Register A data	$\overline{W}$
MOV A,Rn	1	Register A address	Register A data	R
	2 †	Register address	Register A data	$\overline{W}$
MOV B,Rn	1 †	Register address	Register B data	$\overline{W}$
TSTA/CLRC	1	Register A address	Register A data	R
	2 †	Register address	Register data	$\overline{W}$
TSTB	1 †	Register B address	Register data	$\overline{W}$

† Jump to instruction acquisition sequence.

Table A–13. Single Operand Functions, Normal — Addressing Modes (DECD, DJNZ, POP, PUSH, RL, RLC, RR, RRC)

Addressing Mode	Cycle	Address Bus	Data Bus	R/$\overline{W}$
A	1 †	Register A address	Register A data	R
B	1 †	Register B address	Register B data	R
Rn	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data Rn address	R R
	3 †	Rn address	Rn data	R

† See functional modes in Table A–14.

Table A–14. Single Operand Functions, Normal — Functional Modes (DECD, DJNZ, POP, PUSH, RL, RLC, RR, RRC)

Instruction	Cycle	Address Bus	Data Bus	R/$\overline{W}$
PUSH	1	Irrelevant data	Irrelevant data	–
	2 †	SP contents	Register data	$\overline{W}$
POP	1	SP contents	Register data	R
	2 †	Register data	Register data	$\overline{W}$
RR	1	Register data	Register data	$\overline{W}$
RRC	1	Register data	Register data	$\overline{W}$
RL	1	Register data	Register data	$\overline{W}$
RLC	1 †	Register data	Register data	$\overline{W}$
DECD	1	Register data	Register data	$\overline{W}$
	2	Irrelevant data	Irrelevant data	–
	3	Irrelevant data	Irrelevant data	–
	4	Register address	Register data	R
	5 †	Register address	Register data	$\overline{W}$
DJNZ	1	Register address	Register data-1	$\overline{W}$
	2 ‡	Opcode address + 1	Irrelevant data	R
	3 †	Opcode address + 1	Jump PC offset	R
	4	Opcode address + 1	Jump PC offset	R
	5 §	Irrelevant data	Irrelevant data	–
	6 †	Irrelevant data	Irrelevant data	–
	7 †	Irrelevant data	Irrelevant data	–

† Jump to instruction acquisition sequence.
‡ If result is not = 0, jump to state 4.
§ If jump PC offset is positive, jump to state 7.

Table A–15. Double Operand Functions, Peripheral — Addressing Modes (ANDP, BTJOP, BTJZP, MOVP, ORP, XORP)

Addressing Mode	Cycle	Address Bus	Data Bus	R/$\overline{W}$
A, Pn	1	Register A address	Register A data	R
	2 3	Opcode address + 1 Opcode address + 1	Irrelevant data Pn address	R R
	4 5 †	Pn address Pn address	Irrelevant data Pn address	R
B, Pn	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data Pn address	R R
	3 4 †	Pn address Pn address	Irrelevant data Pn address	R R
%n, Pn	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data %n –immediate data	R R
	3 4	Opcode address + 2 Opcode address + 2	Irrelevant data Pn address	R R
	5 6 †	Pn address Pn address	Irrelevant data Pn address	R R
Pn, A	1	Register A address	Register A data	R
	2 3	Opcode address + 1 Opcode address + 1	Irrelevant data Pn address	R R
	4 5 †	Pn address Pn address	Irrelevant data Pn data	R
Pn, B	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data Pn address	R R
	3 4 †	Pn address Pn address	Irrelevant data Pn data	R R

† See functional modes in Table A–16.

Notes: 1) Addressing modes A, Pn and Pn, A fetch their operands the same way.
2) Addressing modes B, Pn and Pn, B fetch their operands the same way.

Table A–16. Double Operand Functions, Peripheral — Functional Modes (ANDP, BTJOP, BTJZP, MOVP, ORP, XORP)

Instruction	Cycle	Address Bus	Data Bus	R/$\overline{W}$
MOVP X, Pn	1	Pn address	Peripheral register data	$\overline{W}$
	2 †	Pn address	Peripheral register data	$\overline{W}$
MOVP Pn, A	1	Register A address	Register data	$\overline{W}$
MOVP Pn, B	1 †	Register B address	Register data	$\overline{W}$
ANDP	1	Pn address	Peripheral register data	$\overline{W}$
	2 †	Pn address	Peripheral register data	$\overline{W}$
ORP	1	Pn address	Peripheral register data	$\overline{W}$
	2 †	Pn address	Peripheral register data	$\overline{W}$
XORP	1	Pn address	Peripheral register data	$\overline{W}$
	2 †	Pn address	Peripheral register data	$\overline{W}$
BTJOP	1	Irrelevant data	Irrelevant data	–
	2 ‡	Opcode address + 1	Irrelevant data	R
	3 †	Opcode address + 1	Jump PC offset	R
	4	Opcode address + 1	Jump PC offset	R
	5 §	Irrelevant data	Irrelevant data	–
	6 †	Irrelevant data	Irrelevant data	–
	7 †	Irrelevant data	Irrelevant data	–
BTJZP	1	Irrelevant data	Irrelevant data	–
	2 ¶	Opcode address + 1	Irrelevant data	R
	3 †	Opcode address + 1	Jump PC offset	R
	4	Opcode address + 1	Jump PC offset	R
	5 §	Irrelevant data	Irrelevant data	–
	6 †	Irrelevant data	Irrelevant data	–
	7 †	Irrelevant data	Irrelevant data	–

† Jump to instruction acquisition sequence.
‡ If bit tested is equal to a 1, jump to state 4.
§ If jump PC offset is positive, jump to state 7.
¶ If bit tested is equal to a 0, jump to state 4.
Note: MOVP X, Pn – X is either register A or B, or an 8-bit immediate value %n.

Table A–17. Move Double — Addressing Mode (MOVD)

Instruction	Cycle	Address Bus	Data Bus	R/$\overline{W}$
%n, Rn	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data MSB of immediate data	R R
	3 4	Opcode address + 2 Opcode address + 2	Irrelevant data LSB of immediate data	R R
	5 †	Irrelevant data	Irrelevant data	–
Rn, Rn	1 2	Opcode address + 1 Opcode address + 1	Irrelevant data Rn source address	R R
	3	Rn source address	Rn data – LSB	R
	4 †	Rn – 1 source addr.	Rn – 1 data – MSB	R
%n(B), Rn	1	Irrelevant data	Irrelevant data	–
	2 3	Opcode address + 1 Opcode address + 1	Irrelevant data MSB of immediate data	R R
	4 5	Opcode address + 2 Opcode address + 2	Irrelevant data LSB of immediate data	R R
	6	Irrelevant data	Irrelevant data	–
	7 †	Irrelevant data	Irrelevant data	–

† See functional mode in Table A–18.

Table A–18. Move Double — Functional Mode (MOVD)

Instruction	Cycle	Address Bus	Data Bus	R/$\overline{W}$
MOVD	1	Irrelevant data	Irrelevant data	–
	2 3	Opcode address + 2/3 Opcode address + 2/3	Irrelevant data Destination Rn address	R R
	4 5	Irrelevant data Dest. Rn address	Irrelevant data LSB register data	– $\overline{W}$
	6	Irrelevant data	Irrelevant data	–
	7 †	Dest. Rn–1 address	MSB register data	$\overline{W}$

† Jump to instruction acquisition sequence.

Note: MOVD — States 2 and 3 will be opcode address + 2 for the %n, Rn and the Rn, Rn addressing modes. States 2 and 3 will be opcode address + 3 for the %n(B), Rn addressing mode.

Table A–19. Relative Jumps — Addressing and Functional Modes (JMP, JN/JLT, JZ/JEQ, JC/JHS, JP/JGT, JPZ/JGE, JNZ/JNE, JNC, JL)

Relative Jumps	Cycle	Address Bus	Data Bus	R/$\overline{W}$
	1 ‡	Opcode address + 1	Irrelevant data	R
	2 †	Opcode address + 1	Jump PC offset	R
	3	Opcode address + 1	Jump PC offset	R
	4 §	Irrelevant data	Irrelevant data	–
	5 †	Irrelevant data	Irrelevant data	–
	6 †	Irrelevant data	Irrelevant data	–

† Jump to instruction acquisition sequence.
‡ If jump condition is true, jump to state 3.
§ If jump offset is positive go to state 6.

Notes: 1) Cycle 1 tests the jump condition. If the jump is true, go to state 3, else execute state 2 and return to the instruction acquisition sequence.
2) Cycle 4 tests whether the jump offset is positive or negative. If the jump offset is positive, go to state 6.

