1979 MICROCOMPUTER FORUM

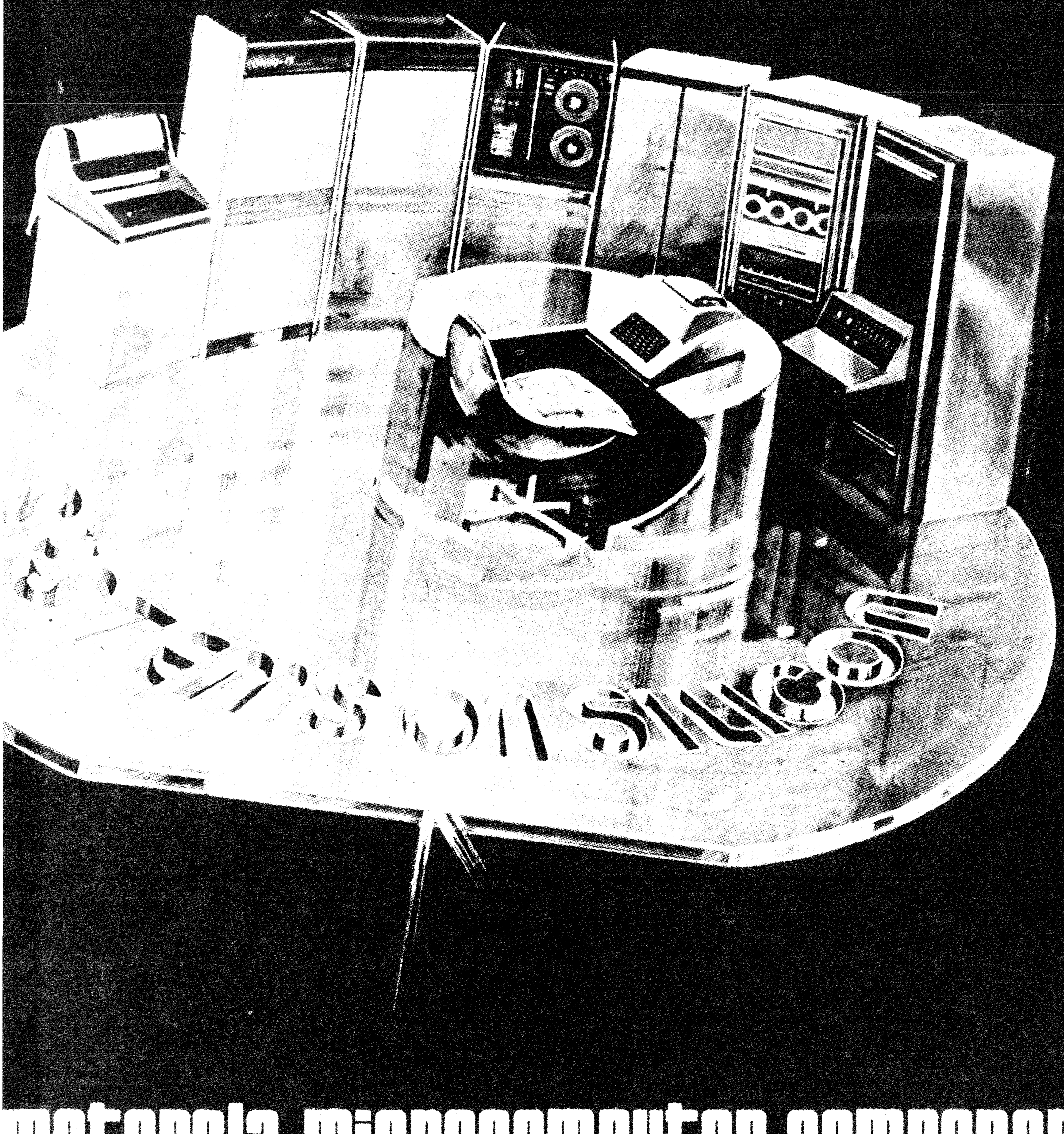
MICROCOMPUTER
FORUM
MICROCOMPUTER
FORUM
MICROCOMPUTER
FORUM

London – Stockholm – Munich – Milan – Paris – Eindhoven

MOTOROLA Semiconductors

Printed in Switzerland

TABLE OF CONTENTS



motorola microcomputer components

microcomputer components

What you should consider before you rush into —

MICROCOMPUTER DESIGN

What's so important about the microprocessor that it should have been catapulted into one of the most significant developments in the electronics field over the last decade? Well, certainly not that microprocessors, themselves, represent anything new in the way of circuit implementation. Computer designers had been using them as integral parts of large computers long before they became the buzz-word of the industry. Rather, it's the fact that technological advances into large-scale integration have made the microprocessor the most effective, most reliable, simplest, and *least expensive* way of accomplishing complex electronic functions today. It's not surprising, therefore, that the microprocessor, or in a larger sense, the microcomputer, is dominating the design thinking for most equipment currently in the planning stage.

The microcomputer is such a powerful performer that the vast majority of all possible applications probably could be served by any one of the dozens of different models available. Yet, for each application there's probably one microcomputer system that serves the purpose *best*. And so, before entrusting your design to any specific processor (or supplier) we offer the following four considerations for your investigation:

Choice

Despite occasional claims to the contrary, there's no such thing as a truly *universal* circuit—not from the standpoint of cost-effectiveness. While our M6800 microcomputer system comes as close as any other to serving the bulk of all potential applications, it's "overqualified" for some, and "underqualified" for others. That's why Motorola manufactures a variety of microcomputer systems:

The M6800 Family—the most pervasive of the general-purpose MPU systems,
The MC3870—a low-cost, single-chip microcomputer for dedicated applications,
The MC141000—a CMOS alternative to the above, for lowest possible power dissipation,
The bipolar M2900 and M10800 systems—for highest speed.

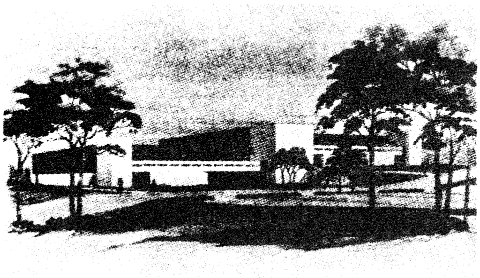
Moreover, each of these has at least one viable alternate source, so that your manufacturing requirements can not easily be compromised.

In addition to components for microcomputer systems, Motorola supplies an extensive line of micromodules—assembled subsystems—for those manufacturers who wish to begin their equipment designs at a higher level.

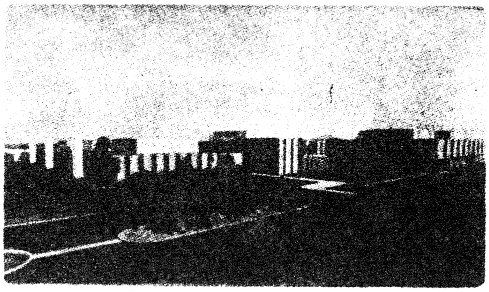
Support

The key to the successful development of a dedicated MPU system and, ultimately, to manufacture and service the system, is an umbrella of support equipment. The Motorola microcomputer product families are complemented with one of the industry's most pervasive arrays of user-oriented development aids, test equipment and support literature. The support hardware system is modularized to permit purchase of just those components required for the complexity of the system to be designed. An extensive software library, including a proven library of user-developed

MOTOROLA MICROPROCESSING MANUFACTURING FACILITIES



AUSTIN, TEXAS
Manufacturing facility for all Motorola MOS products.



MESA, ARIZONA
Bipolar processors and other MPU-related bipolar integrated circuits.

programs, is also available. And Motorola maintains a nationwide network of Field Applications Engineers to assist customers with microcomputer design problems.

Commitment

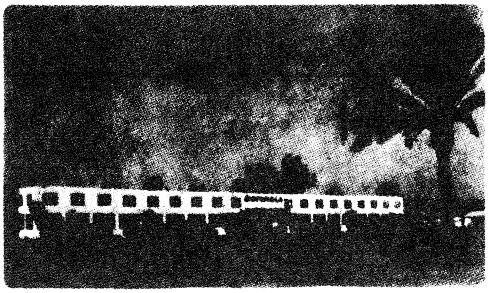
Chances are your first MPU purchase will not be your last. That's why a primary consideration in the choice of an MPU line on which to hang your designs should be the manufacturer's commitment to the expansion of that specific line—expansion of hardware to keep up with the state of the art while maintaining software compatibility with the existing system.

Motorola's commitment to the M6800 line is manifest not only by the introduction of a second-generation MPU (the MC6802), but by a literal explosion of scheduled peripheral products encompassing over 100 new type numbers in 1977 alone. All are bus-compatible with the original system and utilize the original instruction set. This dedication assures the safety of your software investment even as you switch to increasingly cost-effective hardware for future design and production.

Diversification

Even the most complex microprocessor system (or single-chip microcomputer) isn't a complete system. All need additional components of one kind or another to perform an equipment function. More memory, perhaps, for additional storage capacity; interface circuits to match various peripherals; power devices to drive external equipment. Motorola's product line extends far beyond MPU device lines. As a world leader in solid-state products, we supply devices in almost every semiconductor category—from ICs to discretes; from digital to linear; from MOS to bipolar. This proven solid-state capability is your further insurance of total-product support—today, tomorrow, and in the years ahead.

May we help you?



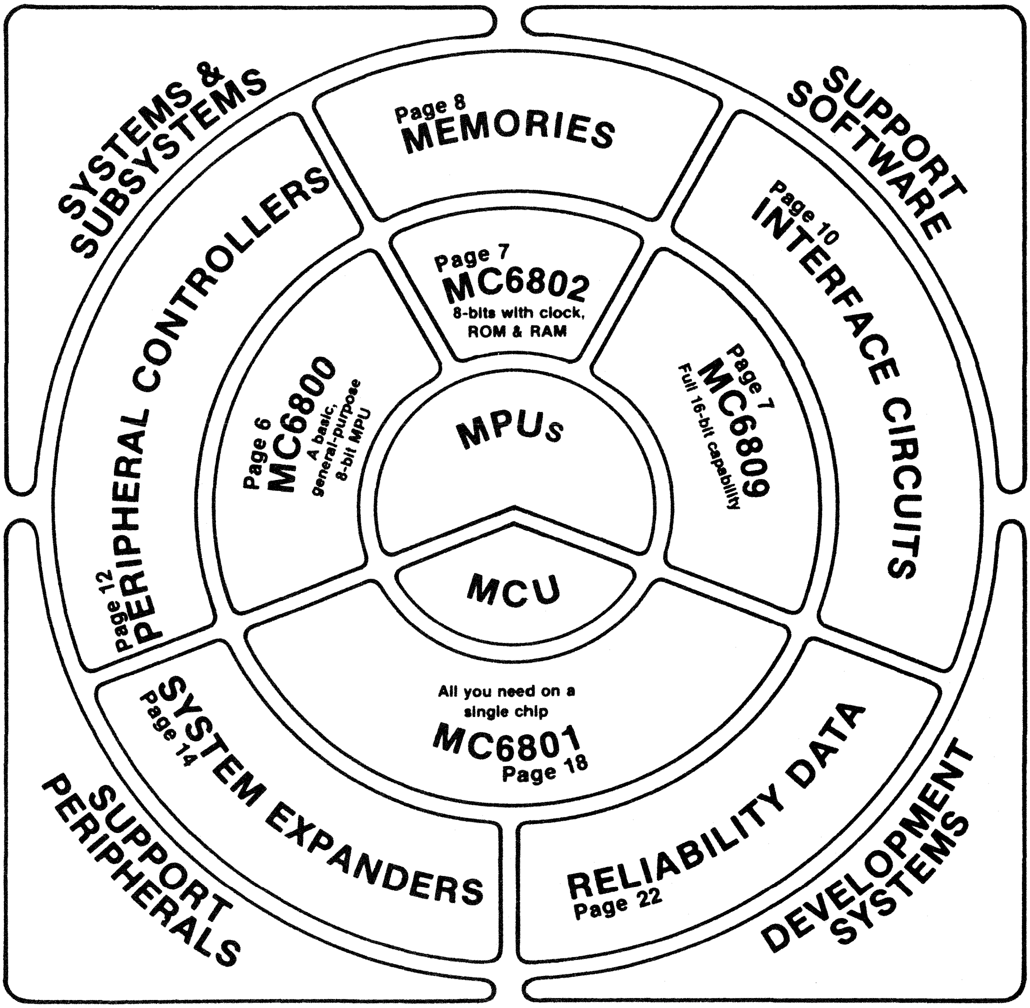
PHOENIX, ARIZONA
Microsystems and support products (56th Street Facility).

table of contents

MC3870
m2900
M6800
M10800
MC141000
NMOS • ECL • CMOS • TTL • 4-BIT • 8-BIT • 16-BIT
microcomputer components...

For a maximum versatility —

the M6800 microcomputer family . . .



at a glance . . .

Inherent in the formula for successful MPU-based designs is the selection of the most cost-effective processor family from among the many systems available today. The M6800 family ranks high in meeting the requirements.

Its NMOS LSI architecture offers bus transfer rates up to 2 MHz, and 8- and 16-bit processing capability.

Its powerful instruction set minimizes memory requirements and enhances system throughout.

The progressive complexity of its basic MPU/MCU building blocks permits system design flexibility that yields cost-effectiveness for any potential applications.

But there are more considerations to the selection of the best microcomputer system than just technical capability. When your production is as heavily dependent on the availability of a set of components as dictated by a commitment to a specific MPU family, a constant and reliable source of supply is of paramount importance. So is the continuing flow of new and related products that guards against system obsolescence.

Motorola is dedicated to the continual expansion of the M6800 system with new products, new designs and expanded peripherals—all tailored to increase the scope and value of your investment.

M6800 LINE FEATURES

For System Design Ease:

- Powerful, variable-length instructions reduce programming complexity and development time.
- 65K memory address capability encompasses the largest program requirements likely to be encountered in microcomputers.
- Single 5-volt power supply operation and bus organization simplifies system design.

For Maximum Throughput:

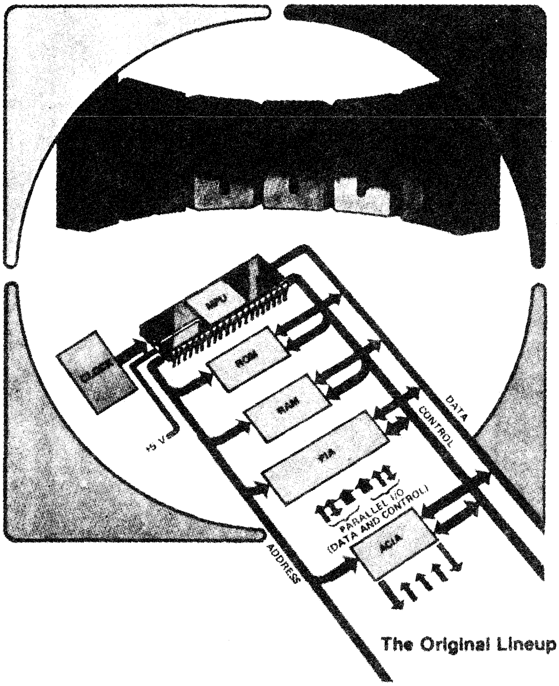
- Bus transfer rates to 2 MHz provide high-speed operation.
- Automatic data stacking during interrupts reduces programming complexity.
- 3-state output.
- Vectored restart.

M6800

the MPU

Whether a microcomputer consists of a totally integrated single chip, or is composed of a number of interactive LSI chips, *the microprocessing unit (MPU) is the central control system that determines the eventual application for which the system is best suited.* Its architecture contains the complex routines that permit the system to respond correctly to each of the different "instructions" associated with a particular system. It controls the flow of signals into and out of the computer, routing each to its proper destination in the required sequence to perform an end function.

The M6800 Family currently includes two standard 8-bit MPUs, with a third, a 16-bit unit, scheduled to join the lineup during 1978. And for maximum on-chip power, a complete single-chip microcomputer will join the Family soon (see Page 18).



The Original Lineup

THE MC6800

This microprocessor was the first of the M6800 MPU Family and still remains a highly cost-effective processor for a great many process-control and data-communications applications. Seventy-two powerful instructions and six different addressing modes give it unexcelled capability and a full range of compatible peripheral chips offer the widest possible latitude in system implementation. After years of field experience, the MC6800 has earned an enviable reputation as one of the easiest to use processors available because:

Its bus organized architecture reduces component count and simplifies interconnection;

Its single 5-V supply requirement reduces system complexity and cost;

Its 16-bit address system permits selective addressing of more than 65,000 memory locations;

Its inherent design treats each peripheral as a memory location, thereby reducing programming complexity.

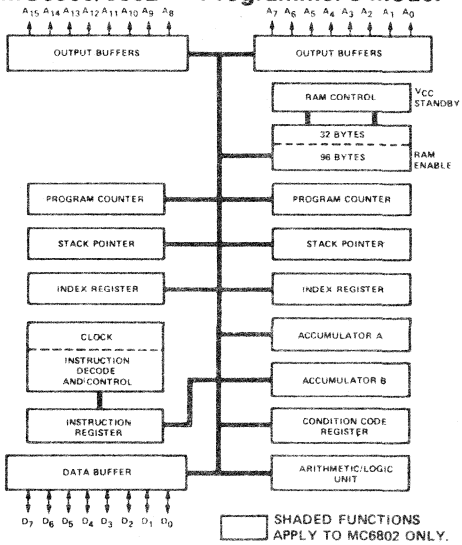
Moreover, to tailor the system to your specific needs at the lowest cost, the MC6800 (and its peripherals) is available in two different packages, three different temperature ranges and three speed ranges, as follows:

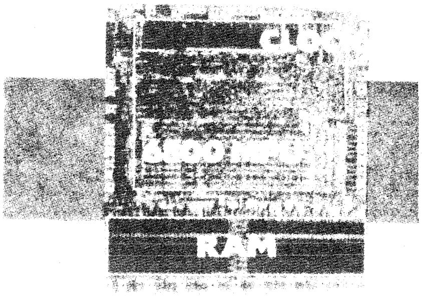
Selection

Temperature Range	Frequency Limit		
	1 MHz	1.5 MHz	2 MHz
0 to 70°C -40 to 85°C -55 to 125°C	MC6800P/L MC6800CL *MC6800CQCS	MC68A00P/L	MC68B00P/L

P suffix = Plastic Package L suffix = Ceramic Package

MC6800/6802 — Programmer's Model





MC6802

Take the basic MC6800 MPU, add an on-chip clock and 128 bytes of RAM, and you essentially have the second generation M6800 MPU chip—the MC6802. This versatile processor has all the attributes of the basic unit:

—It is fully compatible with all the peripherals, features the same MPU architecture and capabilities, and works with the same instruction set—

But it reduces the component count of a minimum microcomputer system to only *two*, compared with a minimum of four with the earlier MPU.

The built-in clock operates at a maximum frequency of 1 MHz but, thoughtfully, the chip designers have added an on-chip divide-by-four circuit to permit the use of an external 4-MHz crystal in lieu of a far more expensive 1-MHz crystal. In addition the first 32 bytes of the built-in RAM may be operated in a low-power mode, from an external power source, to prevent the loss of information during a power-down situation.

Utilizing this MPU, a minimum microcomputer system consists of:

1 MC6802 MPU
1 MC6846 ROM-I/O-Timer Unit (Page 10).

Of course, the system is expandable to any requirement with the adapters, expanders and other peripheral chips that are a part of the M6800 Family.

The MC6802 is available in both ceramic (suffix L) and plastic (suffix P) package.

INTRODUCING THE MC6809 MICROPROCESSOR

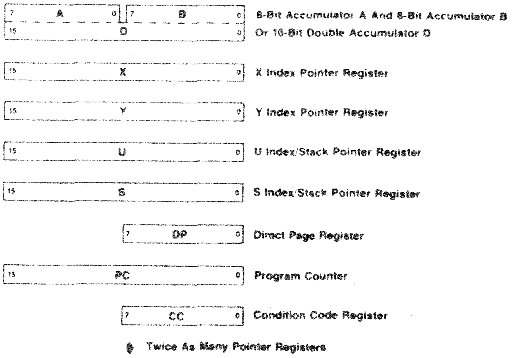
(Coming Soon)

Today, there's a lot of controversy about where a microcomputer turns into a "mini". While a number of benchmarks have been suggested, it is generally conceded that 16-bit processing capability constitutes a minimum "mini" requirement.

Motorola is a microcomputer manufacturer, but the soon-to-be-announced MC6809 Microprocessor at least borders on minicomputer capabilities.

It has 16-bit capability with 50-percent more throughput than the MC6800. It operates at 2 MHz, adds 16 new addressing modes, utilizes an expanded instruction set with high-level language capability, and features a host of other refinements that add functional expansion to, while maintaining compatibility with, the M6800 Microcomputer Components Family.

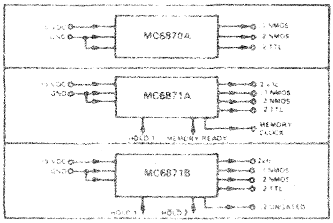
MC6809 Programming Model


7
A
0
7
B
0
15
D
0
8-Bit Accumulator A And 8-Bit Accumulator B
Or 16-Bit Double Accumulator D
15
X
0
X Index Pointer Register
15
Y
0
Y Index Pointer Register
15
U
0
U Index/Stack Pointer Register
15
S
0
S Index/Stack Pointer Register
7
DP
0
Direct Page Register
15
PC
0
Program Counter
7
CC
0
Condition Code Register
Twice As Many Pointer Registers

M6800 CLOCKS

All M6800-based systems operate with two non-overlapping clock phases, $\phi1$ and $\phi2$. A variety of clock modules is available for use with the M6800 MPU (other MPUs have built-in clock). Variations include one monolithic and three hybrid versions offering a number of system design options.

THE HYBRID

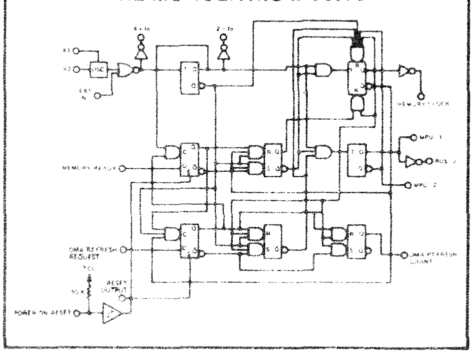


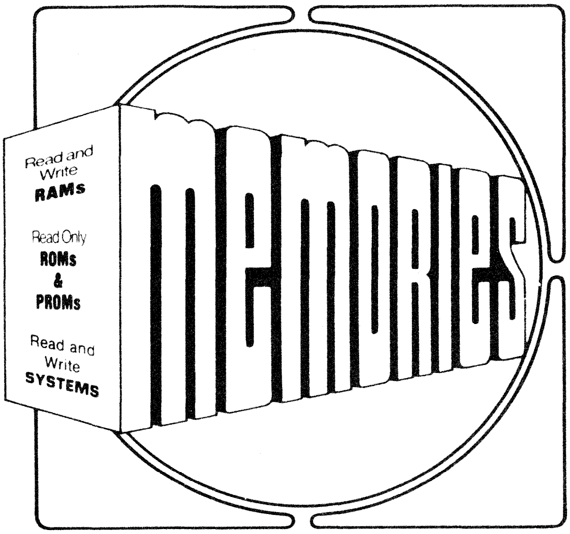
LIMITED FUNCTION
250 kHz to 2.5 MHz

FULL FUNCTION
850 kHz to 2.5 MHz

ALTERNATE FUNCTION
250 kHz to 2.5 MHz

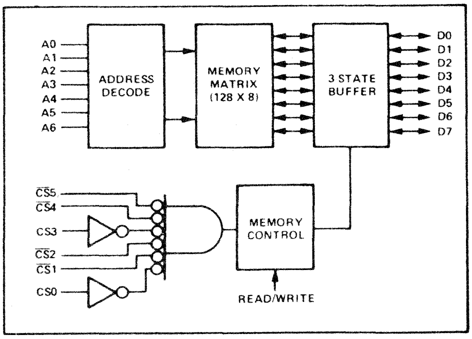
THE MONOLITHIC MC6875





Some microprocessor chips are replete with on-chip memory capacity (see Pages 16, 17, and 18); some aren't. In the interest of system flexibility, the M6800 Family of processors was designed to use external memory. Even the MC6802 processor (128 bytes of on-chip RAM) still requires external ROM and, for very complex applications, may require additional RAM as well. To meet this need, Motorola has developed a series of memories specifically structured for M6800 systems. The memories are bus organized to operate with the MPUs without any additional interface components. Most of the memories described here operate on a single 5-V power supply to obviate additional power supply complexity. However, some devices with additional voltage requirements have been included for their special attributes. These can be adapted to M6800 systems simply by supplying the additional voltages.

RAMs



128 x 8-Bit — MCM6810

This 1024-bit byte-organized memory circuit is designed to take optimum advantage of the M6800 unique features. Fabricated with high-density, N-channel, silicon-gate technology, it provides high-speed access time even at the 5-volt operating levels of the M6800 Family. Three-state, bidirectional input-output buffers interface directly with the bidirectional 8-bit data bus of the overall system. Six individual chip-select inputs, when interconnected with the systems address bus, permit extensive memory expandability and selectivity without the need of separate decoder circuits. Static operation reduces overall system complexity by eliminating refresh requirements.

AVAILABLE OPTIONS

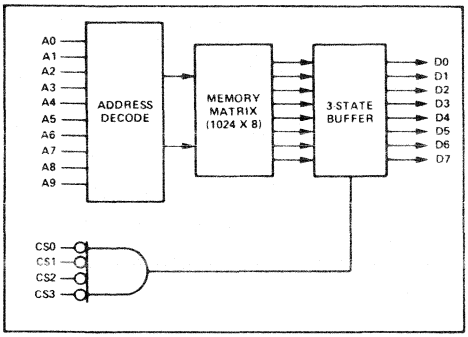
Temperature Range	Frequency Compatibility		
	1 MHz	1.5 MHz	2 MHz
0 to 70°C	MCM6810P/L	MCM68A10P/L	MCM68B10P/L
-40 to +85°C	MCM6810CP/L	MCM68A10CP/L	
-55 to +125°C	MCM6810CJCS		
MIL883B	MCM6810BJCS		
P suffix = Plastic Package		L suffix = Ceramic Package	

Additional Memory Building Blocks

The above RAM is uniquely tailored to the M6800 system by virtue of a single 5-volt supply requirement and 8-bit word organization. Motorola offers additional RAMs that can easily be adapted to expand M6800 memory capability.

Organization	Operation	Voltage Requirements	Type Number
16,384 x 1	Dynamic	+5, -5, +12	MCM4116
1,024 x 4	Static	+5	MCM2114
4,096 x 1	Dynamic	+5, -5, +12	MCM4027A
4,096 x 1	Static	+5	MCM66L41

ROMs — Mask Programmable



1024 x 8-Bit — MCM6830AL MCM68308L

This series of 8K-bit Read-Only memories houses the user program of a dedicated microprocessors. The customer defines the memory content and Motorola supplies the preprogrammed packages per the given instructions. As all other M6800 components, these devices are byte organized and operate from a single 5-volt supply. Memory expansion is provided through multiple chip-select inputs, with the user specifying the active levels of these inputs.

2048 x 8-Bit — MCM68316

Similar to above, but provides 16K-bit storage and 3 programmable chip-select inputs.

4096 x 8-Bit — MCM68332

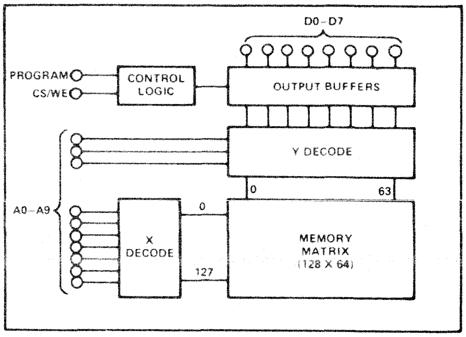
Similar to above, but provides 32K-bit storage and 2 programmable chip-select inputs.

AVAILABLE OPTIONS

Temperature Range	Frequency Compatibility		
	1 MHz	1.5 MHz	2 MHz
8K-Bit			
0 to 70°C	MCM68A30AP MCM68A308P/C	MCM68A30AP/C MCM68A308P/C	MCM68B30AP/C
-40 to +85°C	MCM68A30AC MCM68A308C	—	—
16K-Bit			
0 to 70°C	MCM68A316EP/C	MCM68A316EC	—
32K-Bit			
0 to 70°C	MCM68A332P/C	MCM68A332P/C	—

P suffix = Plastic Package C suffix = Ceramic Package

PROMs — Ultraviolet Erasable



1024 x 8-Bit — MCM68708

For system development, a user-programmable Read-Only Memory is often desirable. This 8K-bit PROM is erasable by exposure to high intensity, ultraviolet light introduced through the transparent lid of the package. Using a commercially available UV-Eraser (Turner Designs), erase time is 30 minutes. For subsequent large-volume production runs, Motorola can supply pin-for-pin compatible mask-programmable ROMs. The device is M6800 compatible, but requires additional power-supply voltage (+12 V, +5 V and -5 V).

AVAILABLE OPTIONS

Temperature Range	Frequency Compatibility		
	1 MHz	1.5 MHz	2 MHz
0 to 70°C	MCM68708L	MCM68A708L	MCM68A708L
-55 to +125°C	MCM68708CJCS	—	—

2048 x 8-Bit — TMS2716L

Similar to above, but provides 16K-bit storage.

2048 x 8-Bit — MCM2716L

Similar to above, but operates from single 5-V supply.

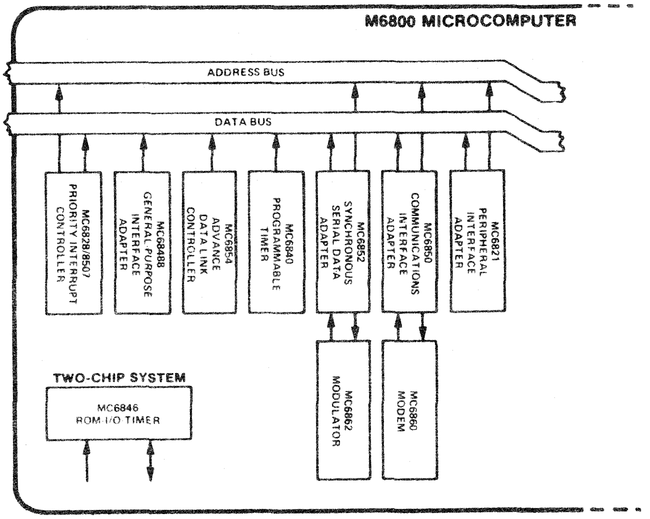
RAM — Memory Systems

To extend the storage capacity of monolithic memory blocks, Motorola has developed a series of memory modules for M6800 microcomputer systems. All of these are compatible with the EXORciser as system development tools, but can be used also with functional microcomputer system where large RAM capacity is required.

MEX6812-1 — 2048 x 8-Bit Static RAM*
MEX6815-3 — 8192 x 8-Bit Dynamic RAM**
MEX6816-1 — 16384 x 8-Bit Dynamic RAM**
MMS68102 — 16384 x 8-Bit Dynamic RAM+

*Requires 5 V power supply only
**Requires +5 V, +12 V and -12 V power supply
+Requires +5 V, -12 V power supply plus battery pack

interface circuits for **peripherals**


M6800 MICROCOMPUTER
ADDRESS BUS
DATA BUS
MC6828/8507 PRIORITY INTERRUPT CONTROLLER
MC68488 GENERAL PURPOSE INTERFACE ADAPTER
MC6854 ADVANCE DATA LINK CONTROLLER
MC6840 PROGRAMMABLE TIMER
MC6852 SYNCHRONOUS SERIAL DATA ADAPTER
MC6850 COMMUNICATIONS INTERFACE ADAPTER
MC6821 PERIPHERAL INTERFACE ADAPTER
TWO-CHIP SYSTEM
MC6846 ROM I/O TIMER
MC6862 MODULATOR
MC6860 MODEM

Simple Microcomputers, or those dedicated to one specific application, conceivably could have all the required circuitry on a single chip. General-purpose microcomputers, and those intended for complex system design, do not. The reason—the design of a chip with the memory and interface circuitry compatible with *all* possible end-use applications would make it cost-effective for *none*.

The M6800 system was conceived and designed to encompass an array of LSI components which, in various combinations, provide low-cost solutions to most eventual microcomputer applications. The choice of microprocessor chips, Page 6, together with the selection of the right memory, Page 8, and the most suitable peripheral interface circuits described here, often results in the most effective and least expensive system that can be configured for most applications.

MC6846* — ROM-I/O-Timer

Highly efficient interface chip contains 2048 bytes of ROM, together with a 16-bit programmable timer-counter and an 8-bit bidirectional data port for peripheral interface. In conjunction with MC6802 MPU, it constitutes a versatile 2-chip microcomputer system. Compatibility with other M6800 interface and peripheral circuits permits system expansion to any required additional complexity at low cost.

The built-in ROM provides read-only storage for a minimum microcomputer system and is mask-programmable to the user's specifications. The timer may be programmed to count events, measure frequencies and time intervals, generate square waves, etc. The I/O port is under software control and includes two "handshake" control lines for asynchronous interface with peripherals.

MC6821* — Peripheral Interface Adapter (PIA)

This parallel oriented periperal interface circuit is one of the most important interface circuits available. Contains two I/O circuit blocks, each capable of controlling an independent 8-bit peripheral data bus. Multiple PIAs can be used with a single system and selectively addressed by means of Chip-Select inputs. Each peripheral data line can be programmed to act as an input or output and each of four control/interrupt lines can operate in one of several control modes.

*Available in package configurations and temperature and frequency ranges to match those of the MC6800 MPU described on Page 6

MC6828/8507 — Priority Interrupt Controller (PIC)

This bipolar device is used to add prioritized responses to inputs of microprocessor systems. The performance has been optimized for the M6800 system, but will serve to eliminate input polling routines from any processor system.

With the PIC, each interrupting device is assigned a unique ROM location which contains the starting address of the appropriate service routine. After the MPU detects and responds to an interrupt, the PIC directs the MPU to the proper memory location.

MC6840* — Programmable Timer

This component is designed to provide variable system time intervals. It has three 16-bit binary counters, three corresponding control registers and a status register. The counters are under software control and may be used to cause system interrupts and/or generate out-put signals. The MC6840 may be utilized for frequency measurements, event counting, interval measuring and similar tasks.

MC68488* — General-Purpose Interface Adapter

The MC68488 GPIA interfaces between the IEEE488 standard instrument bus and the M6800 System. With it, many instruments may be interconnected and remotely and automatically controlled or programmed. Data may be taken from, sent to, or transferred between instruments.