Table A–20. Traps — Addressing and Functional Modes (Trap 0 through Trap 23)

Traps	Cycle	Address Bus	Data Bus	R/$\overline{W}$
Trap 0–7 (Group A) Trap 8–15 (Group B) Trap 16–23 (Group C)	1 1 1	Irrelevant data Irrelevant data Irrelevant data	Irrelevant data Irrelevant data Irrelevant data	– – –
	2	Irrelevant data	Irrelevant data	–
	3 4	Address >FF00+opcode Address >FF00+opcode	Irrelevant data LSB trap vector	R R
	5 6	Address>FF00+opcode –1 Address>FF00+opcode –1	Irrelevant data MSB trap vector	R R
	7	SP contents	PCH contents	$\overline{W}$
	8	Irrelevant data	Irrelevant data	–
	9	SP + 1 contents	PCL contents	$\overline{W}$
	10	Irrelevant data	Irrelevant data	–
	11 †	Irrelevant data	Irrelevant data	–

† Jump to instruction acquisition sequence.

Appendix B

TMS7000 NMOS to CMOS Conversion Guide

This appendix provides information to help upgrade an old TMS7000 NMOS design to a CMOS design. For additional information see the appropriate sections in this manual.

The basic areas of concern when converting from the TMS7000 NMOS devices to the TMS7000 CMOS devices fall into three categories; software, hardware, and electrical specifications. The following sections will outline the issues that need attention when converting from the NMOS to the CMOS devices.

Device conversions covered in this appendix include:

- B.1 Converting from a TMS70x0 device to a TMS70Cx0 device
- B.2 Converting from a TMS70x2 device to a TMS70Cx2 device

B.1 Converting from a TMS70x0 Device to a TMS70xC0 Device

The following areas require attention when converting from a TMS70x0 design to a TMS70Cx0 design:

B.1.1 Software

The instruction set is identical between the TMS70x0 NMOS devices and the TMS70Cx0 CMOS device. The only functional difference involves the IDLE instruction.

The IDLE instruction has no effect on the NMOS devices other than holding the device in a steady state. When an IDLE instruction is executed on a TMS70Cx0 device, it will go into either the wake-up or halt low power mode. The actual mode depends on what value is programmed into the IDLE bit (bit 5) of the Timer 1 control register. (0 for wake-up, 1 for halt).

B.1.2 Hardware

The on-chip hardware differences between the TMS70x0 and the TMS70Cx0 devices are summarized below:

- **RESET**: On the TMS70x0 NMOS devices, the output data bits of ports A, C, and D are set to all 1s after a reset. On CMOS devices, only port A's output data bits are set to all 1s; ports C and D output data bits are not altered during a reset. During initial power-up, the user program should account for this on the TMS70Cx0 devices.

- **INTERRUPTS**: The external interrupts (INT1 and INT3) of the TMS70x0 devices are edge and level triggered. INT3 on the TMS70Cx0 devices is also edge and level triggered. INT1 on the TMS70Cx0 devices is edge triggered only.

  If your present application uses the level sensitive feature on INT1, external circuitry may be required to use the TMS70Cx0 device in a similar manner.

B.1.3 Electrical Specifications

The electrical specification differences between the TMS70x0 and TMS70Cx0 devices may be compared by referencing the electrical specification sections for both device types located in Chapter 4 of this data manual.

Some of the more common differences are listed below:

Operating Ranges		
V_{CC}:	TMS70Cx0:	2.5 V – 6.0 V
	TMS70x0:	4.5 V – 5.5 V
F_{osc} (V_{CC} = 5 V ± 10%):	TMS70Cx0:	5 MHz
	TMS70x0	5 MHz (10 MHz/4 option)
I_{CC} (V_{CC} = 5 V, F_{osc} = 5 MHz):	TMS70Cx0:	12 mA
	TMS70x0:	150 mA
Input Levels:	(V_{CC} = 5 V)	
V_{IH}:	TMS70Cx0 =	3.5 V
	TMS70x0 =	2.0 V
V_{IL}:	TMS70Cx0	(V_{CC} = 5 V) = 1.5 V
	TMS70x0	(V_{CC} = 5 V) = 0.8 V
I_{OL}:	TMS70Cx0:	(V_{CC} = 5 V ± 10%) = 2.0 mA
	TMS70x0:	(V_{CC} = 5 V ± 10%) = 3.2 mA

B.2 Converting from a TMS70x2 Device to a TMS70Cx2 Device

The following areas require attention when converting from a TMS70x2 design to a TMS70Cx2 design:

B.2.1 Software

The instructions set is identical between the TMS70x2 NMOS devices and the TMS70Cx2 CMOS device. The only functional difference involves the IDLE instruction.

The IDLE instruction has no effect on the NMOS devices other than holding the device in a steady state. When an IDLE instruction is executed on a TMS70Cx2 device, the CMOS device will go into one of the wake-up or halt low power modes. The actual mode depends on what values are programmed into the following control bits:

Timer 1 control register 0	— bit 5 (T1HALT)
Timer 2 control register 0	— bit 5 (T2HALT)
Serial control register 0	— bit 7 (SPH)

B.2.2 Hardware

The on-chip hardware differences between the TMS70x2 and the TMS70Cx2 devices are summarized below:

Peripheral file: the peripheral (control) files differ greatly between the NMOS and CMOS devices. Major differences include the locations of the SMODE, SCLTO, and SSTAT control registers along with the additional timer and interrupt control registers associated with TMS70Cx2 devices.

Reset: On the TMS70x2 NMOS devices, the output data bits of ports A, C, and D are set to all 1s after a reset. On CMOS devices, only port As output data bits are set to all 1s; ports C and D output data bits are not altered during a reset. This means that the output value of ports C and D are not changed by a reset. During initial power-up, the user program should account for this on the TMS70Cx2 devices.

I/O pins: Port A pins 5 and 6 on the TMS70x2 are input only. These same pins on the TMS70Cx2 devices are fully bidirectional. The one pinout difference involves the SCLK function. It is multiplexed on the A6 pin on the TMS70x2 devices and the A4 pin on the TMS70Cx2 devices.

External Interrupts: The external interrupts (INT1 and INT3) of the TMS70x2 devices are edge only triggered. The external interrupts on the TMS70Cx2 devices are programmable to be individually triggered in one of the following ways:

- Falling edge only
- Falling edge and level sensitive
- Rising edge only
- Rising edge and level sensitive

The TMS70Cx2 devices may be programmed to function identically to the NMOS TMS70x2 devices by programming both external interrupts on the TMS70Cx2 device to be falling edge only.

Timers: The different timers of the TMS70x2 and TMS70Cx2 devices are summarized below:

- TMS70x2: Timers 1 and 2 are 8-bit timers with 5-bit prescale. Both timers have associated 8-bit capture registers. The minimum resolution is $f_{osc}/16$.
- TMS70Cx2: Timers 1 and 2 are 16-bit timers with 5-bit prescale. Both timers have 16-bit capture registers. The minimum resolution is $f_{osc}/4$.

  Additional user selectable features of the CMOS devices allow each timer to toggle an individual I/O pin on the reload pulse for each timer, and each timer may be individually selected to halt on an IDLE instruction. Also, stopping Timer 1 or 2 will clear the current interrupt flag of the timer stopped.

Serial Port: The different serial ports of the TMS70x2 and TMS70Cx2 devices are summarized below:

- TMS70x2: The peripheral files SMODE, SCLTO, and SSTAT are all addressed at the same location. The SCLK signal is always active during UART operation, and is multiplexed on the A6 pin.
- TMS70Cx2: The peripheral files SMODE, SCLTO and SSTAT are addressed at different locations. The SCLK pin is user selectable as the SCLK signal or as a general purpose I/O pin, and is multiplexed on the A4 pin. Also, the UART or the TMS70Cx2 may be selected to shut down during IDLE.

B.2.3 Electrical Specifications

The electrical specification differences between the TMS70x0 and TMS70Cx0 devices may be compared by referencing the electrical specification sections for both device types located in Chapter 4 of this data manual.