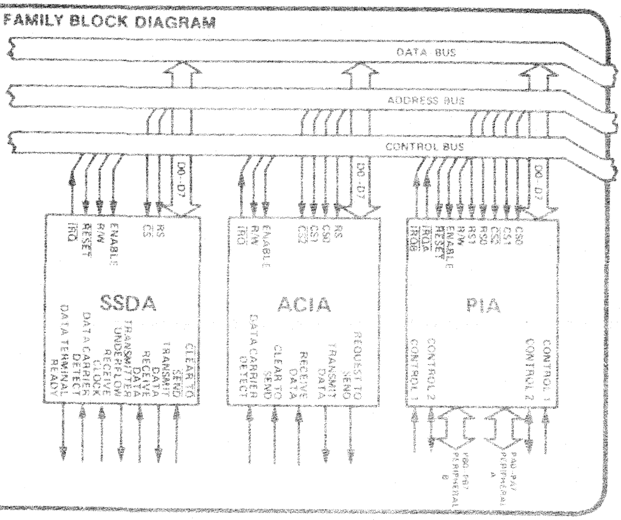
The MC68488 will automatically handle all handshake protocol needed on the instrument bus.

communications and instrumentation

In many applications, input data to a computer comes from program sources that are wired directly to the computer inputs; in others the data is derived from remotely located sources and transmitted to the computer by means of telephone lines. Remote data communications requires additional peripheral equipment—to establish contact; to convert digital signal levels into corresponding transmittable data; to assemble serially transmitted data pulses into byte-sized parallel inputs to the computer (or vice-versa).

The M6800 Family contains a number of compatible LSI components that make the development of communications interface equipment quick, easy and relatively inexpensive.

FAMILY BLOCK DIAGRAM



MC6860* — 0-600 bps Digital Modem

The MC6860 is a MOS subsystem designed to be integrated into a wide range of equipment utilizing serial data communications, including stand-alone modems, data-storage devices, remote-data communications terminals and I/O interfaces for minicomputers.

The modem provides the necessary modulation, demodulation and supervisory control functions to implement a serial data communications link, over a voice grade channel, utilizing frequency shift keying (FSK) at bit rates up to 600 bps.

The modem is compatible with the M6800 Microcomputer Family, interfacing directly with the Asynchronous Communications Interface Adapter to provide low-speed data communications capability.

MC6850* — Asynchronous Communications Interface Adapter (ACIA)

This circuit provides the data formatting and control to interface serial asynchronous data communications information to bus-organized systems.

The parallel data of the bus system is serially transmitted and received by the asynchronous data interface, with proper formatting and error checking. The functional configuration of the ACIA is programmed via the data bus during system initialization. A programmable Control Register provides variable word lengths, clock division ratios, transmit control, receive control, and interrupt control. Three control lines allow the ACIA to interface directly with the MC6860 Digital Modem.

MC6854* — Advanced Data Link Controller (ADLC)

The MC6854 ADLC performs the complex MPU/data communication link function for the "Advanced Data Communication Control Procedure" (ADCCP), High-Level Data Link Control (HLDC) and Synchronous Data Link Control (SDLC) standards.

In a bit-oriented data communication system the data is transmitted and received in a synchronous serial form.

The serial data stream must be converted into parallel, analyzed, and stored (for use by the MPU) in order for data link management to be accomplished. Similarly, parallel data from the MPU system must be serialized with the appropriate frame control information in order to conform to the bit-oriented protocol standards. The Advanced Data Link Controller (ADLC) provides these functions.

MC6852* — Synchronous Serial Data Adapter (SSDA)

Provides interface between the M6800 MPU system and synchronous data terminals such as floppy disk equipment, cassette or cartridge tape controllers, numerical control systems and other systems requiring movement of data blocks. Operates at speeds up to 600 kbps.

MC6862 — Digital Modulator

Offers the necessary modulation and control functions to implement a serial data communications link over voice-grade channels at bit rates of 1200 and 2400 bps.

*Available in package configurations and temperature and frequency ranges to match those of the MC6800 MPU described on Page 6.

New M6800

LSI peripheral controllers

On the one hand there are the microcomputers, and on the other there are the peripherals. Each peripheral has different needs, both functionally and electrically, and, therefore, demands a different interface circuit to adapt it to a specific microcomputer design.

The MC6821 Peripheral Interface Adapter on Page 10 permits first-order peripheral selection and I/O control, but it doesn't provide the complex functional control required by each unique computer peripheral.

Normally, functional peripheral control requires a board-full of SSI/MSI circuits. The LSI circuits described here reduce this requirement to a simple, convenient and relatively inexpensive single package.

MC6843 — Floppy Disk Controller

This 40-pin LSI circuit performs the complex MPU/Floppy Disk interface function. It contains twelve accessible and three nonaccessible internal registers which, together with a Micro-Controller/ROM network, form the communications link between the M6800 MPUs and a wide range of disk drives. Multiple disk drives can be controlled with the addition of external multiplexing rather than additional controllers.

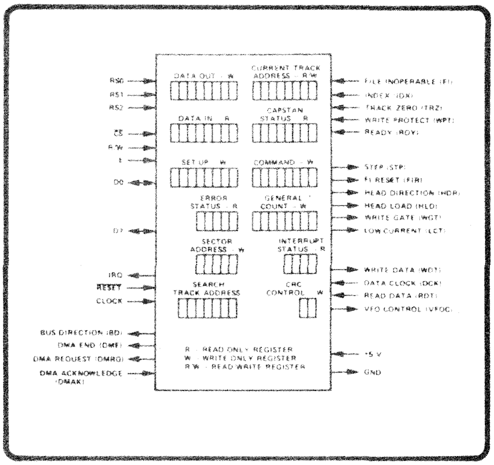
General Description

The three nonaccessible registers provide serial-to-parallel and parallel-to-serial conversions as well as Data-Clock pattern generation and detection. The disk operation is monitored by the MPU via the three status registers. Separate registers provide for Track and Section Address Information.

The Setup Register serves two purposes. One section allows generation of a programmable delay corresponding to the Seek Time of the drive in use. The remaining section provides a programmable settling time delay.

The General Count Register provides the new track number for SEK and STZ commands, and a second count for multi-sector read/write.

One bit in the Command Register selects either Program Control or Direct Memory Access. The remaining bits in the Command Register direct the internal Micro-Control Unit to perform either a micro or macro command. A set of 10 macro commands govern program operation.



Programming Model of MC6843 Floppy Disk Controller

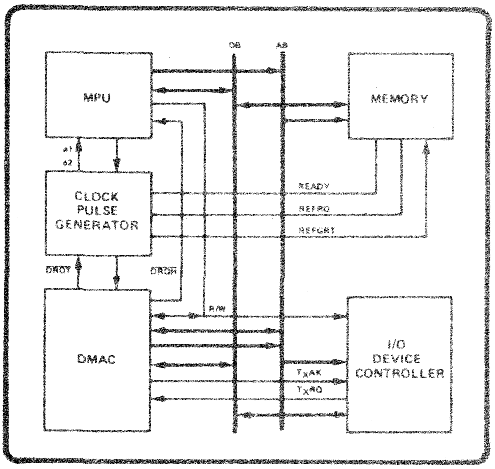
MC6844 — Direct Memory Access Controller

This DMAC works with an M6800 MPU Clock Pulse Generator and an I/O Peripheral Controller, such as the units described here, to facilitate direct access to the computer memory by the peripheral, thus bypassing MPU interactive time delay.

General Description

The MC6844 is operable in three modes: HALT Burst, Cycle Steal and TSC Steal. In the Burst Mode, the MPU is halted by the first transfer request (TxRQ) input and is restarted when the Byte Count Register (BCR) is zero. Each data transfer is synchronized by a pulse input of TxRQ. In the Cycle Steal Mode, the MPU is halted by each TxRQ and is restarted after each one byte of data transferred. In the TSC Steal Mode, DMAC uses the three-state control function of the MPU to control the system bus. One byte of data is transferred during each DMA cycle.

The DMAC has four channels. A Priority Control Register determines which of the channels is enabled. While data is being transferred on one channel, the other channels are inhibited. When one channel completes transferring, the next will become valid for DMA transfer. *The PCR also utilizes a Rotate Control bit.* Priority of DMA transfer is normally fixed in sequential order. The highest priority is in #0 Channel and the lowest is in #3. When this bit is in high level, channel priority is rotated such that the just-serviced channel has the lowest priority in the next DMA transfer.



Typical Direct Memory Access Diagram

MC6845 — CRT Controller

This single-chip Controller provides the complex interface between a cathode-ray terminal and a microprocessor of the M6800 Family. It is designed to simplify the development and production of equipment such as intelligent terminals, word processing and information display devices.

General Description

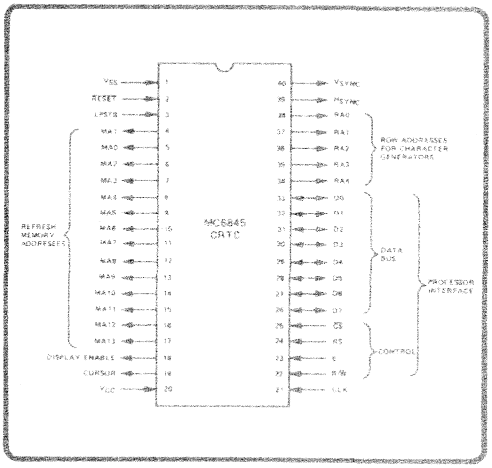
The CRTC consists of the horizontal and vertical counting circuits, a display address generator, a cursor register and comparator, and a light-pen register.

The horizontal and vertical counting circuits generate the signals: Blank, HSYNC, VSYNC, and R0–R4. R0–R4 are row count signals to the external character generator ROM. The numbers of horizontal characters per raster, rasters per character line, character lines per screen and horizontal and vertical SYNC position are programmable by the MPU.

With 14 address lines from the CRTC to the display memory, over 16K of memory may be randomly addressed for display. The CRT may be scrolled or paged through the entire display memory under MPU control.

A light pen strobe input signal allows capture of refresh address in an internal light-pen register.

The cursor control register determines the cursor location on the screen. The cursor format can be programmed for fast-blink, slow-blink, or non-blink appearance, with programmable size.

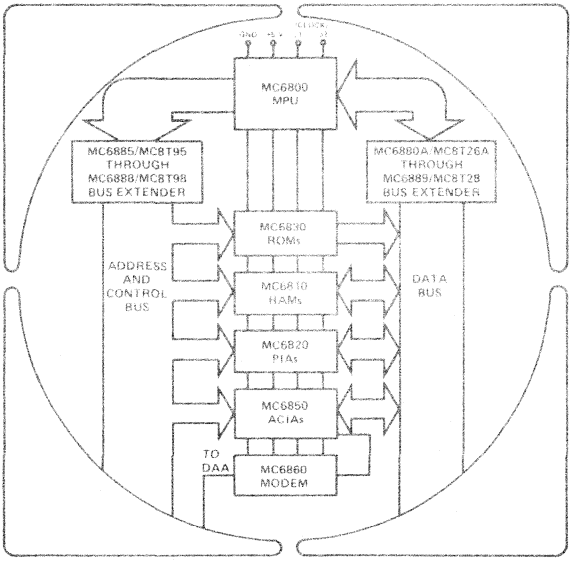


Pin Assignment for MC6845

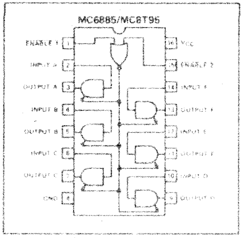
Auxiliary M6800 bipolar circuits

system expanders

The M6800 microprocessor units are capable of directly driving up to eight peripheral components (memories, interface modules, etc.) and one TTL load. A great number of systems can be configured within the framework of these limits. It is conceivable, however, that some very complex systems may require more ancillary circuits than the MPUs can safely accommodate. For systems requiring greater load capacity, and to increase the versatility of the system in general, the monolithic circuits described here have been designed. These circuits are manufactured with bipolar technology, but their operating characteristics are designed for compatibility with other (NMOS) M6800 components.

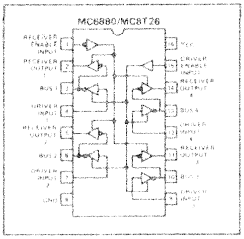


Hex 3-State Buffer/Inverters
MC6885 to MC6888
(MC8T95 to MC8T98)



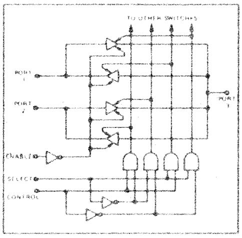
Designed as bus extenders for unidirectional bus systems such as the M6800 Address and Control bus functions. The various devices in this product sequence differ in output (inverting and noninverting) and enable configurations. Schottky technology assures high-speed operation (8 ns typ). High impedance inputs prevent loading of bus system for output requirements up to 40 mA.

Quad Bus Transceiver
MC6880A (MC8T26A) — Inverting
MC6889 (MC8T28) — Noninverting



This component consists of four separate receiver-transmitter combinations, designed for use with a *bidirectional* bus system such as the M6800 data bus. Driver and receiver output currents are 48 mA and 10 mA, respectively. Maximum input current requirement of 200µA at any input pin ensures proper operation with MC6800 MPU. Employs Schottky technology for high-speed operation.

Three-Channel Bidirectional Bus Switch
MC6881



Permits bidirectional exchange of TTL-level signals between multiple microprocessors and the data bus, or the multiplexing of signals to a single processor. Greatly expands versatility of microcomputer systems. Fully compatible with M6800 Family components.

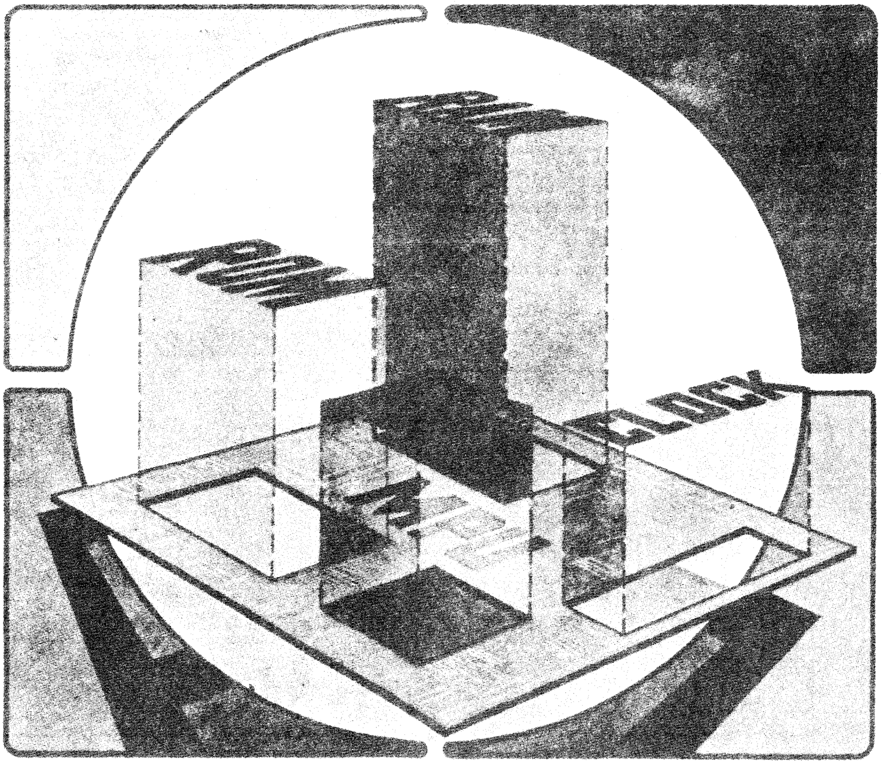
all-on-one-chip microcomputers

Utilization of large-scale integrated circuits always involves a series of compromises. On the one hand, the more circuitry that is incorporated on a single chip, the more limited is the system designer's control over the exact behavior of the ultimate system. On the other hand, the more complex the chip, the fewer the components needed for system implementation and, concurrently, the simpler the design and the lower the cost.

Applied to microcomputers, the previously described M6800 Family of LSI components gives the designer control over system architecture and functional operation while, at the same time, offering large enough building blocks for simplified system designs at low cost. Yet, there are numerous microcomputer applications where the end requirement permits processing of all necessary functions—MPU, Memory, I/O—into a single chip of managable size. For such applications the single-chip microcomputer results in the most cost-effective final system.

You don't just order a single-chip microcomputer; you have it built for you. Since your dedicated program must be incorporated in the read-only memory on the chip, your order constitutes, in effect, a custom order. Yet, because custom programming can be done in the final metallization stage of chip processing, such processing is relatively inexpensive.

Motorola now supplies two single-chip microcomputers of varying design with still another in the design cycle. These components, described on the following pages, merit serious consideration.



For dedicated applications . . .

single chip microcomputers

Any application that can be formatted within the capacity of an on-chip ROM is a likely candidate for one of the single-chip microcomputers described here. The customer develops and tests his proprietary source program and sends it to Motorola for proper processing of the final chip. From receipt of the source program to delivery of prototype product takes approximately 8 weeks.

The MC3870 8-Bit MOS

Take a powerful Arithmetic Logic Unit (ALU);
Plus 2048 bytes of Read-Only Memory (ROM) and 64 bytes of scratchpad RAM;
Add four ports of TTL-compatible Input/Output; a programmable binary timer capable of operating in the Interval Timer mode, the Pulse-Width Measurement mode and the Event Counter Mode; a built-in clock with internal or external timing capability;
Make it completely compatible with the extensive software library of the popular F8 microcomputer . . .
And *you've got as versatile a single-chip microcomputer as modern technology permits.*

The Motorola MC3870 is a complete 8-bit MOS microcomputer utilizing ion-implanted, N-channel, silicon-gate technology and offers maximum cost-effectiveness for a wide range of control and logic-replacement applications. It is simple to implement (requiring only a single +5-volt ±10% power supply) and power saving in operation (requires only 275 mW, typical). Seventy-six instructions, the entire instruction set of the F8 multi-chip family, impart to the MC3870 a high degree of functional versatility.

Functional Pin Description

P0-0 to P0-7, P1-0 to P1-7, P4-0 to P4-7, and P5-0 to P5-7 are 32 lines which can be individually used as either TTL-compatible inputs or as latched outputs.

STROBE is a ready strobe associated with I/O Port 4. This pin, which is normally high, provides a single low pulse after valid data is presented on the P4-0 to P4-7 pins during an output instruction.

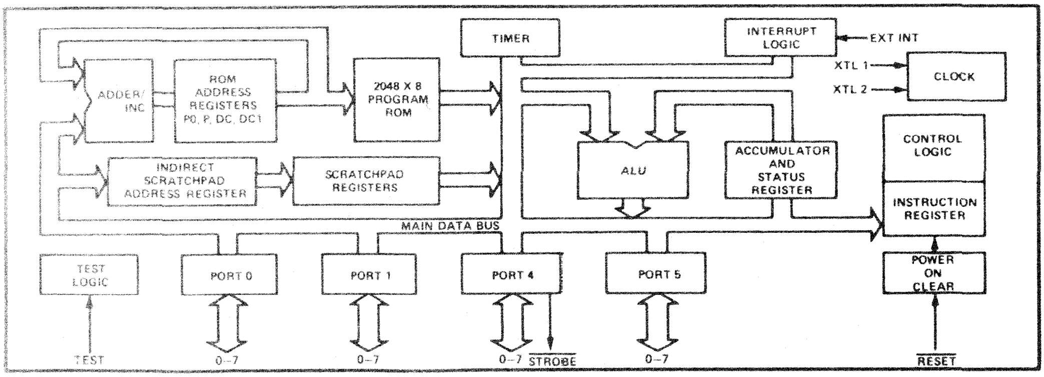
RESET may be used to externally reset the 3870. When pulled low, the 3870 will reset. When then allowed to go high, the 3870 will begin program execution at program location H "000".

EXT INT is the external interrupt input. Its active state is software programmable. This input is also used in conjunction with the timer for pulse-width measurement and event counting.

XTL1 and XTL2 are the time base inputs to which a crystal (1 to 4 MHz), LC network, RC network, or an external single-phase clock may be connected. If timing is not critical, the 3870 will operate from its internal oscillator with no external components.

TEST is an input, used only in testing the 3870. For normal circuit functionality this pin is left unconnected or may be grounded.

V_{CC} is the power supply input (+5 ± 10%).



with Mask- Programmable Read-Only Memory (ROM)

MC141000/1200 Family

Somewhat less sophisticated than the MC3870, this *4-bit* single-chip microcomputer, nevertheless, is more than adequate for a wide range of applications . . . and, it offers some unique advantages. It features CMOS circuitry, providing the lowest possible power consumption, and making it suitable for battery powered, battery back-up, or conventional 5 V operation.

Forty-three basic instructions handle I/O, constant data from ROM, bit control, internal data transfer, arithmetic processing, logic comparison, conditional and nonconditional branching, and subroutines. A 1024 x 8-bit ROM and a 64 x 4-bit RAM handle the on-chip memory requirements.

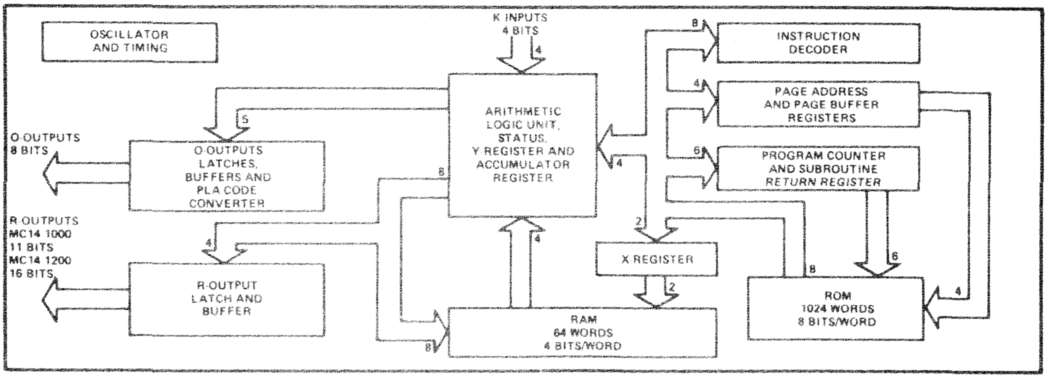
The MC141000/1200 Family is source-program compatible, pin-out compatible and architecturally similar to the well-known PMOS TMS1000 Family, but with the following additional features:

- Power Consumption — only 2.5 mW at 5 V
  only 500 μW at 3 V
- Fully Static Operation
- TTL-Compatible — Drives One TTL Load or Four LSTTL Loads
- Clock Frequency to 700 kHz at VDD = 5 V
- 16 "R" Outputs (MC141200)

Applications

- Appliance Controllers
- Calculators
- Toys
- Radio Controllers
- Communications Controllers
- Data Terminals
- Cash Registers
- Heating/Air-Conditioning Controllers
- Remote Sensing System
- Printing Controllers
- Security Systems
- Power Systems Control
- Automotive Control

The above applications of the MC141000 Family demonstrate its wide potential. Motorola will accept customer programs or will contract complete program development, given the specifications for the application. Customer hardware and software support is already available for developing programs and debugging systems. This consists of one board and a software package using the M6800 EXORciser. Contact your local sales office for status and availability of support equipment.



OSCILLATOR AND TIMING
K INPUTS 4 BITS
4
INSTRUCTION DECODER
8
PAGE ADDRESS AND PAGE BUFFER REGISTERS
4
ARITHMETIC LOGIC UNIT, STATUS, Y REGISTER AND ACCUMULATOR REGISTER
5
O OUTPUTS 8 BITS
O OUTPUTS LATCHES, BUFFERS AND PLA CODE CONVERTER
8
4
PROGRAM COUNTER AND SUBROUTINE RETURN REGISTER
6
R OUTPUTS MC14 1000 11 BITS MC14 1200 16 BITS
4
R-OUTPUT LATCH AND BUFFER
4
2
X REGISTER
2
8
6
ROM 1024 WORDS 8 BITS/WORD
4
RAM 64 WORDS 4 BITS/WORD
8

Single-chip microcomputers . . . .

looking ahead- the MC6801

The first M6800 Microcomputer on a single chip

With expected availability before the end of '78, the MC6801 single-chip microcomputer merits serious attention for the next generation of equipment now being designed.

Why?

Well, for starters, here are a few of the more important reasons:

It is characterized with all the circuitry basic to the M6800 MPU . . .

plus

The on-chip clock oscillator and 128 x 8-bit RAM of the MC6802 . . .

plus

A 2K x 8-bit ROM . . .

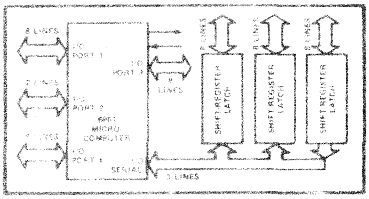
A 16-bit timer . . .

A vast expansion of I/O capacity.

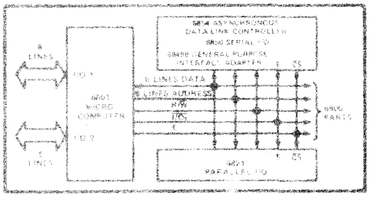
There's not much left to chance in the architecture of the MC6801. There are 31 programmable parallel I/O lines for managing external peripherals, and a serial I/O port for controlling communications equipment. A powerful interrupt capability, with 8 interrupt levels, cuts design costs and boosts performance.

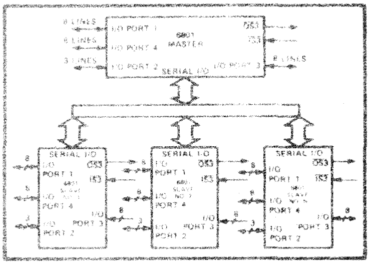
And, just in case the built-in capacity of the MC6801 is *insufficient, for a specific application,* it retains complete compatibility with the rest of the M6800 line. This means memory expandability up to 65K bytes and a wide variety of other functional options that makes an MC6801 system one of the most powerful in the industry.

Low cost Data Communications with simple serial I/O peripherals, such as shift registers is facilitated by an on-chip Serial I/O port.

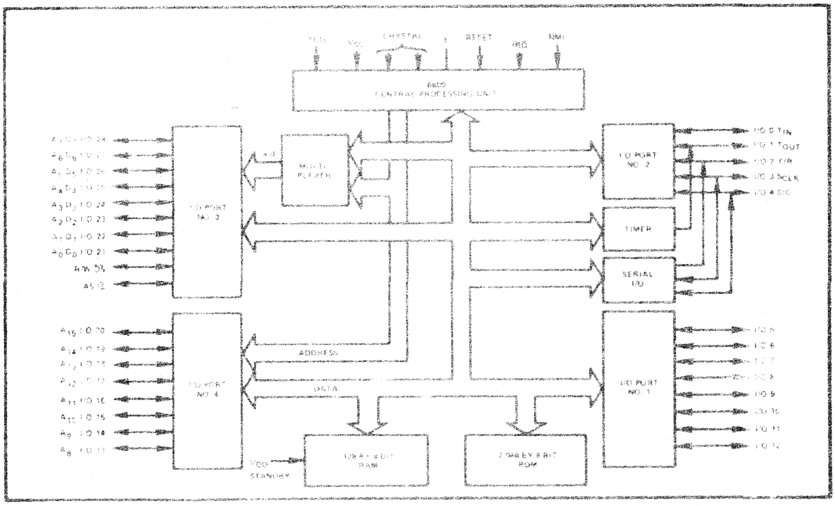


Expanding parallel I/O capacity is simplified by use of I/O ports to expand the chips data bus and address bus to external M6800-oriented peripherals.





Using the Serial I/O port, a number of MC6801s can be arranged in a master-slave multiprocessor setup.



Profile of an advanced single-chip micro-computer— The Motorola MC6801

Where high speed counts . . .

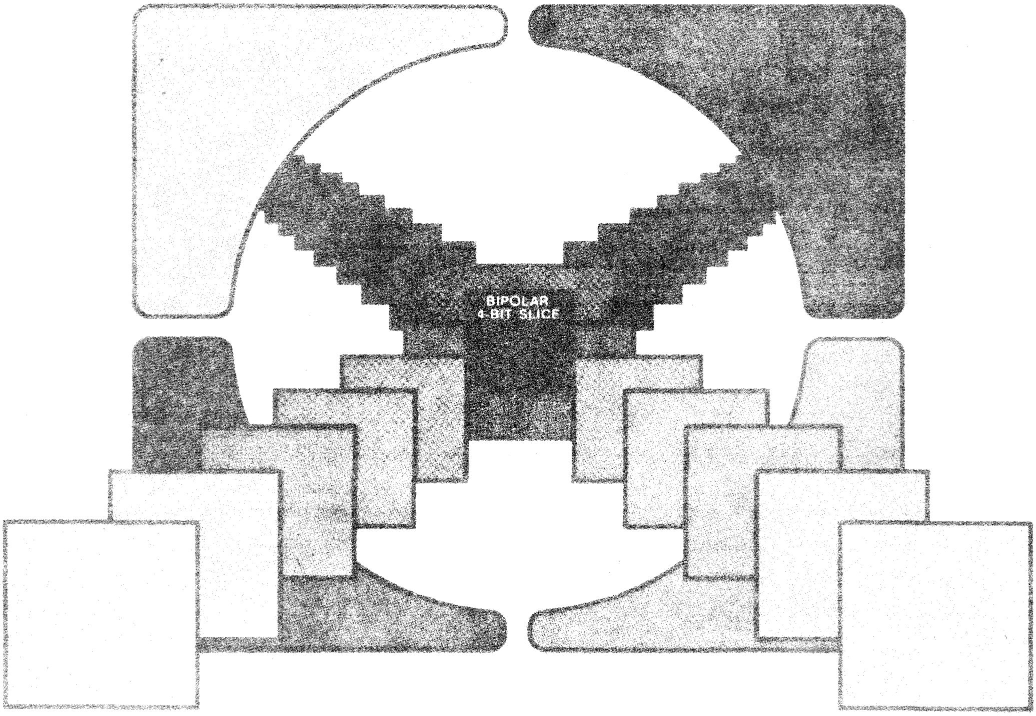
bipolar 4 bit-slice processors

Long before MOS technology became the unofficial standard for microcomputers, computers were being built with bipolar building blocks . . . and they *still* are. While suffering in comparison with MOS circuits in terms of processing simplicity and, therefore, cost (for LSI configurations), they have an overriding advantage in terms of speed. In many applications requiring real-time responses and complex calculations, bipolar processing represents the only alternative.

But even here, Motorola offers the system dedesigner a choice—a choice between two bipolar circuit configurations: TTL (Schottky, of course) and ECL (Pages 20 and 21).

The bipolar approach to microcomputers differs considerably from the MOS approach. Rather than providing complete microcomputers, or even microprocessors, bipolar designers have evolved a "bit slice" concept. This consists of a series of LSI components representing various major functions of a general-purpose computer. These components are cascadable to form systems of any desired complexity. Thus, the basic 4-bit slice building blocks offered by the two Motorola bipolar families can be expanded into 8-bit, 16-bit and even 32-bit machines. Special circuit architecture makes these building blocks easily microprogrammable.

Unlike MOS MPUs, all of which are accompanied by proprietary software, the bipolar families can be designed to utilize software from any existing computer line.



Where high speed counts . . .

bipolar 4 bit slice processors

Speed and versatility are the key attributes of the bipolar 4-bit slice processor families when compared with the MOS components. Speed is the biproduct of bipolar processing; versatility results from the "slice" concept that permits virtually unlimited expansion of the computer system. *Specifically, both families described here consist of 4-bit-wide components that are* structured or "sliced" parallel to the data flow. This permits system expansion to larger word lengths simply by connecting several parts (of each type) in parallel.

The M2900 (LSTTL) Family

System Clock Frequency 8.3 to 9.5 MHz

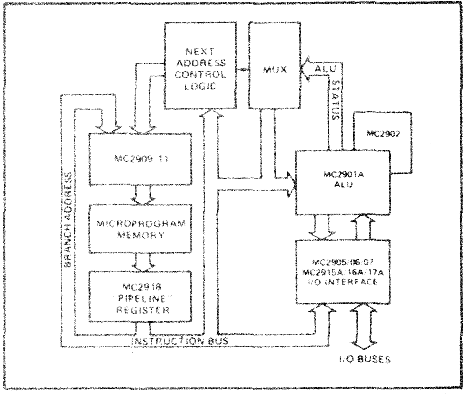
This family of TTL LSI components is microprogrammable for efficient emulation of almost any computing machine.

The heart of the system is the MC2901, a fully expandable 4-bit Arithmetic and Logic Unit (ALU). This device consists of a 16-word by 4-bit two-port RAM, a high-speed ALU, and the associated shifting, decoding and multiplexing circuitry. The ALU function has look-ahead or ripple carry, three-state outputs, and various status-flag outputs. The look-ahead carry function is performed with an MC2902 Look-Ahead Carry Generator in conjunction with the ALU.

The MC2909/2911 are four-bit wide address controllers intended for sequencing through a series of micro instructions contained in the microprogram memory. These controllers have a 4 x 4 stack that allows nesting of subroutines. The system speed can be improved by "pipelining" the contents of the microprogram into MC2918 four-bit registers. Also, the MC2918 register can be used as an address register, condition code register, or for various other register applications.

The I/O interface can be achieved with several different bus transceiver devices. The MC2905/06/07 have high-current sinking, open-collector bus outputs. The driver side has four D-type edge-triggered flip-flops and the receiver side has four D-type latches. The MC2915A/16A/17A are three-state bus output options. These bus transceivers can be used to transfer information from the ALU to the main memory or other bus applications.

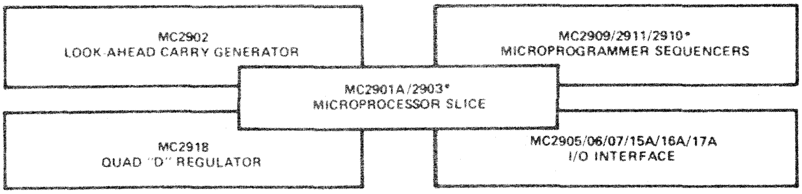
FASTER



Example of basic system implementation with dedicated M2900 components

COMPATIBLE TTL MEMORIES

	Size (Organization)	Device No.	Access Time (ns max)	P_D (mW typ/pkg)	Temperature (°C)
RAMs	1K Bits (1024 x 1)	MCM93415 MCM93425	35	550	0 to +70
ROMs	512 Bits (64 x 8)	MCM5003 MCM5303	75	500	0 to +70 -55 to +125
		MCM5004 MCM5304	75	600	0 to +70 -55 to +125
	4K Bits (512 x 8)	MCM7640 MCM7641	40 typ	500	0 to +70
	4K Bits (1024 x 4)	MCM7642 MCM7643			



*AVAILABLE SOON

Complement of an expanding line of LSI/MSI components designed especially for M2900 system design

The M10800 (ECL) Family

System Clock Frequency 10 to 15 MHz

Offering the fastest cycle times of any available bit-slice processor family, the M10800 series of ECL 4-bit processor slices permits the design of high-speed computer systems.

The core of any M10800-based system is the Arithmetic and Logic Unit (ALU). It operates at system-clock frequencies of 10 to 15 MHz, which represent cycle times of 60 to 100 ns. System word size starts at the ALU width of 4 bits, but can be expanded to n × 4 bits by cascading ALU sections. To support the ALU, Motorola has developed several ECL circuits that take care of most of the housekeeping without restricting the processor design.