Some of the more common differences are listed below:

Operating Ranges		
V_{CC}:	TMS70Cx2:	2.5 V – 6.0 V
	TMS70x2:	4.5 V – 5.5 V
F_{osc} (V_{CC} = 5 V ± 10%):	TMS70Cx2:	6 MHz
	TMS70x2	8 MHz
I_{CC} (V_{CC} = 5 V, F_{osc} = 6 MHz):	TMS70Cx2:	21 mA
	TMS70x2:	210 mA
Input Levels:	(V_{CC} = 5 V)	
V_{IH}:	TMS70Cx2 =	3.5 V
	TMS70x2 =	2.0 V
V_{IL}:	TMS70Cx2	(V_{CC} = 5 V) = 1.5 V
	TMS70x2	(V_{CC} = 5 V) = 0.8 V
I_{OL}:	TMS70Cx2:	(V_{CC} = 5 V ± 10%) = 2.0 mA
	TMS70x2:	(V_{CC} = 5 V ± 10%) = 3.2 mA

Appendix C

Character Sets

The TMS7000 assembler recognizes the ASCII character set listed in Table C–1. Table C–2 lists characters that the assembler does not recognize, but may be recognized and acted upon by other programs. The device service routine for the card reader accepts and stores into the calling program's buffer all the characters listed.

Table C–1. ASCII Character Set

Base		Char	Base		Char	Base		Char	Base		Char
10	16		10	16		10	16		10	16	
0	00	NULL	32	20	SP	64	40	@	96	60	‘
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	"	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	’	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	–	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	>
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Table C–2. Control Characters

Hex Value	Decimal Value	Character
00	0	NUL
01	1	SOH
02	2	STX
03	3	ETX
04	4	EOT
05	5	ENQ
06	6	ACK
07	7	BEL
08	8	BS
09	9	HT
0A	10	LF
0B	11	VT
0C	12	FF
0D	13	CR
0E	14	SO
0F	15	SI
10	16	DLE
11	17	CD1
12	18	CD2
13	19	CD3
14	20	CD4
15	21	NAK
16	22	SYN
17	23	ETB
18	24	CAN
19	25	EM
1A	26	SUB
1B	27	ESC
1C	28	FS
1D	29	GS
1E	30	RS
1F	31	US
7F	127	DEL

Appendix D

Hexadecimal Instruction Table/Opcode Map

HIGH		0000 0	0001 1	0010 2	0011 3	0100 4	1010 5	0110 6	0111 7	1000 8	1001 9	1010 A	1011 B	1100 C	1101 D	1110 E	1111 F
LOW 0000	0	NOP								MOVP Pn,A			TSTA/ CLRC	MOV A,B	MOV A,Rn	JMP	TRAP 15
0001	1	IDLE									MOVP Pn,B			TSTB	MOV B,Rn	JN/ JLT	TRAP 14
0010	2		MOV Rn,A	MOV %n,A	MOV Rn,B	MOV Rn,Rn	MOV %n,B	MOV B,A	MOV %n,Pn	MOVP A,Pn	MOVP B,Pn	MOVP %n,Pn	DEC A	DEC B	DEC Rn	JZ/ JEQ	TRAP 13
0011	3		AND Rn,A	AND %n,A	AND Rn,B	AND Rn,Rn	AND %n,B	AND B,A	AND %n,Pn	ANDP A,Pn	ANDP B,Pn	ANDP %n,Pn	INC A	INC B	INC Rn	JC/ JHS	TRAP 12
0100	4		OR Rn,A	OR %n,A	OR Rn,B	OR Rn,Rn	OR %n,B	OR B,A	OR %n,R	ORP A,Pn	ORP B,Pn	ORP %n,Pn	INV A	INV B	INV Rn	JP/ JGT	TRAP 11
0101	5	EINT	XOR Rn,A	XOR %n,A	XOR Rn,B	XOR Rn,Rn	XOR %n,B	XOR B,A	XOR %n,R	XORP A,Pn	XORP B,Pn	XORP %n,Pn	CLR A	CLR B	CLR Rn	JPZ/ JGE	TRAP 10
0110	6	DINT	BTJO Rn,A	BTJO %n,A	BTJO Rn,B	BTJO Rn,Rn	BTJO %n,B	BTJO B,A	BTJO %n,R	BTJOP A,Pn	BTJOP B,Pn	BTJOP %n,Pn	XCHB A	XCHB B	XCHB Rn	JNZ/ JNE	TRAP 9
0111	7	SETC	BTJZ Rn,A	BTJZ %n,A	BTJZ Rn,B	BTJZ Rn,Rn	BTJZ %n,B	BTJZ B,A	BTJZ %n,R	BTJZP A,Pn	BTJZP B,Pn	BTJZP %n,Pn	SWAP A	SWAP B	SWA PRn	JNC/ JL	TRAP 8
1000	8	POP ST	ADD Rn,A	ADD %n,A	ADD Rn,B	ADD Rn,Rn	ADD %n,B	ADD B,A	ADD %n,R	MOVD %n,Rn	MOVD Rn,RN	MOVD %n,(B), Rn	PUSH A	PUSH B	PUSH Rn	TRAP 23	TRAP 7
1001	9	STSP	ADC Rn,A	ADC %n,A	ADC Rn,B	ADC Rn,Rn	ADC %n,B	ADC B,A	ADC %n,R				POP A	POP B	POP Rn	TRAP 22	TRAP 6
1010	A	RETS	SUB Rn,A	SUB %n,A	SUB Rn,B	SUB Rn,Rn	SUB %n,B	SUB B,A	SUB %n,R	LDA @n	LDA *Rn	LDA @n(B)	DJNZ A	DJNZ B	DJNZ Rn	TRAP 21	TRAP 5
1011	B	RETI	SBB Rn,A	SBB %n,A	SBB Rn,B	SBB Rn,Rn	SBB %n,B	SBB B,A	SBB %n,R	STA @n	STA *Rn	STA @n(B)	DECD A	DECD B	DECD Rn	TRAP 20	TRAP 4
1100	C		MPY Rn,A	MPY %n,A	MPY Rn,B	MPY Rn,Rn	MPY %n,B	MPY B,A	MPY %n,R	BR @n	BR *Rn	BR @n(B)	RR A	RR B	RR Rn	TRAP 19	TRAP 3
1101	D	LDSP	CMP Rn,A	CMP %n,A	CMP Rn,B	CMP Rn,Rn	CMP %n,B	CMP B,A	CMP %n,R	CMPA @n	CMPA *Rn	CMPA @n(B)	RRC A	RRC B	RRC Rn	TRAP 18	TRAP 2
1110	E	PUS H ST	DAC Rn,A	DAC %n,A	DAC Rn,B	DAC Rn,Rn	DAC %n,B	DAC B,A	DAC %n,R	CALL @n	CALL *Rn	CALL @n(B)	RL A	RL B	RL Rn	TRAP 17	TRAP 1
1111	F		DSB Rn,A	DSB %n,A	DSB Rn,B	DSB Rn,Rn	DSB %n,B	DSB B,A	DSB %n,R				RLC A	RLC B	RLC Rn	TRAP 16	TRAP 0

A – Register A
B – Register A
Rn – Register File Register
Pn – Peripheral File Register
%n – Immediate Addressing
@n – Direct Addressing
*Rn – Indirect Addressing

Appendix E

Instruction Opcode Set

	Single Operand			Dual Operand										Peripheral					Extended			Other	Status Word	
	A	B	Rn	A, B	B, A	Rn, A	%n ,A	Rn, B	%n ,B	Rn, Rn	%n ,Rn	A, Rn	B, Rn	A, Pn	Pn, A	B, Pn	Pn, B	%n, Pn	†	‡	§		¶	»
ADC					69	19	29	39	59	49	79												X	
ADD					68	18	28	38	58	48	78												X	
AND					63	13	23	33	53	43	73												X	
ANDP														83		93		A3					X	
BTJO					66	16	26	36	56	46	76												X	
BTJOP														86		96		A6					X	
BTJZ					67	17	27	37	57	47	77												X	
BTJZP														87		97		A7					X	
BR																			8C	9C	AC			
CALL																			8E	9E	AE			
CLR	B5	C5	D5																				X	
CLRC																						B0	X	
CMP					6D	1D	2D	3D	5D	4D	7D												X	
CMPA																			8D	9D	AD			
DAC					6E	1E	2E	3E	5E	4E	7E												X	
DEC	B2	C2	D2																				X	
DECD	BB	CB	DB																				X	
DINT																						06	X	
DJNZ	BA	CA	DA																				X	
DSB					6F	1F	2F	3F	5F	4F	7F												X	
EINT																						05	X	
IDLE																						01	X	
INC	B3	C3	D3																				X	
INV	B4	C4	D4																				X	
JMP																						E0		