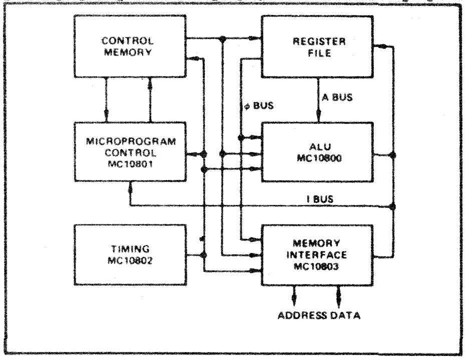
Intended to address the instructions stored in the microprogram memory, the MC10801 Microprogram Controller provides a 4-bit address that can be expanded to any size by cascading controllers. A memory interface unit, the MC10803, also has a cascadable 4-bit output bus, but it connects to the address bus of the main memory and supplies all the read and write addresses.

Acting as a register file, stack or I/O buffer, the MC10806 dual-port memory provides 32 words × 9 bits of temporary storage and can be accessed through either of its ports. For high-speed mathematical operations, the MC10808 Multibit Shifter can handle up to 16 bits and, under software control, can do left-shift, right-shift or rotate operations. Additional MC10808s can be cascaded for larger word lengths.

Other support circuits include the MC10802 Timing Generator and Clock Controller, the MC10804 and MC10805 Bidirectional Bus Translators (ECL-to-TTL and vice-versa) and all of the MECL 10,000 series of logic circuits.

MECL is a trademark of Motorola Inc.

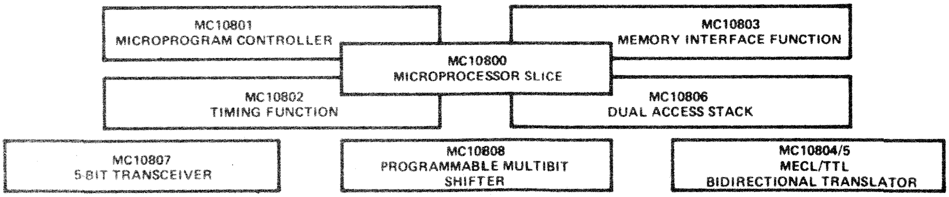
FASTEST



One example of high performance system implementation using basic M10800 Building Blocks.

COMPATIBLE ECL MEMORIES

	Size (Organization)	Device No.	Access Time (ns max)	PD (mW typ/pkg)	Temperature (°C)
RAMs	1K Bits (1024 x 1)	MCM10146	29	500	0 to +75
	256 Bits (256 x 1)	MCM10144	26	420	
		MCM10152	15	500	
	128 Bits (128 x 1)	MCM10147	12	420	-30 to +85
	64 Bits (16 x 4)	MCM10145	15	625	0 to +75
	16 Bits (8 x 2)	MCM10143	14.5	610	-30 to +85
ROMs	256 Bits (32 x 8)	MCM10139A	10 typ	500	0 to +70
	1K Bits (256 x 4)	MCM10149	25	540	-30 to +85


MC10801 MICROPROGRAM CONTROLLER
MC10803 MEMORY INTERFACE FUNCTION
MC10800 MICROPROCESSOR SLICE
MC10802 TIMING FUNCTION
MC10806 DUAL ACCESS STACK
MC10807 5-BIT TRANSCEIVER
MC10808 PROGRAMMABLE MULTIBIT SHIFTER
MC10804/5 MECL/TTL BIDIRECTIONAL TRANSLATOR

Present complement of an expanding line of LSI/MSI components designed especially for M10800 system design.

Commitment to . . .

component reliability

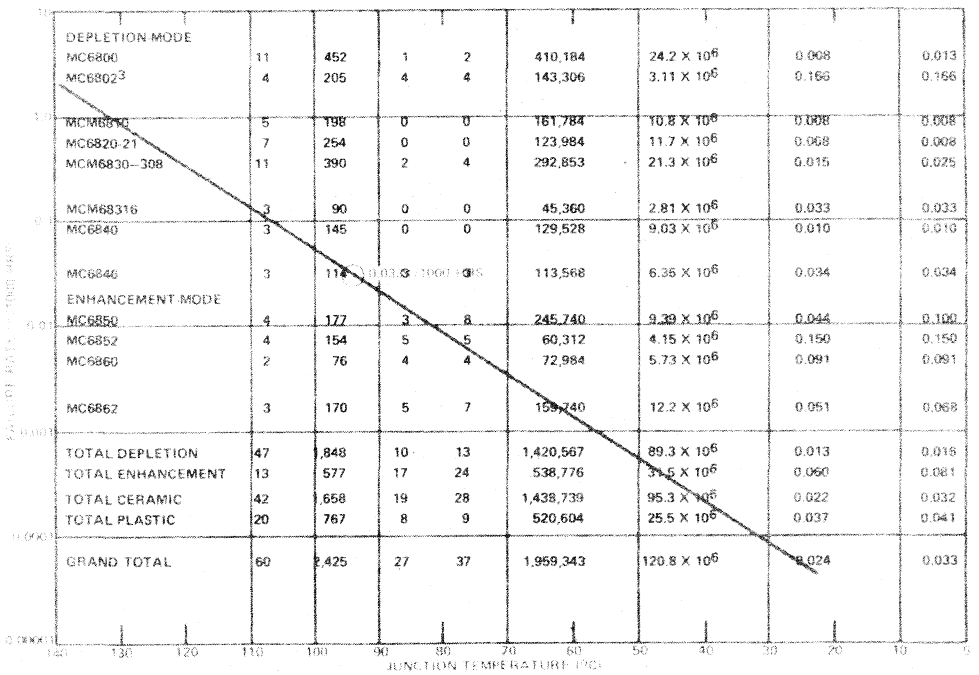
Reliability data is necessary for the component designer to evaluate and correct failure mechanisms toward the continual improvement of component reliability. It is also required by the system designer to relate component life to system requirements.

Motorola Reliability Report Number 7750 describes two tests applied to the Motorola M6800 Microprocessor Family—a long-term operation life test, and an environmental temperature-humidity-bias test. The report projects the results of 120 million equivalent device hours, on 12 different device types, as an impressively low *worst-case failure rate of 0.033% per 1000 hours, at the maximum rated temperature of 70° C.* The worst-case estimate was obtained by considering as failures even those devices that ran slightly "out of spec" during the test, but which would not normally cause malfunctioning of the associated equipment.

Environmental (THB) tests are being conducted principally on plastic-packaged devices which are much more subject to failures of this nature than the corresponding ceramic-package devices. Tests are conducted at an ambient temperature of 85°C, at a relative humidity of 85%, with a 5-volt bias to accelerate corrosion type failure mechanisms. Available data under these severe operating conditions indicates a (worst-case) median life of 18000 hours.

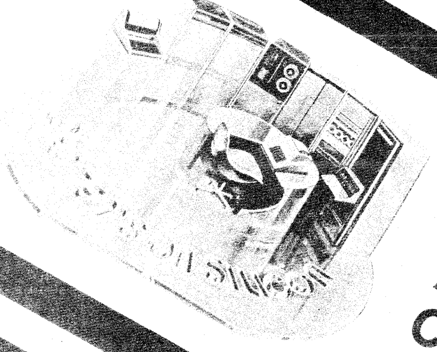
M6800 FAMILY OPERATING LIFE TEST DATA SUMMARY

FAILURE RATES AS A FUNCTION OF AMBIENT TEMPERATURE



DEVICE TYPE	WAFER LOTS	DEVICES	FAILURES FUNCT	FAILURES TOTAL	125°C ACTUAL DEVICE-HOURS	70°C EQUIV DEVICE-HOURS	70°C SYSTEM[1] FAILURE RATE	70°C TOTAL[2] FAILURE RATE
DEPLETION-MODE								
MC6800	11	452	1	2	410,184	24.2×10^6	0.008	0.013
MC6802[3]	4	205	4	4	143,306	3.11×10^6	0.166	0.166
MCM6810	5	198	0	0	161,784	10.8×10^6	0.008	0.008
MC6820-21	7	254	0	0	123,984	11.7×10^6	0.008	0.008
MCM6830-308	11	390	2	4	292,853	21.3×10^6	0.015	0.025
MCM68316	3	90	0	0	45,360	2.81×10^6	0.033	0.033
MC6840	3	145	0	0	129,528	9.03×10^6	0.010	0.010
MC6846	3	114	3	3	113,568	6.35×10^6	0.034	0.034
ENHANCEMENT-MODE								
MC6850	4	177	3	8	245,740	9.39×10^6	0.044	0.100
MC6852	4	154	5	5	60,312	4.15×10^6	0.150	0.150
MC6860	2	76	4	4	72,984	5.73×10^6	0.091	0.091
MC6862	3	170	5	7	159,740	12.2×10^6	0.051	0.068
TOTAL DEPLETION	47	1,848	10	13	1,420,567	89.3×10^6	0.013	0.016
TOTAL ENHANCEMENT	13	577	17	24	538,776	31.5×10^6	0.060	0.081
TOTAL CERAMIC	42	1,658	19	28	1,438,739	95.3×10^6	0.022	0.032
TOTAL PLASTIC	20	767	8	9	520,604	25.5×10^6	0.037	0.041
GRAND TOTAL	60	2,425	27	37	1,959,343	120.8×10^6	0.024	0.033

NOTES: [1]BASED ON FUNCTIONAL FAILURES [2]BASED ON TOTAL FAILURES [3]BASED ON INTERIM RESULTS



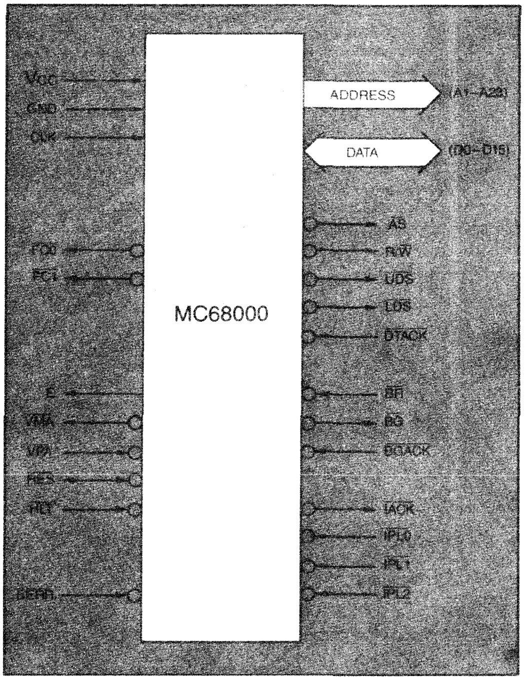
MOTOROLA'S ADVANCED COMPUTER SYSTEM ON SILICON

MC68000

MOTOROLA Semiconductor Products Inc.

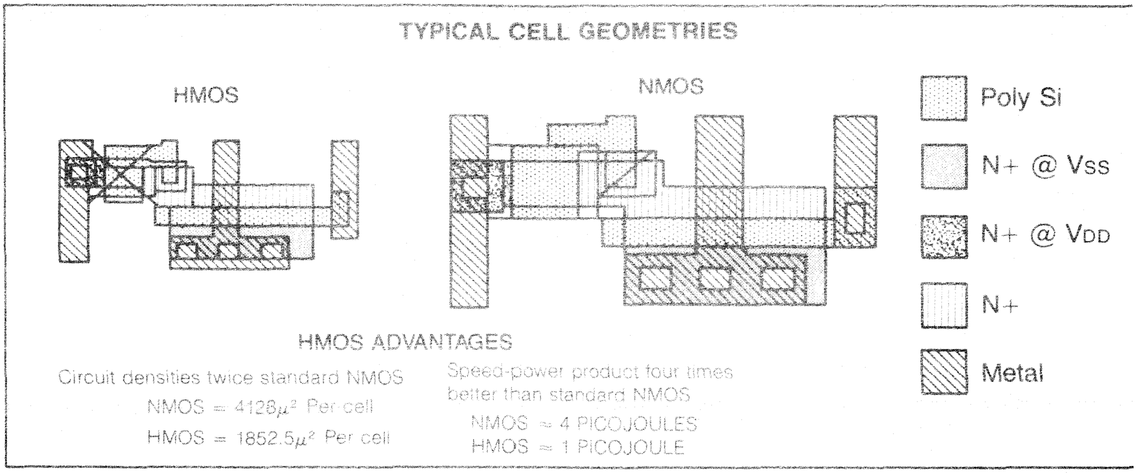
INTRODUCING THE MC68000 ... MOTOROLA'S ADVANCED COMPUTER SYSTEM ON SILICON

The MC68,000 microprocessor is housed in a 64-pin package that allows the use of separate (non-multiplexed) address and data buses. This large package provides optimum flexibility while at the same time maximizing bus through-put.



PIN IDENTIFICATION & DEFINITIONS

A1–A23	Address Leads	23-bit address bus; capable of addressing 16,777,216 bytes in conjunction with UDS and LDS.
D0–D15	Data Leads	16-bit data bus; transfers 8 or 16 bits of information.
AS	Address Strobe	Indicates valid address & provides a bus lock for indivisible operations.
R/W	Read/Write	Defines bus operation as Read or Write and controls external bus buffers.
UDS, LDS	Data Strobes	Identifies the byte(s) to be operated on according to R/W and AS.
DTACK	Data Transfer Acknowledge	Allows the bus cycle to synchronize with slow devices or memories.
BR	Bus Request	Input to the Processor from a device requesting the bus.
BG	Bus Grant	Output from the processor granting bus arbitration.
BGACK	Bus Grant Acknowledge	Confirmation signal from BG indicating a valid selection from the arbitration process.
IACK	Interrupt Acknowledge	Identifies that the bus is performing an interrupt service cycle.
IPL0, IPL1, IPL2	Interrupt Priority Level	Provides the priority level of the interrupting function to the processor.
FC0, FC1	Function Code	Provides external devices with information about the current bus cycle.
CLK	Clock	Master TTL input clock to the processor.
RES	Reset	Provides reset (initialization) signal to the processor and peripheral devices.
HLT	Halt	Stops the processor and allows single stepping.
BERR	Bus Error	Provides termination of a bus cycle if no response or an invalid response is received.
E VPA	Enable Valid Peripheral Address	Enable clock for M6800 systems. Identifies addressed area as a 6800 compatible area.
VMA	Valid Memory Address	Indicates to 6800 family devices that a valid address is on the bus.
V_{cc}	+5 Volts	—
GND	Ground (2 pins)	—



HMOS ADVANTAGES

Circuit densities twice standard NMOS
NMOS = 4128μ^2 Per cell
HMOS = 1852.5μ^2 Per cell

Speed-power product four times better than standard NMOS
NMOS ≈ 4 PICOJOULES
HMOS ≈ 1 PICOJOULE

Figure 1: Comparison of HMOS and NMOS Technologies
HMOS Technology used for the MC68000 results in significant improvements to Circuit Densities and Speed-Power Products

Advances in semiconductor technology have provided the capability to put on a single silicon chip, a microprocessor at least an order of magnitude higher in performance and circuit complexity than has been previously available. The MC68000 is the first of a family of such VLSI microprocessors from Motorola. It combines state-of-the-art technology and advanced circuit design techniques with computer sciences to achieve an architecturally advanced 16-bit microprocessor containing over 68000 active devices on a silicon chip. This high density of active elements coupled with an order of magnitude increase in performance over the original MC6800 is the direct result of significant advances in semiconductor technology. Advances such as dry PLASMA etching, projection printing, and HMOS (High density short channel MOS) circuit design techniques (Figure 1) have provided a sound technological base that has allowed Motorola's system engineers, computer scientists and marketing engineers a large degree of innovative freedom. The goals of applying this innovative freedom to microprocessors are to make the microprocessor easy to use, more reliable and more flexible for applications, while maximizing performance.

The resources available to the MC68000 user consist of the following:

- 32-bit data and address registers
- 16 mega-byte direct addressing range
- 61 powerful instruction types
- operations on six main data types
- memory mapped I/O
- 14 addressing modes

Particular emphasis has been given to the architecture to make it orthogonal (regular) with respect to the registers, instructions (including all addressing modes), and data types. Orthogonality makes the architecture easy to learn and program, and, in the process, reduces both the time required to write programs and the space required to store programs. The net result is a great reduction in the cost and risk o developing software.

High systems throughput (up to an aggregate o two million instruction and data word transfers pe second) is achieved even with readily availabl standard product memories with comparatively slow access times. The design flexibility of the data bu allows the mixing of slow and fast memories o peripherals with the processor, automatically optimizing the transfer rate on every access to keep th system operating at peak efficiency.

The hardware design of the CPU was heavily influenced by advances made in software technology High level language compilers as well as code produced from high level languages must run efficientl on the new generation 16-bit and 32-bit microprocessors. The MC68000 supports high level language with its consistent architecture, multiple registers anc stacks, large addressing range and high level language oriented instructions (LINK, UNLINK, CHK etc.). Also, operating systems for controlling th software operating environment of the MC68000 MPU are supported by privileged instructions, memor management, a powerful vectored multi-level interrupt and trap structure, and specific instruction (EXG, LDM, STM, TRAP, etc.).

The processor also provides both hardware an software interlocks for multiprocessor systems. Th CPU chip contains bus arbitration logic for a share bus and shared memory environment (shared wit other MC68000 processors, DMA devices, etc.). Multiprocessor systems are also supported with softwar instructions (TEST and SET, TEST and RESET, etc. The MC68000 offers the maximum flexibility fo microprocessor based multiprocessor systems.