† Direct
‡ Indirect
§ Indexed
¶ Condition Bits
» Interrupt Enable

	Single Operand			Dual Operand										Peripheral					Extended			Other	Status Word	
	A	B	Rn	A, B	B, A	Rn ,A	%n ,A	Rn ,B	%n ,B	Rn, Rn	%n ,Rn	A, Rn	B, Rn	A, Pn	Pn, A	B, Pn	Pn, B	%n ,Pn	†	‡	§		¶	»
JC/JHS																						E3		
JN/JLT																						E1		
JNC/JL																						E7		
JNZ/JNE																						E6		
JP/JGT																						E4		
JPZ/JGE																						E5		
JZ/JEQ																						E2		
LDA																			8A	9A	AA		X	
LDSP																						0D		
MOV				C0	62	12	22	32	52	42	72	D0	D1									X		
MOVD																			88	98	A8		X	
MOVP														82	80	92	91	A2				X		
MPY					6C	1C	2C	3C	5C	4C	7C												X	
NOP																						00		
OR					64	14	24	34	54	44	74												X	
ORP														84		94		A4				X		
POP	89	C9	D9																			08	X	
PUSH	88	C8	D8																			0E	X	
RETI																						0B		
RETS																						0A		
RL	BE	CE	DE																				X	
RLC	BF	CF	DF																				X	
RR	BC	CC	DC																				X	
RRC	BD	CD	DD																				X	
SBB					6B	1B	2B	3B	5B	4B	7B												X	
SETC																						07	X	
STA																			8B	9B	AB		X	
STSP																						09	X	
SUB					6A	1A	2A	3A	5A	4A	7A												X	
SWAP	B7	C7	D7																				X	
TSTA																						80	X	
TSTB																						C1	X	
TRAP																						E3-EF	X	
XCHB	B6		D6																				X	
XOR					65	15	25	35	55	45	75												X	
XORP														85		95		A5				X		

† Direct
‡ Indirect
§ Indexed
¶ Condition Bits
» Interrupt Enable

Appendix F

Glossary

A

ADDR: Port a data-direction register.

ALU: Arithmetic logic unit.

APORT: Port A data register.

assembler: Any program that converts mnemonic and symbolic machine code into machine language.

ASYNC: Communications mode, bit 1 in the serial mode register (SMODE).

asynchronous communication mode: A mode used by the serial port to communicate with peripheral devices. Requires framing bits but does not require a synchronizing clock.

B

BPORT: Port B data register.

BRKDT: Break detect, bit 6 in the serial port status register (SSTAT).

C

C bit: Carry bit in the status register.

CDDR: Port C data-direction register.

CHAR1, CHAR2: Number of bits per character, bits 2 and 3 in the serial mode register (SMODE).

CLK: Serial clock source, bit 6 in serial control register 1 (SCTL1).

CPORT: Port C data register.

CRC: Customer Response Center.

CrossWare: Texas Instruments macro assemblers and linkers.

CS: Chip select.

D

DDDR: Port D data-direction register.

DDR: Data direction register.

direct memory addressing mode: Uses a 16-bit address that contains an operand.

DIP: Dual-inline package.

directive: A mnemonic instruction to the assembler, executed during assembly.

DPORT: Port D data register.

dual register addressing mode: Uses a source and a destination register as 8-bit operands.

E

EC1: Timer 1 event counter.

EC2: Timer 2 event counter.

EDDR: Port E data-direction register.

EPORT: Port E data register.

ER: Error reset, bit 4 of serial control register 0 (SCTL0).

EVM: Evaluation module.

expression: A sequence of symbols, constants, and operators, to which a numerical value can be assigned during assembly.

extended addressing mode: An addressing mode which uses a 16-bit address.

F

FE: Framing error, bit 6 of the serial port status register (SSTAT).

FDDR: Port F data-direction register.

FFE: Form factor emulator; an EPROM or piggyback device that emulates or replaces a masked-ROM device.

F_{osc}: External oscillator frequency.

FPORT: Port F data register.

full-expansion mode: A TMS7000 operating mode that extends addressing capability to the full 64K-byte limit.

G

GDDR: Port G data-direction register.

GPORT: Port G data register.

H

halt mode: A low-power mode entered by the CMOS devices in which the on-chip timer logic is disabled.

I

I bit: Global interrupt enable bit (in the status register).

immediate addressing mode: Uses an immediate 8-bit address.

indexed addressing mode: Generates a 16-bit address by adding the contents of register B to a 16-bit direct memory address.

IOCNT0: I/O control register 0.

IOCNT1: I/O control register 1.

IOCNT2: I/O control register 2.

isosynchronous communication mode: A hybrid communications protocol which combines features of asynchronous and serial I/O communications; uses framing bits and a serial clock.

link control file: Contains commands which control the link process.

linker: Collects and interconnects relocatable elements to produce an absolute element.

M

mask option: A device option, such as a clock option, that is placed on a manufacturing template, or mask, copying the actual circuit onto the silicon device; cannot be changed by software.

MC pin: Mode control pin. When this pin is set to 1 (5 V), the microprocessor mode of device operation is entered.

microprocessor mode: A mode of operation intended for applications that do not justify the use of on-chip ROM. All memory accesses except for

internal RAM and on-chip peripheral file locations are addressed externally.

MULTI: Multiprocessor mode, bit 0 of the serial mode register (SMODE).

MUX: Multiplexor. This is a mask option for TMS70Cx8.

N

N bit: Sign bit in the status register.

NPRF: New Product Release Form.

O

OE: Overrun error, bit 4 in the serial port status register (SSTAT).

OTPROM: One time programmble ROM.

P

PC: Program counter.

PE: Parity error, bit 3 in the serial port status register (SSTAT).

PEN: Parity enable, bit 4 in the serial mode register (SMODE).

peripheral-expansion mode: An operating mode which allows use of on-chip ROM and also allows addressing off-chip locations (peripheral devices).

peripheral file addressing mode: Refers to instructions that perform I/O tasks; either the source or the destination is a peripheral file register.

peripheral file instructions: MOVP, ANDP, ORP, XORP, BTJOP, and BTJZP.

PEVEN: Parity even, bit 5 of the serial mode register (SMODE).

PF: Peripheral file.

piggyback: A device used as a form-factor emulator for masked-ROM devices.

PLA: Programmed logic array.

PLCC: Plastic-leaded chip carrier.

program counter relative addressing mode: Used by all jump instructions; adds an offset to the PC value to form the address.

PWM: Pulse width modulation.

Q

QFP: Quad flat package.

R

R bit: Function of the EPROM part TMS77C82 that prevents a protected code from being modified or read externally.

RF: Register file.

RTC: Regional Technology Center.

RXBUF: Receiver buffer.

RXD: Receive data, line A5.

RXEN: Receiver enable, bit 2 in serial control register 0 (SCTL0).

RXRDY: Receiver ready, bit 1 in the serial status register (SSTAT).

RXSHF: RX shift register.

S

SCAT: Strip chip architecture technology.

SCLK: Serial clock source, pin A6.

SCTL0: Serial port control register 0.

SCTL1: Serial port control register 1.

serial I/O Mode: A serial-port communication mode which uses an external clock to synchronize the receiver and the transmitter; stop bits are also used.

single register addressing mode: Uses a single register that contains an 8-bit operand.

single-chip mode: An operation mode in which the device functions as a standalone microcomputer with no off-chip memory expansion bus.

SIO: Serial I/O or communications mode, bit 6, serial mode register (SMODE).

SLEEP: Sleep, bit 5, serial control register 1 (SCTL1).

SMODE: Serial port mode register.

SP: Stack pointer.

SSTAT: Serial port status register.

ST: Status register.

START: Timer 3 start, bit 7, serial control register 1 (SCTL1).

STOP: Stop, bit 7, serial mode register (SMODE).

T

TMP: Prefix for devices that conform to the final electrical specifications but have not completed quality and reliability verification.

TMS: Device prefix for fully qualified production devices.

TMX: Device prefix for experimental devices that are not representative of the device's final electrical specifications.

TXBUF: Transmitter buffer, write-only PF register P23.

TXD: Transmission data, uses line B3.

TXEN: Transmit enable, bit 0, serial control register 0 (SCTL0).

TXRDY: Transmitter ready, bit 0, serial port status register (SSTAT).

TXSHF: Transmitter shift register.

T1CTL: Timer 1 control register.

T1CTL0: Timer 1 control register 0/LSB capture reload register value.

T1CTL1: Timer 1 control register 1/MSB readout reload register.

T1DATA: Timer 1 data register.

T1LSDATA: Timer 1 LSB decrementer latch/LSB decrementer value.

T1MSDATA: Timer 1 MSB decrementer latch/MSB readout latch.

T1OUT: Timer 1 output.

T2CTL: Timer 2 control register.

T2CTL0: Timer 2 control register 0/LSB capture latch value.

T2CTL1: Timer 2 control register 1/MSB readout reload register.

T2DATA: Timer 2 data register.

T2OUT: Timer 2 output.

T2LSDATA: Timer 2 LSB decrementer latch/LSB decrementer value.

T2MSDATA: Timer 2 MSB decrementer latch/MSB readout latch.

T3DATA: Timer 3 data register.

T3ENB: Timer 3 enable, bit 2, serial control register 1 (SCTL1).

T3FLG: Timer 3 flag, bit 3, serial control register 1 (SCTL1).

U

UR: Software UART reset, bit 6, serial control register 0 (SCTL0).

W

wake-up mode: A low-power mode entered by the CMOS devices in which the oscillator and timer logic remain active.

WU bit: Wake-up, bit 4, serial control register 1 (SCTL1).

WUT: Wake-up temporary flag.

X

XDS: Extended development support.

Z

Z bit: Zero bit, status register.

Index

Sy

$ASG, assign values to variable components verb, 8-9, 8-11, 8-12, 8-18
$DEF keyword, 8-13
$ELSE, alternate conditional block verb, 8-21
$ELSE, alternate conditional block verb. *See* $IF, $ELSE
$END, 8-5
  end macro definition verb, 8-18, 8-22
$ENDIF, terminate conditional block verb, 8-23
$ENDIF, terminate conditional block verb. *See* $IF, $ENDIF
$IF, begin conditional block verb, 8-7, 8-24
$MAC keyword, 8-13
$MACRO, 8-2, 8-9
  macro definition verb, 8-2, 8-5, 8-9, 8-16, 8-18, 8-26
$PCALL keyword, 8-14
$POPL keyword, 8-14
$PSYM keyword, 8-14
$REF keyword, 8-13
$REL keyword, 8-13
$STR keyword, 8-13
$UNDF keyword, 8-13

A

A6/SCLK/EC2, 3-14, 3-63, 3-64
A7/EC1, 3-14, 3-23, 3-63
absolute code, 5-16, 5-20, 5-26, 5-40, 5-43, 5-52, 7-2
ADC, Add with Carry Instruction, 6-8, 6-16, 9-25
ADD, Add Instruction, 6-8, 6-17, 9-25
addition instructions, 6-16, 6-17, 6-30, 6-38, 9-25, 9-38
ADDR, 3-23
address space, 3-7
address/data bus, 3-10, 3-15, 3-27
addressing modes, 6-3
  direct memory, 6-6
  dual register, 6-4
  immediate, 6-5
  indexed, 6-7
  peripheral file, 6-4
  program counter relative, 6-5
  register file indirect, 6-6
  single register, 6-3
ALATCH, 3-14, 3-26
AND, Logical AND Instruction, 6-8, 6-18
ANDP, 3-24, 3-60
  AND peripheral register instruction, 3-78, 6-8, 6-19
APORT, 3-23
arithmetic operators, 5-10, 8-7
assembler, 5-1, 5-2, 5-50, 5-63, 7-1
assembler cross-reference listing, 5-53
assembler output, 5-50
assembler source listing, 5-50
assembler symbol table, 8-9
assembly language, 5-1, 6-1, 6-3, 6-4, 6-8
assembly process, 5-1
assembly-time constants, 5-7
AST, 8-9
ASYNC bit, 3-71
asynchronous communication mode, 3-68, 3-71, 3-83, 9-12
attribute component (of a variable), 8-9

B

B3/TXD, 3-15
bidirectional I/O logic, 3-12

binary integers, 5-6
binary mode (mode variables), 8-10
Boolean operators, 8-7
BPORT, 3-24
BR, branch instruction, 6-9, 6-20, 9-31
BRKDT bit, 3-76
BTJO, bit test and jump if one instruction, 6-9, 6-21
BTJOP, bit test and jump if one - peripheral instruction, 6-9, 6-22
BTJZ, bit test and jump if zero instruction, 6-9, 6-23
BTJZP, bit test and jump if zero – peripheral instruction, 6-9, 6-24
bus activity tables, A-1
bus control signals, 3-14, 3-26
  ALATCH, 3-14
  CLKOUT, 3-14
  $\overline{\text{ENABLE}}$, 3-14
  R/$\overline{\text{W}}$, 3-14
BYTE, 5-50

C

C (carry) bit, 3-8, 6-27, 6-60, 9-24
CALL, call instruction, 6-9, 6-25, 9-28
capture latch, 3-61
cascade bit, 3-64
CDDR, 3-24
ceramic oscillator clock option, 3-33
CHAR1, CHAR2 bits, 3-72
character constants, 5-7
character sets, C-1
character strings, 5-9
chip select option, 11-6
CLK bit, 3-74, 3-78
CLKIN, 11-5
CLKOUT, 3-14, 3-26
clock options, 3-31—3-35, 11-5
  crystal oscillator, 3-33
  R-C oscillator, 3-33
clock source, 3-83
CLR, clear instruction, 6-10, 6-26
CLRC, clear the carry bit instruction, 6-10, 6-27
CMODE bit, 3-72
CMOS devices, 3-61
  See also Chapter 2 and Chapter 4
  clock options, 3-33
CMP, compare instruction, 6-10, 6-28, 9-24
CMPA, compare accumulator extended instruction, 6-10, 6-29, 9-24
command field, 5-2, 5-5
comment field, 5-2, 5-5
common-relocatable code, 7-2
communication mode, 3-73
  asynchronous, 3-68, 3-71, 3-83, 9-12
  isosynchronous, 3-68, 3-71, 3-84, 9-12
  serial I/O, 3-68, 9-12
compare instructions, 6-28, 6-29
conditional jumps, 6-41
conditional processing, 8-21, 8-23, 8-24
constants, 5-6, 5-10, 8-7
  assembly-time, 5-7
  binary integers, 5-6
  characters, 5-7
  decimal integers, 5-6
  hexadecimal integers, 5-6
counter, 3-90
CPORT, 3-24
CrossWare, ordering information, 11-19
crystal clock source, 3-31
crystal oscillator clock option, 3-33, 11-5

D

DAC, decimal add with carry instruction, 6-10, 6-30
DATA, 5-50
data register, 3-24
data-direction register, 3-24
data-relocatable code, 5-26, 5-29, 5-43, 7-2
DDDR, 3-24, 3-28
DEC, decrement instruction, 6-10, 6-31
DECD, decrement double instruction, 6-10, 6-32
decimal integer constants, 5-6
decimal integers, 5-6
DEF, 5-50, 7-6
development support, 10-1—10-12
  ordering information, 11-19
device initialization, 3-39
DINT, disable interrupts instruction, 6-10
direct memory addressing mode, 6-6, 9-28
directives
  for linking programs, 5-13

DEF, 5-25
LOAD, 5-35
REF, 5-42
SREF, 5-45
miscellaneous
COPY, 5-21
END, 5-30
MLIB, 5-37
that affect assembler output, 5-13
IDT, 5-33
LIST, 5-34
OPTION, 5-38
PAGE, 5-39
TITL, 5-47
UNL, 5-48
that affect the location counter, 5-13
AORG, 5-16
BES, 5-17
BSS, 5-18
CEND, 5-20
CSEG, 5-22
DEND, 5-26
DORG, 5-27
DSEG, 5-29
EVEN, 5-32
PEND, 5-40
PSEG, 5-41
RORG, 5-43
that initialize constants, 5-13
BYTE, 5-19
DATA, 5-24
EQU, 5-31
TEXT, 5-46
divide-by-2 clock option, 3-31, 11-5
divide-by-4 clock option, 3-31, 11-5
division instructions, 9-43, 9-44
DJNZ, decrement register and jump if not zero instruction, 6-10, 6-34
dollar sign ($), 5-8
DPORT, 3-24, 3-28
DSB, decimal subtract with borrow instruction, 6-11, 6-35
dual register addressing mode, 6-4
dummy section, 5-27

E

EDDR, 3-21, 3-24
EINT, enable interrupts instruction, 3-49, 6-11, 6-36
ENABLE, 3-14, 3-26
END, 5-50, 8-5
END linker command, 7-5
END macro definition verb, 8-22
EPORT, 3-21, 3-24
EPROM devices, 2-18, 2-20
EQU, 5-50
ER bit, 3-75
error messages
assembler, 5-51, 5-53
macros, 8-32
evaluation of arithmetic expressions, 5-11
event counter, 3-53
EVM, ordering information, 11-19—11-20
expressions, 5-10
arithmetic evaluation, 5-11
using arithmetic operators, 5-10
using externally defined symbols, 5-12
using logical operands, 5-11
using parentheses, 5-11
using relocatable symbols, 5-11
well-defined, 5-11
extended addressing modes, 6-3
direct, 9-28
indexed, 9-28
register file indirect, 9-28
external clock, 3-23, 3-24, 11-5
external clock source, 3-31, 3-33
external event-counter mode, 3-23
external interrupts, 3-50, 3-51
external references, 5-60
externally defined symbols, 5-12