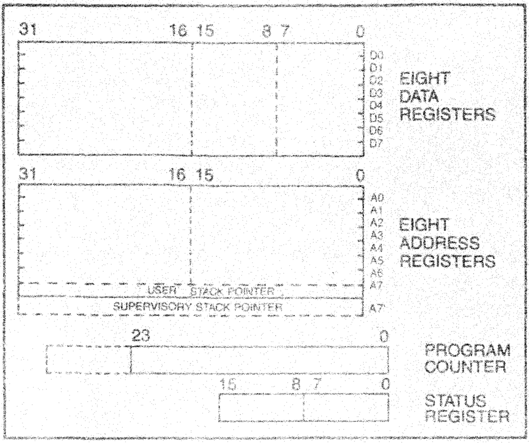


Figure 2: MC68000 Programming Model

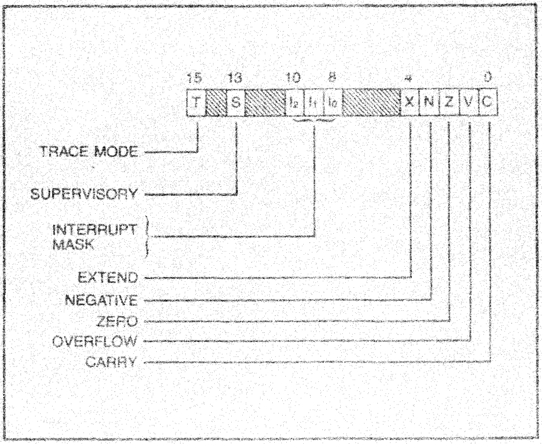


Figure 3: MC68000 Status Register

THE MC68000 CPU

Advanced architecture processors must not only offer efficient solutions to large complex problems but must be able to handle the small, simple problems with proportional efficiency. The CPU has been designed to offer the maximum in performance and versatility to solve simple and complex problems efficiently.

The MC68000 offers sixteen 32-bit registers in addition to the 24-bit program counter and 16-bit status register (Figure 2). The first eight registers (D0–D7) are used as data registers for byte (8-bit), word (16-bit) and long word (32-bit) operations. The second set of eight registers (A0–A7) may be used as software Stack Pointers and Base Address Registers. In addition, the second set of eight registers may be used for word and long word data operations. All of the sixteen registers may be used as Index Registers.

The 24-bit Program Counter provides a memory addressing range of more than 16 mega-bytes (actually 16,777,216 bytes). This large range of addressing capability, coupled with a Memory Management Unit, allows large, modular programs to be developed and operated without resorting to cumbersome and time consuming software bookkeeping and paging techniques.

The Status Register (Figure 3) contains the Interrupt Level Mask (8 levels available) as well as the Condition Code; Overflow (V), Zero (Z), Negative (N), Carry (C), and Extend (X). Additional status bits indicate that the processor is in a TRACE (T) mode or in a SUPERVISORY (S) state . Ample space remains in the Status Register for future extensions of the M68000 family.

Six basic data types are supported. These data types are:

- Bits
- BCD digits
- ASCII characters
- Bytes (8-bits)
- Words (16-bits)
- Long words (32-bits)

In addition operations on other data types such as memory addresses, status word data, etc. are provided for in the instruction set.

DEFINITIONS:

EA	= Effective Address
Ax	= Address Register
Dx	= Data Register
Rx	= Address or Data Register used as Index Register
SR	= Status Register
PC	= Program Counter
D_8	= Eight-Bit Offset
D_{16}	= Sixteen-Bit Offset
N	= 1 for Byte, 2 for Word and 4 for Long Word
( )	= Contents of
←	= Replaces

TABLE I: MC68000 DATA ADDRESSING MODES

REGISTER DIRECT ADDRESSING	
Data Register Direct	$EA = D_x$
Address Register Direct	$EA = A_x$
Status Register Direct	EA = SR
ABSOLUTE DATA ADDRESSING	
A. Absolute Short	EA = (Next Word)
B. Absolute Long	EA = (Next two Words)
PROGRAM COUNTER RELATIVE ADDRESSING	
Relative with Offset	$EA = (PC) + D_{16}$
Relative with Index & Offset	$EA = (PC) + (Rx) + D_8$
REGISTER INDIRECT ADDRESSING	
Register Indirect	EA = (Ax)
Post-increment Register Indirect	EA = (Ax), Ax ← Ax + N
Pre-decrement Register Indirect	Ax ← Ax − N, EA = (Ax)
Register Indirect with Offset	$EA = (Ax) + D_{16}$
Indexed Register Indirect with Offset	$EA = (Ax) + (Rx) + D_8$
IMMEDIATE DATA ADDRESSING	
Immediate	DATA = Next Word(s)
Quick Immediate	INHERENT DATA

The 14 flexible addressing modes, shown in Table I, include five basic types:

- Register Direct
- Immediate
- Register Indirect
- Absolute
- Program Counter Relative

Included in the addressing modes is the capability to do Post-incrementing, Pre-decrementing, Offsetting and Indexing.

THE INSTRUCTION SET

The MC68000 instruction set is rich and full as evidenced by the 61 distinct types shown in Table II. Special emphasis during the design has been given to the instruction set's support of structured high level languages that facilitate ease of programming. Each instruction, with few exceptions, operates on bytes, words, and long words and most instructions can use any of the 14 addressing modes. Combining instruction types, data types, and addressing modes, over 1000 useful instructions are provided. These instructions include signed and unsigned multiply and divide, "quick" arithmetic operations, BCD arithmetic and extended operations (through traps). The processor offers the most comprehensive and flexible instruction set of any microprocessor of any class, available today. Additionally, it's highly orthogonal, proprietary microcoded structure provides a sound flexible base for the future.

REDUCED SOFTWARE COST AND RISK

Advances in VLSI semiconductor technology have resulted in a significant reduction in the cost of computer hardware in recent years. The MC68000 microprocessor, for example, provides in a single integrated circuit package computing power that just a decade ago would have been three or four orders of magnitude more expensive. Software costs during this same period of time have, as a percentage of total system cost, increased significantly. This has been due primarily to inflation and the labor intensive nature of programming. Without significant architectural advances in computers, this trend can do nothing but continue. One of Motorola's major goals in developing this new microprocessor has been to reduce the costs of software. Many innovative features have been incorporated to make programming easier, faster and more reliable.

An Orthogonal 16-BIT MPU — The highly orthogonal or regular structure of the MC68000 microprocessor greatly simplifies the effort required to write programs in Assembly Language as well as in High Level Languages. Operations on integer data in registers and memory are independent of the data itself. Separate special instructions that operate on byte (8-bit), word (16-bit) and long-word (32-bit) integers are not necessary. The programmer merely has to remember one mnemonic for each type of operation and then specify data size, source addressing mode and destination addressing mode. This has helped keep the total number of instruction mnemonics for the M68000 to an easily remembered, yet complete, 61 types, eleven fewer than on Motorola's MC6800.

The dual operand nature of many of the instructions significantly increases the flexibility and power of this new Motorola microprocessor. Consistency again is maintained since *all* data registers and memory locations may be either a source or destination for most operations on integer data.

TABLE II: MC68000 INSTRUCTION SET SUMMARY

MNEMONIC	DESCRIPTION
ABCD	Add Decimal with Extend
ADD	Add
ADDX	Add with Extend
AND	Logical And
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
BCC	Branch Conditionally
BCHG	Bit Test and Change
BCLR	Bit Test and Clear
BRA	Branch Always
BSET	Bit Test and Set
BSR	Branch to Subroutine
BTST	Bit Test
CHK	Check Register Against Bounds
CLR	Clear Operand
CMP	Arithmetic Compare
DCNT	Decrement and Branch Non-Zero
DIVS	Signed Divide
DIVU	Unsigned Divide
EOR	Exclusive Or
EXG	Exchange Registers
EXT	Sign Extend
JMP	Jump
JSR	Jump to Subroutine
LDM	Load Multiple Registers
LDQ	Load Register Quick
LEA	Load Effective Address
LINK	Link Stack
LSL	Logical Shift Left
LSR	Logical Shift Right
MOVE	Move
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Two's Complement
NEGX	Two's Complement with Extend
NOP	No Operation
NOT	One's Complement
OR	Logical Or
PACK	Pack ASCII to BCD
PEA	Push Effective Address
RESET	Reset External Devices
ROTL	Rotate Left without Extend
ROTR	Rotate Right without Extend
ROTXL	Rotate Left with Extend
ROTXR	Rotate Right with Extend
RTR	Return and Restore
RTS	Return from Subroutine
SBCD	Subtract Decimal with Extend
SCC	Set Conditional
STM	Store Multiple Registers
STOP	Stop
SUB	Subtract
SUBX	Subtract with Extend
SWAP	Swap Data Register Halves
TAS	Test and Set Operand
TRAP	Trap
TRAPV	Trap on Overflow
TST	Test
UNLK	Unlink Stack
UNPK	Unpack BCD to ASCII

The addressing modes have been kept simple without sacrificing efficiency. All fourteen addressing modes operate consistently and are independent of the instruction operation itself. Additionally, all address registers may be used for the Direct, Register Indirect and Indexed addressing modes. (Immediate, Program Counter Relative and Absolute addressing by definition do not use address registers). For increased flexibility, any data register — as well as any address register — may be used as an Index Register. Address register consistency is maintained for stacking operations since any of the eight address registers may be utilized as User Program Stack pointers with the Register Indirect Post-increment/Predecrement addressing modes. Register A7, however, is a special register that, in addition to its normal addressing capability, functions as the System Stack Pointer when stacking the Program Counter and Status Register for subroutine calls, traps and interrupts: while in the supervisory mode.

Structured Modular Programming — The art of programming microprocessors has evolved rapidly in the past few years. Numerous advanced techniques have been developed to allow easier, more consistent and reliable generation of software. In general, these techniques require that the programmer be more disciplined in observing a defined *programming structure* such as modular programming. Modular programming allows a required function or process to be broken down in short modules or subroutines that are concisely defined and easily programmed and tested. Such a technique is greatly simplified by the availability of advanced macro assemblers and block structured High Level Languages such as PASCAL. Such concepts are virtually useless, however, unless parameters are easily transferred between and within software modules that operate on a reentrant and recursive basis. (To be reentrant a routine must be usable by interrupt and non-interrupt driven programs without the loss of data. A recursive routine is one that may call or use itself). The MC68000 microprocessor provides the necessary architectural features to allow efficient reentrant modular programming. The "LINK" and "UNLINK" instructions reduce subroutine call overhead in two complementary instructions by allowing the manipulation of linked lists of data areas on the stack. The "STM" (Store Multiple Registers) and "LDM" (Load Multiple Registers) instructions also reduce subroutine call programming overhead. These allow the loading or storing, via an effective address, multiple registers that are specified by the programmer. Sixteen software trap vectors are provided with the "TRAP" instruction and are useful in operating system call routines or user generated "macro routines." Other instructions that support modern structured programming techniques are PEA (Push Effective Address), LEA (Load Effective Address), RTR (Return to Restore) as well as the normal JSR, BSR and RTS.

Of course, the powerful vectored priority interrupt structure of the microprocessor allows straightforward generation of reentrant modular Input/Output routines. Eight maskable levels of priority with 192 vector locations provide maximum flexibility for I/O control. (A total of 256 vector locations are available for interrupts, hardware traps, and software traps.)

Improved Software Testability — One of the major tasks the system programmer encounters when writing software for microcomputers is the detection and correction of errors, or "debugging." The time taken to "debug" software nearly always exceeds the time it takes to write the software. In practice, the old 20/80 rule often applies: "The last 20% of the job requires 80% of the effort." The microprocessor incorporates several features that reduce the chance for errors. These features, such as Orthogonality and the Structured Modular Programming capability, have already been discussed.

Of major importance to the systems programmer are features that have been incorporated specifically to detect the occurrence of programming errors or "bugs." Several hardware traps, provided to indicate abnormal internal conditions of the MC68000 processor, detect the following error conditions:

Word access with an odd address

Illegal instructions

Unimplemented instructions

Illegal addressing mode

Illegal Memory access (bus error)

Overflow on divide (divide by zero)

Overflow condition code (separate instruction TRAPV)

Additionally, the sixteen software TRAP instructions may be utilized by the programmer to provide applications oriented error detection or correction routines.

An additional error detection tool is the CHK (Check Register Against Bounds) instruction used for array bound checking by verifying that 0 ≤ (REG) < LIMIT. A trap occurs if the register contents are negative or greater than the limit.

Finally, the MC68000 includes a facility that allows instruction-by-instruction tracing of a program being debugged. This TRACE MODE results in a trap being made to a tracing routine after each instruction execution. The TRACE MODE is available to the programmer when the microprocessor is in the SUPERVISORY state as well as the USER state, but may only be entered while in the supervisory state. The SUPERVISORY/USER states provide an additional degree of error protection for the microprocessor by providing memory protection of selected areas of memory when an external memory management device is used.

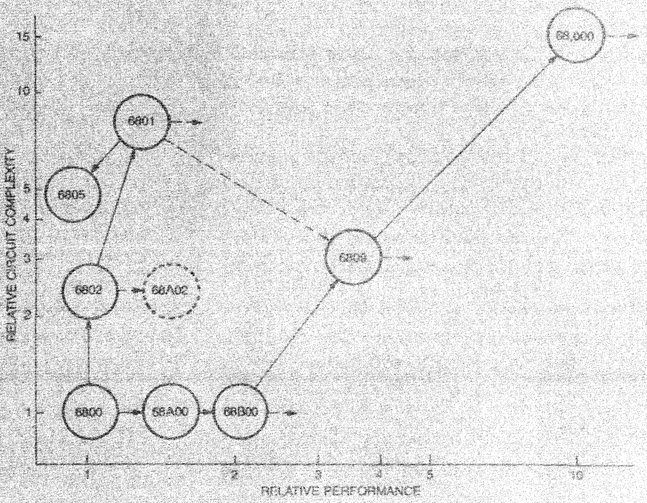


Figure 4: Motorola's Microprocessor Evolution

FUTURE FLEXIBILITY

Microprocessor VLSI circuit technology is advancing at an ever increasing rate. For example, the Motorola MC6800 — originally introduced in 1974 — has evolved into a number of more advanced products. This evolution has been along two paths: increased functionality, with the MC6802 and MC6801 microcomputers, and increased performance with the MC68A00, MC68B00 and MC6809 microprocessors. (Figure 4). The sound, well planned, architectural base provided by the original MC6800 made it possible to develop these improved products while taking full advantage of the major speed and density enhancements to NMOS VLSI. This was accomplished while maintaining an unprecedented degree of compatibility and consistency with the original MC6800 MPU.

Similarly, a major consideration in the development of the MC68000 microprocessor has been to provide a good, solid, but flexible, base for future extensibility. Several architectural concepts have been incorporated that will allow this advanced product to be enhanced as semiconductor technological advances are made. For example, the highly orthogonal structure of the CPU allows operations on 8-bit, 16-bit and 32-bit integers without the need for concatenation of registers or multiplexing of internal data buses. This regular structure of the CPU lends itself to a more consistent, reliable design that can be easily expanded.

The MC68000 incorporates a proprietary multi-level micro-programmed structure that allows significant versatility in the implementation of instructions. In fact, more than one-eighth of the instruction op-code map has been set aside specifically for implementation of future instructions. In the interim, user implementation of instructions not currently in the instruction set is possible through the use of the TRAP instruction, as well as the hardware trap structure.

MEMORY MANAGEMENT OF LARGE ADDRESSING SPACE

The ever-decreasing costs of semiconductor memories in combination with the use of high level languages and sophisticated disc operating systems allow Motorola's new generation of high performance microprocessors to be used in complex, memory intensive applications. In order to meet the needs of such applications, the MC68000 is capable of directly addressing more than 16 mega-bytes of memory. This large address space is directly accessed and managed very efficiently on a word or byte basis since operand size is specified by the instruction. The use of Upper Data Strobe (UDS) and Lower Data Strobe (LDS) signals allows easy access to high order bytes, low order bytes, or words.

Several additional useful features are provided that allow the programmer to efficiently manage memory usage. Powerful memory addressing modes such as Register Indirect, Indexed, Short and Long Absolute, and Program Counter Relative allow well-ordered access to specific memory locations. These addressing modes allow easy address calculations (Register Indirect and Indexed), direct access to memory location (Short and Long Absolute) and position independent or relocatable coding (Program Counter Relative). Of course, the Pre-decrement/Post-increment Register Indirect Addressing modes also allow efficient management of data in memory by permitting the programmer to generate as many as eight concurrent stacks or queues. Another feature that allows the programmer to manage the use of memory is the CHK (Check Register Against Bounds) Instruction. This instruction permits the software implementation of a basic memory protection/management structure.

Still another significant feature provided in the MC68000 microprocessor is the distinction between a USER and a SUPERVISOR mode. The SUPERVISOR mode permits certain protected operations within the processor system. Of particular interest is that an external Memory Management Controller may be used when the processor is in the USER mode to manage the large address space for the programmer. The controller's memory management operations are transparent to the programmer when in the USER mode and can be changed or updated only in the SUPERVISOR mode. The Memory Management Controller provides both management of a variable number of variable size segments (Memory Segmentation) and dynamic management of multi-task memory relocation and protection. The Memory Management Controller regulates access to storage segments that are dedicated to read only data, read/write data, program code and protected data/code.

REDUCED CODE DENSITY AND IMPROVED SPEED

With the advent of low cost, very high density VLSI RAMS and ROMS, it might incorrectly be assumed that the number of bytes of code needed to execute a given program is no longer important. Code density, however, is very critical, since microprocessor speed is highly dependent upon the number of executed instruction words. During the early development of Motorola's MC68000 microprocessor, extensive studies were made of the use of instructions and sequences of instructions in many microprocessor applications. These studies identified not only statically frequent instructions but also dynamically frequent instructions. (The dynamic frequency of instructions is a measure of how often an instruction is executed while static frequency is a measure of how often it occurs in a program listing or is encountered by an assembler). The major contributer to the increased efficiency, as a result of the studies, is the highly regular or orthogonal structure of the architecture. The consistency of the architecture, instruction set, and addressing modes significantly reduces the number of instructions needed to accomplish a given task. Additionally, many instructions have been included to specifically improve code density and speed. For example, single word Add and Subtract instructions using Quick Immediate addressing allow fast, small value arithmetic operations on data registers and memory. A Load Quick Immediate (LDQ) provides the ability to load a small (8-bit) signed word into any register in a single word operation. In order to improve the speed of loop operations, a single word instruction for Decrement Count by One and Branch if non-zero (DCNT) is included. Of course, the TRAP, Store Multiple Registers (STM), Load Multiple Registers (LDM), Link Stack (LINK), Unlink Stack (UNLK) and Check Limit (CHK) instructions significantly reduce code requirements for subroutines, operating system calls and stacking operations.

Other instructions that help reduce coding requirements and improve performance of arithmetic operations are Signed and Unsigned Multiply (MULS and MULU), Signed and Unsigned Divide (DIVS and DIVU), BCD Arithmetic (ABCD, SBCD, PACK and UNPK) as well as the standard binary integer operations. In order to improve the efficiency of moving or transferring data, a powerful MOVE data instruction has been incorporated that allows the transfer of bytes, words and long words and operates in all data addressing modes. Thus; register-to-register, register-to-memory, memory-to-register and memory-to-memory transfers are permitted.

In addition to the powerful instructions that provide a substantial improvement in processor through-put, numerous architectural features significantly reduce the execution times for all instructions. The separate (non-multiplexed) address and data buses, instruction pre-fetch pipeline and 32-bit internal registers are major contributors to the processor's unequaled performance. As an example of the performance capability of the MC68000 Table III and the accompanying graphs in figures 5 and 6 summarize the execution times for a number of common instructions. For comparison purposes, similar information is provided for Zilog's Z-8000 microprocessor. It is interesting to note that the MC68000 has significantly faster execution times.

TABLE III — EXECUTION TIMES FOR *MOVB R, SRC* INSTRUCTION FOR VARIOUS ADDRESSING MODES

Source Addressing	Motorola MC68000	Zilog Z-8000
Register	0.5us	0.75us
Indirect Register	1.0	1.75
Absolute Addressing (Direct)	1.5	2.25
Indexed Addressing	1.5	2.50
Immediate	1.0	1.00

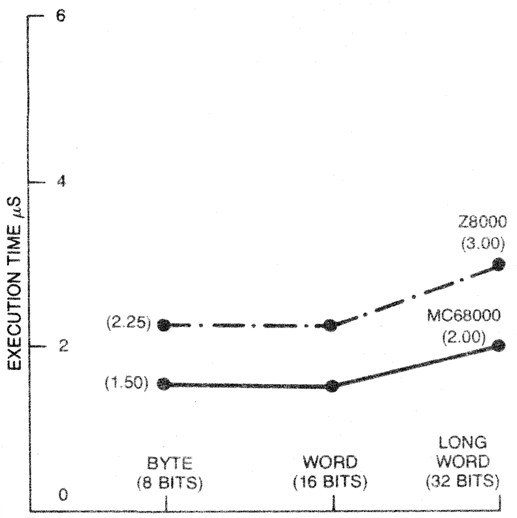


FIGURE 5: Execution Time for the *Add Data Element to a register from a short Absolute Address* **Instruction.**

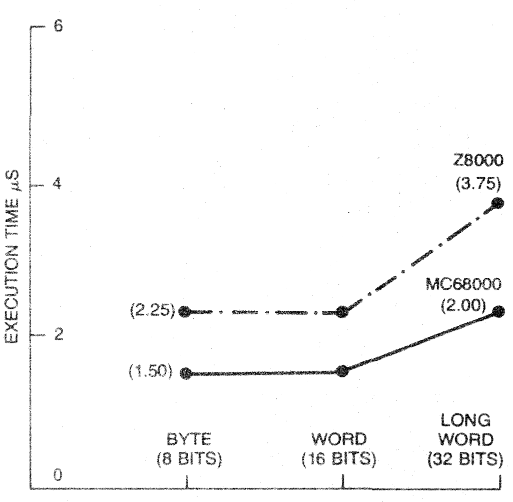


FIGURE 6: Execution Time for the *move a data element from memory to a register from short Absolute Address* **Instruction.**

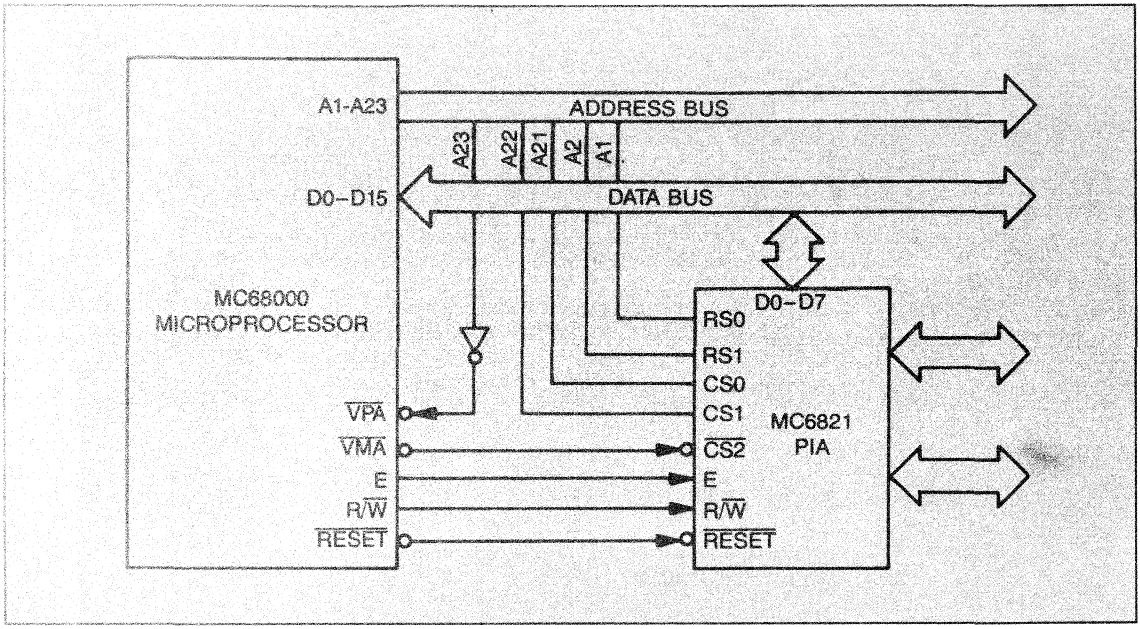


Figure 7: Example of MC68000 Interface Connections for MC6821 Peripheral Interface Adapter

SOFTWARE SUPPORT AND MC6800 COMPATIBILITY

The system designers and programmers using the MC68000 in an application have available a complete, compatible system of hardware and software. The microprocessor is supported by a full range of software development tools including disc operating systems, debug aids, assemblers, and high level languages. In addition, a translator will allow the present M6800 Family user to convert existing programs to run on the MC68000 with a minimum of programmer intervention.

The careful planning of this new microprocessor provides a superset of the MC6800 instruction set enhanced by the addition of more and larger registers, powerful orthogonal structure and many flexible addressing modes. This allows efficient translation of existing MC6800 programs, which can then be further optunized by taking full advantage of the versatile and powerful features of the MC68000.

This careful planning of similarities between the MC68000 and the MC6800 does not stop at software compatibility (by translation) but also extends to peripheral controller interfacing. Motorola's extensive line of intelligent M6800 family peripherals (including the MC6854 Advanced Data Link Controller and the MC68488 General Purpose Interface Adapter) can be directly and easily interfaced to the MC68000. Three signal lines; Enable (E), Valid Memory Address (VMA), and Valid Peripheral Address (VPA) are provided to simplify the interface to Motorola's standard MC6800 peripherals as shown in Figure 7. Interface to the new MC6801E (Single Chip Programmable Controller) is also possible, allowing user implementation of specialized input/output functions. In addition, the MC68000 is supported by unique peripheral controllers expected of an advanced architecture microprocessor, including a DMA Controller and a Memory Management Unit.

The MC68000 is not just a component. By a unique blend of VLSI design, software engineering and careful planning, the MC68000 it *is* Motorola's Advanced Computer System on Silicon.

AUTHORIZED MOTOROLA SEMICONDUCTOR DISTRIBUTORS

†**ALABAMA**, Huntsville	Hall-Mark Electronics	(205) 837-8700
ALABAMA, Huntsville	Pioneer Electronics	(205) 837-9300
ARIZONA, Phoenix	Hamilton/Avnet Electronics	(602) 275-7851
†**ARIZONA**, Phoenix	Liberty Electronics	(602) 249-2232
ARIZONA, Phoenix	Sterling Electronics	(602) 258-4531
CALIFORNIA, Costa Mesa	Avnet	(714) 754-6111 (213) 558-2345
CALIFORNIA, Culver City	Hamilton Electro Sales/L.A.	(213) 558-2000
†**CALIFORNIA**, El Segundo	Liberty Electronics Corp.	(213) 322-8100
CALIFORNIA, Irvine	Cramer/Los Angeles	(213) 771-8300
CALIFORNIA, Irvine	Schweber Electronics	(213) 537-4321 (714) 556-3800
†**CALIFORNIA**, Mountain View	Elmar Electronics, Inc.	(415) 961-3611
CALIFORNIA, Mountain View	Hamilton/Avnet Electronics	(415) 961-8600
CALIFORNIA, Palo Alto	Kieruiff Electronics, Inc.	(415) 968-6292
CALIFORNIA, San Diego	Hamilton/Avnet Electronics	(714) 279-2421
†**CALIFORNIA**, San Diego	Liberty Electronics	(714) 565-9171
#**CALIFORNIA**, Sunnyvale	Western Microtechnology Sales	(408) 737-1660
†**COLORADO**, Commerce City	Elmar Electronics, Inc.	(303) 287-9611
COLORADO, Denver	Hamilton/Avnet Electronics	(303) 534-1212
CONNECTICUT, Danbury	Schweber Electronics	(203) 792-3500
CONNECTICUT, Georgetown	Hamilton/Avnet Electronics	(203) 762-0361
CONNECTICUT, North Haven	Cramer/Connecticut	(203) 239-5641
FLORIDA, Clearwater	Hamilton/Avnet Electronics	(813) 576-3930
FLORIDA, Ft. Lauderdale	Hall-Mark Electronics	(305) 971-9280
FLORIDA, Ft. Lauderdale	Hamilton/Avnet Electronics	(305) 971-2900
FLORIDA, Hollywood	Schweber Electronics	(305) 927-0511
FLORIDA, Orlando	Cramer/Orlando	(305) 894-1511
†**FLORIDA**, Orlando	Hall-Mark Electronics	(305) 855-4020
GEORGIA, Atlanta	Schweber Electronics	(404) 449-9170
GEORGIA, Norcross	Hamilton/Avnet Electronics	(404) 448-0806
††**ILLINOIS**, Chicago	Newark Electronics Corp.	(312) 638-4411
ILLINOIS, Elmhurst Industrial Park	Semiconductor Specialists, Inc./Chicago	(312) 279-1000
ILLINOIS, Elk Grove Village	Pioneer/Chicago	(312) 437-9680
ILLINOIS, Elk Grove Village	Schweber Electronics	(312) 593-2740
ILLINOIS, Mt. Prospect	Cramer/Chicago	(312) 593-8230
ILLINOIS, Schiller Park	Hamilton/Avnet Electronics	(312) 678-6310
INDIANA, Indianapolis	Graham Electronics Supply, Inc.	(317) 634-8202
†**INDIANA**, Indianapolis	Pioneer/Indianapolis	(317) 849-7300
KANSAS, Overland Park	Hamilton/Avnet Electronics	(913) 888-8900
KANSAS, Shawnee Mission	Hall-Mark Electronics	(913) 888-4747
LOUISIANA, Metairie	Sterling Electronics	(504) 887-7610
MARYLAND, Gaithersburg	Cramer/Washington	(301) 948-0110
†**MARYLAND**, Gaithersburg	Pioneer/Washington Electronics	(301) 948-0710
MARYLAND, Hanover	Hamilton/Avnet Electronics	(301) 796-5684
MARYLAND, *Rockville*	*Schweber Electronics*	(301) 881-3300
MARYLAND, Savage	Pyttronic Industries, Inc.	(301) 953-9650
****MASSACHUSETTS**, Burlington	Zeus Components	(617) 273-0750
MASSACHUSETTS, Lexington	Harvey Electronics	(617) 861-9200
†**MASSACHUSETTS**, Newton	Cramer Electronics, Inc.	(617) 969-7700
MASSACHUSETTS, Newton	Impact Sales Co., Inc	(617) 964-7740
MASSACHUSETTS, Waltham	Schweber Electronics	(617) 890-8484
MASSACHUSETTS, Woburn	Hamilton/Avnet Electronics	(617) 933-8000
MICHIGAN, Livonia	Hamilton/Avnet Electronics	(313) 522-4700
†**MICHIGAN**, Livonia	Pioneer-Standard Electronics	(313) 525-1800
MICHIGAN, Livonia	R S Electronics	(313) 525-1155
MINNESOTA, Bloomington	Hall-Mark Electronics	(612) 884-9056
MINNESOTA, Eden Prairie	Schweber Electronics	(612) 941-5280
†**MINNESOTA**, Edina	Cramer-Minnesota	(612) 835-7811
MINNESOTA, Edina	Hamilton/Avnet Electronics	(612) 941-3801
MISSOURI, Earth City	Hall-Mark Electronics	(314) 291-5350
MISSOURI, Hazelwood	Hamilton/Avnet Electronics	(314) 731-1144
†**MISSOURI**, Kansas City	LCOMP/Kansas City, Inc	(816) 221-2400
†**MISSOURI**, Maryland Heights	LCOMP/St. Louis, Inc.	(314) 291-6200
NEW JERSEY, Cedar Grove	Hamilton/Avnet Electronics	(201) 239-0800
NEW JERSEY, Mt Laurel	Hamilton/Avnet Electronics	(609) 234-2133
NEW JERSEY, Pennsauken	RESCO Electronics	(609) 662-4000 (215) 925-6900
NEW JERSEY, Somerset	Schweber Electronics	(201) 469-6008
NEW MEXICO, Albuquerque	Hamilton/Avnet Electronics	(505) 765-1500
*****NEW YORK**, Elmsford	Zeus Components	(914) 592-4121
NEW YORK, Farmingdale	Harrison Radio Corporation	(516) 293-7979
NEW YORK, Hauppauge, L.I.	Cramer/Long Island	(516) 231-5600
†**NEW YORK**, Rochester	Cramer/Rochester	(716) 275-0300
NEW YORK, Rochester	Hamilton/Avnet Electronics	(716) 442-7820
NEW YORK, Rochester	Schweber Electronics	(716) 461-4000
†**NEW YORK**, Syracuse	Cramer/Syracuse	(315) 437-6671
NEW YORK, East Syracuse	Hamilton/Avnet Electronics	(315) 437-2642
†**NEW YORK**, Westbury, L.I.	Schweber Electronics	(516) 334-7474
NEW YORK, Woodbury, L.I.	Harvey Electronics	(516) 921-8700
NORTH CAROLINA, Greensboro	Pioneer/Washington	(919) 273-4441
NORTH CAROLINA, Raleigh	Hall-Mark Electronics	(919) 832-4465
NORTH CAROLINA, Raleigh	Hamilton/Avnet Electronics	(919) 829-8030
NORTH CAROLINA, Winston-Salem	Cramer/Winston-Salem	(919) 725-8711
OHIO, Beachwood	Schweber Electronics	(216) 464-2970
OHIO, Cincinnati	Hughes-Peters	(513) 351-2000
OHIO, Cleveland	Hamilton/Avnet Electronics	(216) 461-1400
†**OHIO**, Cleveland	Pioneer-Standard Electronics, Inc.	(216) 587-3600
OHIO, Columbus	Hughes-Peters	(614) 294-5351
OHIO, Dayton	Hamilton/Avnet Electronics	(513) 433-0610
†***OHIO**, Dayton*	*Pioneer/Dayton*	(513) 236-9900
OHIO, Solon	Cramer/Cleveland	(216) 248-8400
OKLAHOMA, Tulsa	Hall-Mark Electronics	(918) 835-8458
PENNSYLVANIA, Horsham	Pioneer/Washington	(215) 674-4000
PENNSYLVANIA, Montgomeryville	Pyttronic Industries	(215) 643-2850
††**PENNSYLVANIA**, Philadelphia	Philadelphia Electronics Inc.	(215) 568-7400
†**PENNSYLVANIA**, Pittsburgh	Pioneer-Standard Electronics	(412) 782-2300
SOUTH CAROLINA, Columbia	Dixie Electronics	(803) 779-5332
TEXAS, Austin	Hall-Mark Electronics	(512) 837-2814
TEXAS, Austin	Sterling Electronics	(512) 836-1341
†**TEXAS**, Dallas	Hall-Mark Electronics	(214) 234-7400
TEXAS, Dallas	Hamilton/Avnet Electronics	(214) 661-8661
TEXAS, *Dallas*	*Schweber Electronics*	(214) 661-5010
TEXAS, Dallas	Sterling Electronics	(214) 357-9131
TEXAS, El Paso	Midland Speciality Co.	(915) 533-9555
††**TEXAS**, Dallas	Trevino Electronics, Inc.	(214) 358-2418
TEXAS, Houston	Hall-Mark Electronics	(713) 781-6100
TEXAS, Houston	Hamilton/Avnet Electronics	(713) 780-1771
†**TEXAS**, Houston	Sterling Electronics, Inc.	(713) 627-9800
UTAH, Salt Lake City	Hamilton/Avnet Electronics	(801) 972-2800
WASHINGTON, Bellevue	Hamilton/Avnet Electronics	(206) 746-8750
WASHINGTON, Bellevue	Liberty Electronics Corp.	(206) 453-8300
†**WASHINGTON**, Seattle	Almac/Stroum Electronics	(206) 763-2300
†**WISCONSIN**, Milwaukee	Marsh Electronics	(414) 475-6000
WISCONSIN, New Berlin	Hamilton/Avnet Electronics	(414) 784-4510