F

FDDR, 3-21, 3-24
FE bit, 3-76
FORMAT linker command, 7-5
FPORT, 3-21, 3-24
frame bit, 3-83
full-expansion mode, 3-27
memory map, 3-29

G

GDDR, 3-21, 3-24

global interrupt enable (I) bit, 3-8, 3-40, 3-43, 3-44, 3-47, 3-48, 3-49
global interrupt enable bit, 3-8
GPORT, 3-21, 3-24

H

halt mode, 3-36, 3-37
HALT/DELAY pin, 3-38—3-61
hardware UART, 3-68, 9-19
hexadecimal integer constants, 5-6

I

I (global interrupt enable) bit, 3-49, 9-24, 9-32
I/O control registers, 3-45
I/O ports, 3-10—3-15, 3-28
  full-expansion mode, 3-28
  peripheral-expansion mode, 3-26
  single-chip mode, 3-22, 3-23
IADD, 3-76
IDLE, 3-36
  idle until interrupt instruction, 3-37, 6-11, 6-37
IDT, 5-50, 7-6
immediate addressing mode, 3-25, 6-5
INC, increment instruction, 6-11, 6-38
INCLUDE linker command, 7-5
indexed addressing mode, 6-7, 9-28
instruction timing, A-1
INT4, 3-75
Intel 8051, 3-68
Intel protocol, 3-71, 3-88
interrupt
  DINT instruction, 6-33
  edge-sensitive, 3-43
  EINT instruction, 6-36
  external, 3-50, 3-51
  level 0, 3-39
  level-sensitive, 3-43
  logic for maskable interrupts, 3-43
  multiple, 3-49
  priority, 3-39
  RETI instruction, 6-53
  timer interrupts, 3-66
interrupts, 3-39—3-52, 9-32
  EINT instruction, 6-35
INTn clear bit, 3-49
INTn enable bit, 3-47
INTn flag bit, 3-47, 3-66
INV, Invert Instruction, 6-11, 6-39
IOCNT0 register, 3-16, 3-22, 3-25, 3-27, 3-29, 3-45
IOCNT1 register, 3-45, 3-47
IOCNT2 register, 3-45, 3-47
IPC, 9-27
isosynchronous communication mode, 3-68, 3-71, 3-84, 9-12

J

J<cnd>, conditional jump instruction, 6-41
JC, 6-11
JEQ, 6-11
JGE, 6-11
JGT, 6-11
JHS, 6-11
JL, 6-11
JMP, jump unconditional instruction, 6-11, 6-40
JNC, 6-11
JNE, 6-11
JNZ, 6-11
JP, 6-11
JPZ, 6-11
jump instructions, 6-21, 6-22, 6-23, 6-24, 6-34, 6-39, 9-24
  conditional jumps, 6-40
JZ, 6-11

K

keywords, 8-13
  parameter attribute components, 8-13
    $PCALL, 8-13
    $POPL, 8-13
    $PSYM, 8-13
  symbol attribute components, 8-13
    $DEF, 8-13
    $MAC, 8-13
    $REF, 8-13
    $REL, 8-13
    $STR, 8-13
    $UNDF, 8-13

L

label field, 5-2, 5-5, 5-8

LDA, load register A instruction, 6-11, 6-42
LDSP, load stack pointer instruction, 6-12, 6-43
length component (of a variable), 8-9
link control file, 7-4
link editor, 7-4—7-5
linker commands, 7-4
linking directives, 7-6
  DEF, 5-25, 5-35, 7-6
  IDT, 5-33, 7-6
  REF, 5-42, 7-6
  SREF, 5-45, 7-6
linking program modules, 7-1
LIST, 5-50
location counter, 5-5
logical AND, 8-7
logical NOT, 8-7
logical operands, 5-11
logical OR, 8-7
low-power modes, 3-36, 3-61
  halt, 3-36
  halt mode, 3-37
  wake-up, 3-36
  wake-up mode, 3-37

M

MACLIB files, 8-2
macro assembler, 5-1
macro libraries, 8-2
macro symbol table, 8-9
macros
  assembler symbol table, 8-9
  assigning parameter values, 8-15, 8-26
  calls, 8-1
  conditional processing, 8-21, 8-23, 8-24
  constants, 8-7
  declaring variables, 8-27
  definition, 8-2, 8-26
  error messages, 8-32
  keywords, 8-13
  MACLIB files, 8-2
  macro libraries, 8-2
  MLIB directive, 8-2
  MLIST files, 8-3
  MST, 8-9
  search order, 8-2
  strings, 8-7
  substitution, 8-1
  symbol components, 8-11
  symbols, 8-9
  variable components, 8-9
  variables, 8-9
    binary mode access, 8-10
    definition, 8-9
    macro symbol table, 8-9
    parameters, 8-9
    string mode access, 8-10
    symbol table, 8-9
    unqualified variables, 8-11
    variable qualifiers, 8-11
  verbs, 8-18
mask options, 3-31, 11-5
MC pin, 3-16, 3-22, 3-25, 3-27, 3-29
mechanical data, 11-7
memory modes, 3-16—3-30
  full-expansion, 3-27
  microprocessor, 3-29
  microprocessor mode, 9-2
  peripheral-expansion, 3-25
  single-chip, 3-22
microprocessor mode, 3-29, 9-2
  interface example, 9-2
  memory map, 3-30
miscellaneous, directives, 5-14
MLIB, 8-2
MLIST files, 8-3
mnemonics, 5-1
mode control (MC) pin, 3-16
model statements, 8-28
Motorola 6801, 3-68
Motorola protocol, 3-71, 3-86—3-88
MOV, move instruction, 6-12, 6-44
MOVD, move double instruction, 6-12, 6-45
move instructions, 6-44, 6-45, 6-46
MOVP, 3-23, 3-48
  move to/from peripheral register instruction, 6-12, 6-46
MPY, multiply instruction, 6-12, 6-47, 9-30
MST, 8-9
MULTI bit, 3-71
multiple interrupts, 3-49
multiplication instructions, 6-47, 9-30, 9-42
multiprocessor communication mode
  Intel protocol, 3-71

Motorola protocol, 3-71, 3-86
multiprocessor communication modes, 3-86
multiprocessor communication protocol, Intel protocol, 3-88
multiprocessor protocols, 3-68, 3-71
Intel 8051, 3-68
Motorola 6801, 3-68
MUX option, 11-6

N

N (sign) bit, 3-8, 9-24
NMOS devices. *See* Chapter 2 and Chapter 4
NMOS to CMOS conversion guide, B-1
NOP, no operation instruction, 6-12, 6-48

O

object code, 5-50, 5-55, 5-60
object program, 5-1
object record format, 5-59—5-64
OE bit, 3-76
offset calculation, 6-5
on-chip RAM, 3-7
on-chip timer/event counter, 3-14
operand field, 5-2, 5-5, 5-8, 5-10
operators, 8-7
OR, logical OR instruction, 6-12, 6-49
ORP, OR peripheral register instruction, 3-78, 6-13, 6-50
oscillator options, 3-33, 11-5

P

P10, 3-24
P4, 3-23
P5, 3-23
P6, 3-24
P8, 3-24
packaging, 11-7
PAGE, 5-50
parameter attribute component keywords, 8-13
parameters, 8-15
as macro variables, 8-9
parentheses, 5-11
parity enable, 3-72
PC, 3-9
PE bit, 3-75
PEN bit, 3-72
peripheral file, 3-7
peripheral-expansion mode, 3-25
memory map, 3-26
peripheral-file addressing mode, 6-4
peripheral-file instructions, 6-70
peripheral-file instructions, 3-7, 3-25, 6-19, 6-22, 6-24, 6-46, 6-50, 9-34
AND peripheral register, 3-24
move to/from peripheral register instruction, 3-25
OR peripheral register, 3-24
XOR peripheral register, 3-24
PEVEN bit, 3-72
PF, 3-7
POP, POP from stack instruction, 6-13, 6-51
port A, 3-10, 3-14, 3-23, 3-26
port B, 3-10, 3-14, 3-24, 3-26
port C, 3-10, 3-15, 3-24, 3-27
port D, 3-10, 3-15, 3-24, 3-27
port E, 3-10, 3-13, 3-15, 3-24, 3-27
port F, 3-10, 3-13, 3-15, 3-24, 3-27
port G, 3-10, 3-13, 3-15, 3-24, 3-27
port symbols, 5-8
power-down mode, 3-36
power-up reset, 3-42
PRE3(1), PRE3(0) bits, 3-77
predefined symbols, 5-8
prescaler, 3-64, 3-90
program counter, 3-9
program counter relative addressing mode, 6-5
program-relocatable code, 5-20, 5-26, 5-40, 5-41, 5-43, 7-2
programmable timer/event counters, 3-53
prototyping, 11-2
prototyping devices, 2-18, 2-20
Pulse flip-flop, 3-43, 3-48
PUSH, push on stack instruction, 6-13, 6-52

R

R/$\overline{W}$, 3-14, 3-26
R-C oscillator clock option, 3-33, 11-5
R0, 3-7

R1, 3-7
RAM, 3-7
realtime clock mode, 3-63
receiver, 3-68
receiver buffer, 3-78
REF, 5-50, 7-6
referencing externally defined symbols, 5-42, 5-45
register A, 3-7, 6-42, 6-61, 6-66
register B, 3-7, 6-67
register file, 3-7
register file indirect addressing mode, 6-6, 9-28
register symbols, 5-8
registers, 3-7—3-9
  write-only, 9-34
relational operators, 8-7
relocatable code, 7-2
relocatable symbols, 5-11
relocation types
  common-relocatable, 5-23, 5-26, 5-27, 5-28, 5-43
  data-relocatable, 5-26, 5-27, 5-29, 5-43
  program-relocatable, 5-20, 5-22, 5-26, 5-29, 5-30, 5-40, 5-41, 5-43
RESET, 3-11, 3-24, 3-34, 3-36, 3-39
reset, 3-39, 9-32
RETI, return from interrupt instruction, 3-49, 6-13, 6-53
RETS, return from subroutine instruction, 6-13, 6-54, 9-29
RF, 3-7
RL, rotate left instruction, 6-13, 6-55, 9-26
RLC, rotate left through carry instruction, 6-13, 6-56, 9-26
rotate instructions, 6-55, 6-56, 6-57, 6-58, 9-26
RR, rotate right instruction, 6-14, 6-57, 9-26
RRC, rotate right through carry instruction, 6-14, 6-58, 9-26
RX, 3-68, 3-78
RXBUF, 3-72
RXBUF register, 3-70, 3-78
RXD bit, 3-23, 3-70
RXEN, 3-73
RXRDY, 3-78
RXRDY bit, 3-75
RXSHF register, 3-70

S

SBB, subtract with borrow instruction, 6-14, 6-59, 9-25
SCLK, 3-23, 3-64, 3-74, 3-78
SCLKEN bit, 3-73
SCTL0 register, 3-70, 3-72
  ER, 3-75
  PRE3(1), PRE3(0), 3-77
  RXEN, 3-73
  SCLKEN, 3-73
  SPH, 3-74
  TXEN, 3-73
  UR, 3-74
SCTL1 register, 3-76
  CLK, 3-78
  SLEEP, 3-78
  START, 3-78
  T3ENB, 3-77
  T3FLG, 3-77
  WU, 3-77
SE70CP160
  key features, 2-18
  pin descriptions, 2-27
  pinouts, 2-24
SE70CP160 devices, external interrupts, 3-50
SE70CP160A, 4-47—4-50
  specifications, 4-47—4-50
SE70CP168
  key features, 2-20
  pin descriptions, 2-27
search order (macros), 8-2
Serial I/O mode, 3-68, 3-74, 3-85, 9-12
serial mode
  ASYNC, 3-71
  CHAR1, CHAR2, 3-72
  CMODE, 3-72
  MULTI, 3-71
  PEN, 3-72
  PEVEN, 3-72
  STOP, 3-72
serial port, 3-68—3-96, 9-19
  asynchronous communication mode, 3-68
  communication modes, 3-70
  hardware UART example, 9-12
  initialization, 3-89
  interrupts, 3-95
    INT4, 3-95
  isosynchronous communication mode, 3-68

multiprocessor protocols, 3-68, 3-71
serial I/O mode, 3-68
software UART example, 9-12
timing, 4-29, 4-38, 4-57
serial port communication modes, 3-83
serial port register
RXBUF, 3-70, 3-78
SCTL0, 3-70
SMODE, 3-70
SSTAT, 3-70, 3-74
T3DATA, 3-70, 3-78
TXBUF, 3-70
serial port registers, 3-69
SETC, set carry instruction, 6-14, 6-60
shifting, 9-30
sign bit, 3-8
single register addressing mode, 6-3
single-chip mode, 3-7, 3-14, 3-22
SLEEP bit, 3-78, 3-86
SMODE register, 3-70
software UART, 9-13
SOURCE, 3-63
source program, 5-1
source statement format, 5-2, 5-50
SP, 3-7
SPH bit, 3-74
SREF, 5-50, 7-6
SSTAT register, 3-70, 3-74
BRKDT, 3-76
FE, 3-76
IADD, 3-76
OE, 3-76
PE, 3-75
RXRDY, 3-75
TXE, 3-75
TXRDY, 3-75
ST, 3-8
STA, store register A instruction, 6-14, 6-61
stack, 3-7—3-8, 9-27
stack operations, 6-51, 6-52, 6-62, 9-27, 9-41
initialization, 3-8
stack pointer, 3-7, 6-43, 6-62
initialization after reset, 3-42
start bit, 3-78, 3-83
status register, 3-8, 9-24
carry bit, 3-8
global interrupt enable bit, 3-8
sign bit, 3-8
zero bit, 3-8
stop bit, 3-72, 3-83
string component (of a variable), 8-9
string mode (macro variables), 8-10
strings, 5-6, 5-7, 5-9, 8-7
single quotes, 5-7
STSP, store stack pointer instruction, 6-14, 6-62
SUB, subtract instruction, 6-14, 6-63, 9-25
subroutine instructions, 6-25, 6-54, 6-65, 9-28
subtraction instructions, 6-31, 6-32, 6-35, 6-59, 6-63, 9-25
SWAP, swap nibbles instruction, 6-14, 6-64, 9-26
symbol attribute component keywords, 8-13
symbol components (of a macro variable), 8-11
symbolic addressing, 5-49
symbols, 5-7, 5-8, 5-10
character string, 5-9
externally defined, 5-12
predefined, 5-8
relocatable, 5-11
terms, 5-8
Sync flip-flop, 3-43

T1CTL, 3-36, 3-60, 3-65
T1DATA, 3-65
T2CTL, 3-56, 3-57, 3-58, 3-59, 3-60
T2DATA, 3-65
T3, 3-68
T3DATA register, 3-70, 3-78, 3-90
T3ENB bit, 3-77
T3FLG bit, 3-77
tag characters, 5-56—5-64
TASK linker command, 7-5
terms (as symbols), 5-8
TEXT, 5-50
Timer 1, 3-14, 3-54
Timer 1 capture latch, 3-61, 3-62
Timer 1 data and control registers, 3-56
Timer 2, 3-2, 3-3, 3-6, 3-14, 3-23, 3-53
Timer 3, 3-53, 3-68, 3-70, 3-78, 3-90
timer clock, 3-64
timer interrupts, 3-66

timer output function, 3-67
TITL, 5-50
TMS7000 family devices summary, 2-1
TMS70C48, peripheral memory map, 3-20
TMS70CTx0 devices
  pin descriptions, 2-9
  pinouts, 2-7
TMS70Cx0 devices
  clock options, 3-33
  external interrupts, 3-50
  interrupts, 3-39
  key features, 2-5, 2-6
  memory map, 3-17
  peripheral memory map, 3-18
  pin descriptions, 2-8
  pinouts, 2-7
  port configuration, 3-12
TMS70Cx2 devices
  clock options, 3-37
  external interrupts, 3-50
  initialization routine, 3-41
  interrupts, 3-39
  key features, 2-10
  memory map, 3-17
  peripheral memory map, 3-19
  pin descriptions, 2-12
  pinouts, 2-11
  port A, 3-14
  timer operation, 3-65
  timer output function, 3-67
TMS70Cx8 devices
  external interrupts, 3-50
  port configuration, 3-13
TMS77C82
  pin descriptions, 2-12
  pinouts, 2-11
transmitter, 3-68
transmitter buffer, 3-79
TRAP, trap to subroutine instruction, 6-14, 6-65, 9-28
TSTA, test register A instruction, 6-15, 6-66
TSTB, test register B instruction, 6-15, 6-67
TX, 3-68, 3-79
TXBUF, 3-72
TXBUF register, 3-70
  serial port register, TXBUF, 3-79
TXD bit, 3-70
TXE bit, 3-75
TXEN bit, 3-73
TXRDY bit, 3-75
TXSHF register, 3-70

U

UART, 3-68—3-96, 9-13
UNL, 5-50
unqualified variables (in macros), 8-11
UR bit, 3-74
USART, 3-68

V

value component (of a variable), 8-9
variable components
  attribute, 8-9
  length, 8-9
  string, 8-9
  value, 8-9
variable qualifiers, 8-11
variables, 8-9
verbs, 8-18
  $ASG, 8-19
  $ELSE, 8-21
  $END, 8-22
  $ENDIF, 8-23
  $IF, 8-24
  $MACRO, 8-26
  $VAR, 8-27

W

wake-up mode, 3-36, 3-37
well-defined expressions, 5-11
write-only registers, 9-34
WU bit, 3-77, 3-87
WUT flag, 3-87

X

XCHB, exchange with register B instruction, 6-15, 6-68
XDS, ordering information, 11-19
XOR, exclusive OR instruction, 6-15, 6-69
XORP, exclusive OR peripheral register instruction, 3-78, 6-15, 6-70

XTAL oscillator clock option, 3-32, 11-5
XTAL1, 3-31, 11-5
XTAL2, 11-5
XTAL2/CLKIN, 3-31

Z

Z (zero) bit, 3-8, 9-24

TI Sales Offices

BELGIQUE/BELGIË
S.A. Texas Instruments Belgium N.V.
11, Avenue Jules Bordetlaan 11,
1140 Bruxelles/Brussel
Tel : (02) 242 30 80
Telex : 61161 TEXBEL

DANMARK
Texas Instruments A/S
Borupvang 2D,
DK-2750 Ballerup
Tel : (44) 68 74 00
Telefax : (44) 68 64 00

DEUTSCHLAND
Texas Instruments
Deutschland GmbH.
Haggertystraße 1
8050 Freising
Tel : (08161) 80-0 od. Nbst
Telex : 5 26 529 texin d
Btx : *28050#

Kurfürstendamm 195-196
1000 Berlin 15
Tel : (030) 8 82 73 65
Telex : 5 26 529 texin d

Düsseldorfer Straße 40
6236 Eschborn 1
Tel : (06196) 80 70
Telex : 5 26 529 texin d

Gildehofcenter
Hollestrasse 3
4300 Essen 1
Tel : (0201) 24 25-0
Fax : (0201) 236640
Telex : 5 26 529 texin d

Kirchhorster Straße 2
3000 Hannover 51
Tel : (0511) 64 68-0
Telex : 5 26 529 texin d

Maybachstraße II
7302 Ostfildern 2 (Nellingen)
Tel : (0711) 34 03-0
Telex : 5 26 529 texin d

ESPAÑA
Texas Instruments España S.A.
c/ Gobelas 43.
Ctra de La Coruña km. 14
La Florida
28023 Madrid
Tel : (1) 372 8051
Telex : 23634

C/Diputacion, 279-3-5
08007 Barcelona
Tel : (3) 317 91 80
Telex : 50436
Fax : (3) 301 84 61

FRANCE
Texas Instruments France
8-10 Avenue Morane Saulnier - B.P. 67
78141 Vélizy Villacoublay cedex
Tel : Standard : (1) 30 70 10 03
Service Technique : (1) 30 70 11 33
Telex : 698707 F

HOLLAND
Texas Instruments Holland B.V.
Hogehilweg 19
Postbus 12995
1100 AZ Amsterdam-Zuidoost
Tel : (020) 5602911
Telex : 12196

HUNGARY
Texas Instruments International
Budaörsi u.42, H-1112 Budapest
Tel : (1) 1 66 66 17
Fax : (1) 1 66 61 61
Telex : 2 27 676

ITALIA
Texas Instruments Italia S.p.A.
Centro Direzionale Colleoni
Palazzo Perseo - Via Paracelso, 12
20041 Agrate Brianza (Mi)
Tel : (039) 63221
Fax : (039) 632299

Via Castello della Magliana, 38
00148 Roma
Tel : (6) 657 2651
Telex : 610587 ROTEX I
Fax : (6) 657 0447

Via Amendola, 17
40100 Bologna
Tel : (051) 554004

NORGE
Texas Instruments Norge A/S
PB 106
Refstad (Sinsenveien 53)
0513 Oslo 5
Tel : (02) 155090

PORTUGAL
Texas Instruments Equipamento
Electronico (Portugal) LDA.
Ing. Frederico Ulricho, 2650
Moreira Da Maia
4470 Maia
Tel : (2) 948 1003
Telex : 22485

REPUBLIC OF IRELAND
Texas Instruments Ireland Ltd
7/8 Harcourt Street
Dublin 2
Tel : (01) 755233
Telex : 32626

SCHWEIZ/SUISSE
Texas Instruments Switzerland AG
Riedstraße 6
CH-8953 Dietikon
Tel : (01) 74 42 811
Telex : 825 260 TEXIN
Fax : (01) 74 13 357

SUOMI FINLAND
Texas Instruments OY
P.O. Box 86,
02321 Espoo
Tel : (0) 802 6517
Fax : (0) 802 6519
Telex : 121457

SVERIGE
Texas Instruments
International Trade Corporation
(Sverigefilialen)
Box 30,
S-164 93 Kista
Visit address : Isafjordsgatan 7, Kista
Tel : (08) 752 58 00
Telefax : (08) 751 97 15
Telex : 10377 SVENTEX S

UNITED KINGDOM
Texas Instruments Ltd.
Manton Lane,
Bedford,
England, MK41 7PA
Tel : (0234) 270 111
Telex : 82178

Technical Enquiry Service
Tel : (0234) 223000

TEXAS INSTRUMENTS

TI Regional Technology Centres

DEUTSCHLAND
Texas Instruments
Deutschland GmbH.
Haggertystraße 1
8050 Freising
Tel : (08161) 80 40 43

Frankfurt/Main
Düsseldorfer Straße 40
6236 Eschborn
Tel : (0 61 96) 80 74 18

Kirchhorster Straße 2
3000 Hannover 51
Tel : (0511) 64 80 21

Maybachstraße 11
7302 Ostfildern 2 (Nellingen)
Stuttgart
Tel : (0711) 34 03-0

FRANCE
Centre de Technologie
Texas Instruments France
8-10 Avenue Morane Saulnier, B.P. 67
78141 Vélizy Villacoublay cedex
Tel : Standard : (1) 30 70 10 03
Service Technique : (1) 30 70 11 33
Telex : 698707 F

Centre Européen de Développement
et Siège Social
Texas Instruments France
B. P. 5
06271 Villeneuve-Loubet cedex
Tel : 93 22 20 01
Telex : 470127 F

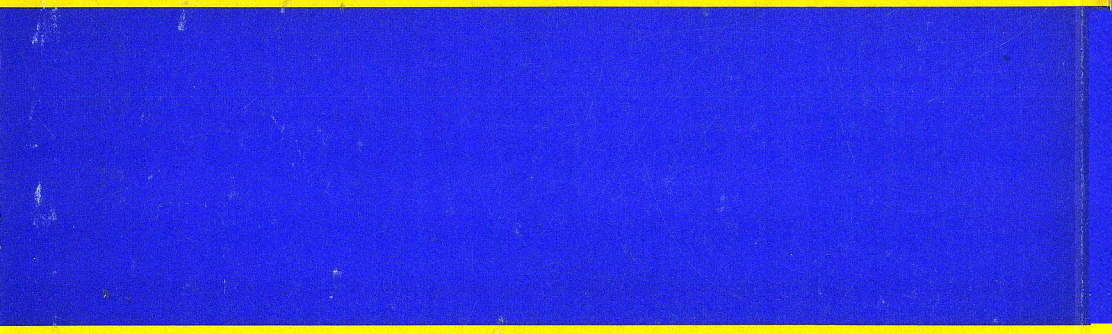
HOLLAND
Texas Instruments Holland B.V.
Hogehilweg 19
Postbus 12995
1100 AZ Amsterdam-Zuidoost
Tel : (020) 5602911
Telex : 12196

ITALIA
Texas Instruments Italia S.p.A.
Centro Direzionale Colleoni
Palazzo Perseo - Via Paracelso, 12
20041 Agrate Brianza (Mi)
Tel : 039-63221
Fax : (039) 632299

SVERIGE
Texas Instruments
International Trade Corporation
(Sverigefilialen)
Box 30
S-164 93 Kista
Isafjordsgatan 7
Tel : (08) 752 58 00
Telefax : (08) 751 97 15
Telex : 10377 SVENTEX

UNITED KINGDOM
Texas Instruments Ltd.
Regional Technology Centre
Manton Lane,
Bedford,
England, MK41 7PA
Tel : (0234) 270 111
Telex : 82178

Technical Enquiry Service
Tel : (0234) 223000



ISBN2-86886-005-2
Printed in France
01/91

DB103