CANADA

CALGARY, Alberta	L. A. Varah, Ltd	(403) 276-8818
DOWNSVIEW, Ontario	Zentronics, Ltd.	(416) 635-2822
EDMONTON, Alberta	Bowtek Electric Co., Ltd.	(403) 426-1072
HAMILTON, Ontario	L. A. Varah, Ltd.	(416) 561-9311
LONDON, Ontario	C. M. Peterson Co., Ltd.	(519) 434-3204
MISSISSAUGA, Ontario	Hamilton/Avnet Int'l Canada Ltd.	(416) 677-7432
MONTREAL, Quebec	Cesco Electronics Ltd.	(514) 735-5511
OTTAWA, Ontario	Zentronics, Ltd.	(613) 238-3591
OTTAWA, Ontario	Hamilton/Avnet Int'l Canada Ltd.	(613) 226-1700
QUEBEC CITY, Quebec	Cesco Electronics, Ltd	(418) 524-4641
ST. LAURENT, Quebec	Hamilton/Avnet Int'l Canada Ltd.	(514) 331-6443
TOWN OF MOUNT ROYAL, Quebec	Zentronics, Ltd.	(514) 735-5361
VANCOUVER, B.C.	Intek Electronics, Ltd.	(604) 324-6831
VANCOUVER, B.C.	L. A. Varah, Ltd.	(604) 873-3211
†**WILLOWDALE**, Ontario	Electro Sonic, Inc.	(416) 494-1555
WINNIPEG, Manitoba	L. A. Varah, Ltd.	(204) 633-6190

***Power Products**
****Military Products**
#LS TTL, M2900 Series, TTL RAMs
†Subsystem Products Also
††Subsystem Products

MOTOROLA SEMICONDUCTOR SALES OFFICES

MOTOROLA SEMICONDUCTOR AMERICAS DISTRICT OFFICES

ALABAMA, Huntsville 35805, 2611 Artie St., Suite 4 (205) 533-1650
ARIZONA, Phoenix 85018, 4350 E. Camelback Road, Suite 230F (602) 994-6326
CALIFORNIA, Encino/Sherman Oaks 91403, 15355 Morrison St., Suite 105 (213) 986-6650
Mail to: P.O. Box 9031, Van Nuys, CA 91409
CALIFORNIA, Orange 92668, One City Blvd. West,
Bank of America Tower Bldg., Suite 722 . . . (Orange Exch.) (714) 634-2844
Mail to: P.O. Box 11987, Santa Ana CA 92711 . . (L.A. Exch.) (213) 865-9552
CALIFORNIA, San Diego 92111, 7071 Convoy Court, Suite 210 (714) 560-4644
CALIFORNIA, San Jose 95117, 4000 Moorpark Ave., Suite 216 (408) 985-0510
COLORADO, Denver 80237, 3515 S. Tamarac, Suite 330 (303) 773-6800
CONNECTICUT, New Haven/Hamden 06518, 3074 Whitney Avenue,
Building C, Room 1 & 2 (203) 281-0771
FLORIDA, Pompano Beach/Ft. Lauderdale 33309,
1001 N.W. 62nd Street, Suite 310 (305) 491-8141
FLORIDA, Altamonte Springs/Maitland 32701, 253 Whooping Loop (305) 831-3422
FLORIDA, St. Petersburg 33702,
9720 Executive Center Drive North, Suite 108 (813) 576-6030
GEORGIA, *Atlanta 30328, 6085 Barfield Rd., Suite 114* (404) 394-6627
ILLINOIS, Chicago/Park Ridge 60068,
1460 Renaissance Drive, Suite 310 (312) 576-2600
INDIANA, Fort Wayne 46808, Franklin National Life Insurance Bldg.
2100 Goshen Road, Suite 208 (219) 484-0436
INDIANA, Indianapolis 46250, 6525 East 82nd Street, Suite 106 (317) 849-7050
IOWA, Cedar Rapids 52402, 206 Collins Road N.E. (319) 377-9439
KANSAS, Kansas City/Mission 66202, 6700 W. Squibb Road, Suite 104 (913) 384-3050
MASSACHUSETTS, Boston/Lexington 02173, 2 Militia Drive (617) 861-1350
MICHIGAN, Benton Harbor/Douglas 49406, 36 South Center (616) 857-2159
Mail to: P.O. Box 567, Saugatuck, MI 49453
MICHIGAN, Detroit/Westland 48185, Holiday Park *Plaza*, Suite 210 (313) 261-6200
8623 N. Wayne Road
MINNESOTA, Minneapolis 55426 6950 Wayzata Blvd., Suite 405 (612) 545-0251
MISSOURI, St. Louis 63141, 760 Office Parkway (314) 872-7681
NEW JERSEY, River Edge 07661, 337 Johnson Avenue (201) 488-1200
NEW YORK, Poughkeepsie/Fishkill 12524, Route 9 (914) 896-8970
NEW YORK, Long Island/Hauppauge 11787, 350 Vanderbilt Motor Parkway (516) 231-9000
NEW YORK, Rochester 14618, 3360 Monroe Avenue (716) 381-7220
NEW YORK, Syracuse 13211, 123 Pickard Bldg., E. Molloy Road (315) 454-9373
NORTH CAROLINA, Raleigh 27612, 3716 National Dr., Suite 101 (919) 782-7604
OHIO, Cleveland 44143, 840 Brainard Road (216) 461-3160
OHIO, Dayton 45439, 3490 South Dixie Drive, Room 130 (513) 294-2231
OHIO, Columbus/Worthington 43085, 933 High Street, Suite 116 (614) 846-9460
OKLAHOMA, Tulsa 74145, 4833 South Sheridan, Suite 406 (918) 664-5227
OREGON, *Portland 97221, 1730 S.W. Skyline Blvd., Suite 228* (503) 297-2235
PENNSYLVANIA, Philadelphia/Ft. Washington 19034,
501 Office Center Drive, Suite 128 (215) 643-4500
TENNESSEE, Knoxville 37919, 9041 Executive Park Drive,
Building 400, Suite 403 (615) 690-5592
TEXAS, Austin 78752, 7715 Chevy Chase 4, Suite 125 (512) 452-7673
TEXAS, Dallas 75234, 4825 LBJ Freeway, Suite 140 (214) 661-9829
TEXAS, Ft. Worth 76113, 6335 Grapevine Highway (Ext. 555) (817) 284-4661
TEXAS, Houston 77018, 2190 North Loop West, Suite 303 (713) 688-4583
VIRGINIA, Charlottesville 22901, 400 Preston Ave., Suite 300 (804) 977-3691
WASHINGTON, Seattle/Kirkland 98033,
10604 N.E. 38th Place, Suite 119 (206) 827-4661
***WASHINGTON, D.C./MARYLAND**, Lanham 20801,*
5900 Princess Garden Parkway, Suite 804 (301) 577-2600
WISCONSIN, Milwaukee/Wauwatosa 53226, 909 North Mayfair Road (414) 476-5554
Field Applications Engineering Available Through All Sales Offices

MOTOROLA SEMICONDUCTOR INTER-COMPANY OFFICES

ARIZONA, Scottsdale 85251, 8201 E. McDowell Road, Room 3195 (602) 949-3811
FLORIDA, Ft. Lauderdale 33313, 8000 W. Sunrise Blvd. (305) 473-6000
ILLINOIS, Franklin Park/Schaumburg 60196
CONSUMER/1229 E. Algonquin Road, Room 2024 (312) 576-2788
ILLINOIS, Schaumburg 60196
INDUSTRIAL/1301 E. Algonquin Road, Room 2707 (312) 576-5518

MOTOROLA SEMICONDUCTOR PRODUCTS — CANADA

ONTARIO, Downsview/Toronto N3N 1Y4 (416) 661-6400
490 Norfinch Drive
ONTARIO, Ottawa K1Z 6A6 (613) 729-4361
1007 Merivale Road, Room 106
QUEBEC, Montreal H4T 1G1 (514) 731-6881
7800 Cote de Liesse Road
BRITISH COLUMBIA, North Vancouver (604) 985-4618
706 East 7th Street

MOTOROLA SEMICONDUCTOR INTERNATIONAL SALES OFFICES

ARGENTINA, Buenos Aires Motorola International, Inc.
Uruguay 485, Floor 6th, Office "C" 46-9797
AUSTRALIA, Sydney Motorola Semiconductor Products Div.
37-43 Alexander St., Suite 204 Motorola Australia Pty. Ltd.
Crow's Nest, N.S.W. 2065 43-4409/4299
DENMARK, DK2800 Lyngby Motorola Semiconductors
Bredebovej 23 (01) 88 44 55
BRAZIL, Sao Paulo Motorola Semiconductores do Brazil, Ltd.
Caixa Postal 8718 70-2395 or 70-7286
ENGLAND, Wembley, Middlesex Motorola Semiconductors Ltd.
York House, Empire Way 01-902-8836
ENGLAND, Manchester, 2. Lancashire M2 5WS Motorola Semiconductors Ltd.
Television House, 10/12 Mount Street 061-833-0731
FRANCE, Paris 75007 Motorola Semiconducteurs, S.A.
15-17 Avenue de Segur 551-50-61
Mail: Boite Postale 101-07, 75326 Paris Cedex 07 551 50 61
GERMANY, 3012 Langenhagen/Hannover Motorola GmbH, Geschaftsbereich Halbleiter
Hans-Boeckler-Strasse 30 (0511) 77-20-37
GERMANY, 8043 Munich, Unterfoehring Motorola GmbH, Geschaftsbereich Halbleiter
Muenchner Strasse 18 089-95-1041
GERMANY, Nurnberg Motorola GmbH, Geschaftsbereich Halbleiter
Virnsberger Strasse 43 (0911) 65761
GERMANY, 7032 Sindelfingen Motorola GmbH, Geschaftsbereich Halbleiter
Stralsunder Strasse 1, P.O.B. 450 (07031) 83074
GERMANY, 6200 Wiesbaden Motorola GmbH, Geschaftsbereich Halbleiter
A. Lincoln Strasse 28 (06121) 76-1921
HOLLAND, Utrecht Motorola N.V.
Emmalaan, 41 030 510207
HONG KONG, Kowloon City, Kowloon Motorola Semiconductors Hong Kong Ltd.
10th Floor, Block C, Eldex Industrial Building, 3-69 4717
21 Matauwie Road, P.O. Box 9064
ISRAEL, Tel Aviv 67-899 Motorola Israel, Ltd.
16, Kremenetski Street 03-38973
ITALY, 40137 Bologna Motorola Semiconduttori S.p.A.
Via Portanova 266905
ITALY, 20129 Milan Motorola Semiconduttori S.p.A.
Via Ciro Menotti 11 738-6141
ITALY, 00162 Rome Motorola Semiconduttori S.p.A.
Via Constantino Maes 68 83,14,7,46
JAPAN, Osaka Motorola Semiconductors, Ltd.
Shin Osaka Chisan Bldg., 7-1-26, Nishinakajima, Yodogawa-Ku 06-262-4637
JAPAN, Tokyo Motorola Semiconductors Japan Ltd.
12-18, Jingyu-Mae, 6-Chome, Shibuya-Ku 03-499-1241
MEXICO, Mexico 20, D.F. Productos Semiconductores Motorola de Mexico, S.A.
Tecuyotitla No. 191-A, Col. Florida (905) 524-6181
PUERTO RICO, San Juan/Santurce 00907 Motorola Americas Inc.
Sonsid Condominium, 1319 Ashford Avenue (809) 723-9350
SINGAPORE, 9 Motorola Singapore Pty. Ltd.
Room 520 Plaza Singapora, 68 Orchard Road 361-755
SWEDEN, S-171 40 Solna Motorola Semiconductor AB
Vireberisvagen 19 08/82 02 95
SWITZERLAND, 1211 Geneva-Montbrilliant 20 Motorola Semiconductor Products Inc.
16, chemin de la Voie-Creuse, P.O. Box 8 (022) 33 56 07
SWITZERLAND, 8702 Zollikon-Zurich Motorola Semiconductor Products Inc.
101, Alte Landstrasse — P.O. Box 62 (051) 65 56 56/07
TAIWAN, Taipei Motorola Asia, Ltd.
4/F Sunrise Plaza Building, #2 Far Deh Road, Section 3 7528944-8

MOTOROLA Semiconductor Products Inc.
P.O. BOX 20912 • PHOENIX, ARIZONA 85036 • A SUBSIDIARY OF MOTOROLA INC.

MC 68000

PROPOSED PERIPHERALS

M. HILL
20 APR 79

MC68000 PROPOSED PERIPHERALS

THIS PRESENTATION IS ADVANCED INFORMATION ON PROPOSED PRODUCTS AND MOTOROLA RESERVES THE RIGHT OT MAKE CHANGES TO ANY PRODUCTS HEREIN TO IMPROVE RELIABILITY, FUNCTION, OR DESIGN. MOTOROLA DOES NOT ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCTS OR CIRCUITS DESCRIBED HEREIN. NEITHER DOES IT CONVEY ANY LICENSE UNDER ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS.

EXISTING M6800 PERIPHERAL PARTS

PIA - PERIPHERAL INTERFACE ADAPTER

ACIA - ASYNCHRONOUS COMMUNICATIONS INTERFACE ADAPTER

ADLC - ADVANCED DATA LINK CONTROLLER

SSDA - SYNCHRONOUS SERIAL DATA LINK CONTROLLER

PTM - PROGRAMMABLE TIMER MODULE

GPIA - GENERAL PURPOSE INTERFACE ADAPTER

VDG - VIDEO DISPLAY GENERATOR

CRTC - CRT CONTROLLER

NEW MC68000 PROPOSED PERIPHERAL PARTS

CONTROLLERS:

DMAC - DIRECT MEMORY ACCESS CONTROLLER

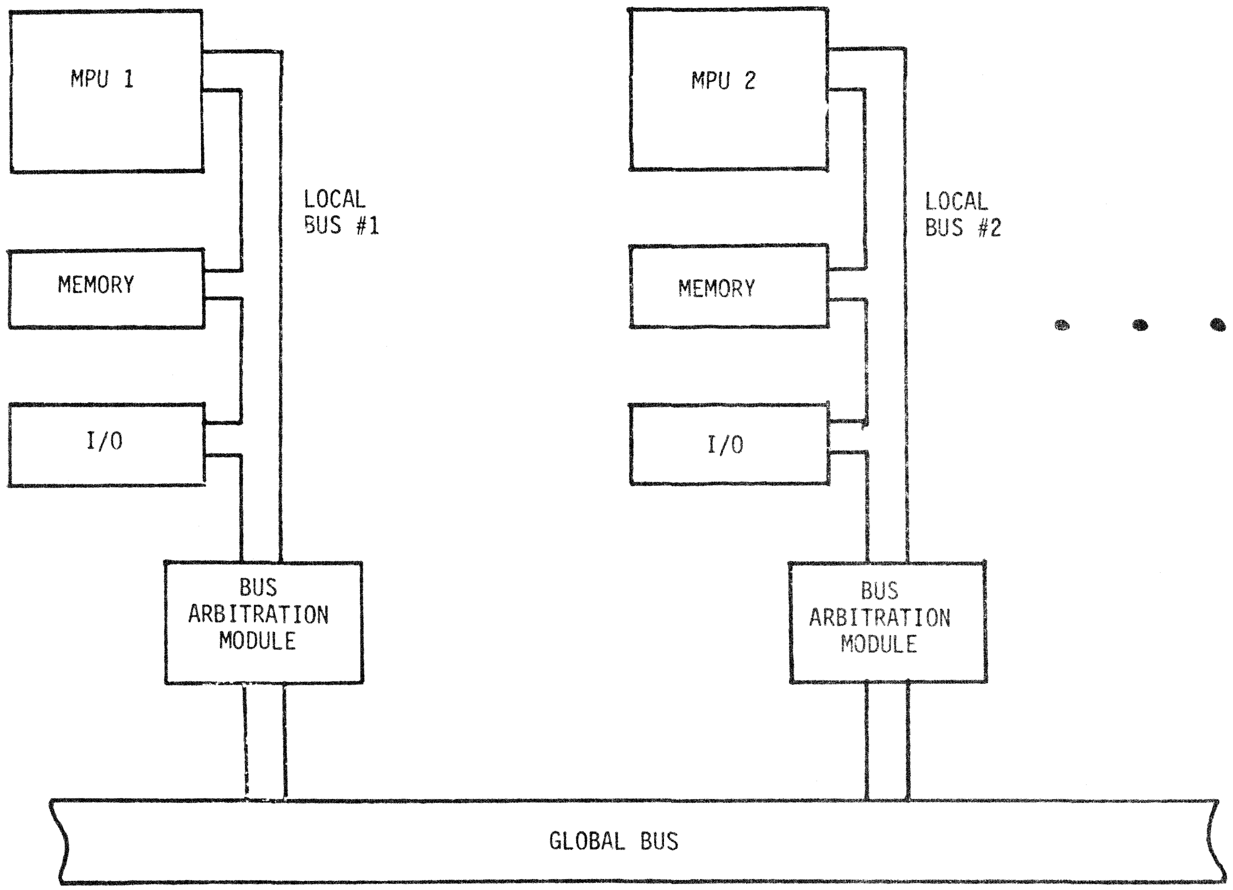
PPU - PERIPHERAL PROCESSOR UNIT

SYSTEM LEVEL MANAGERS:

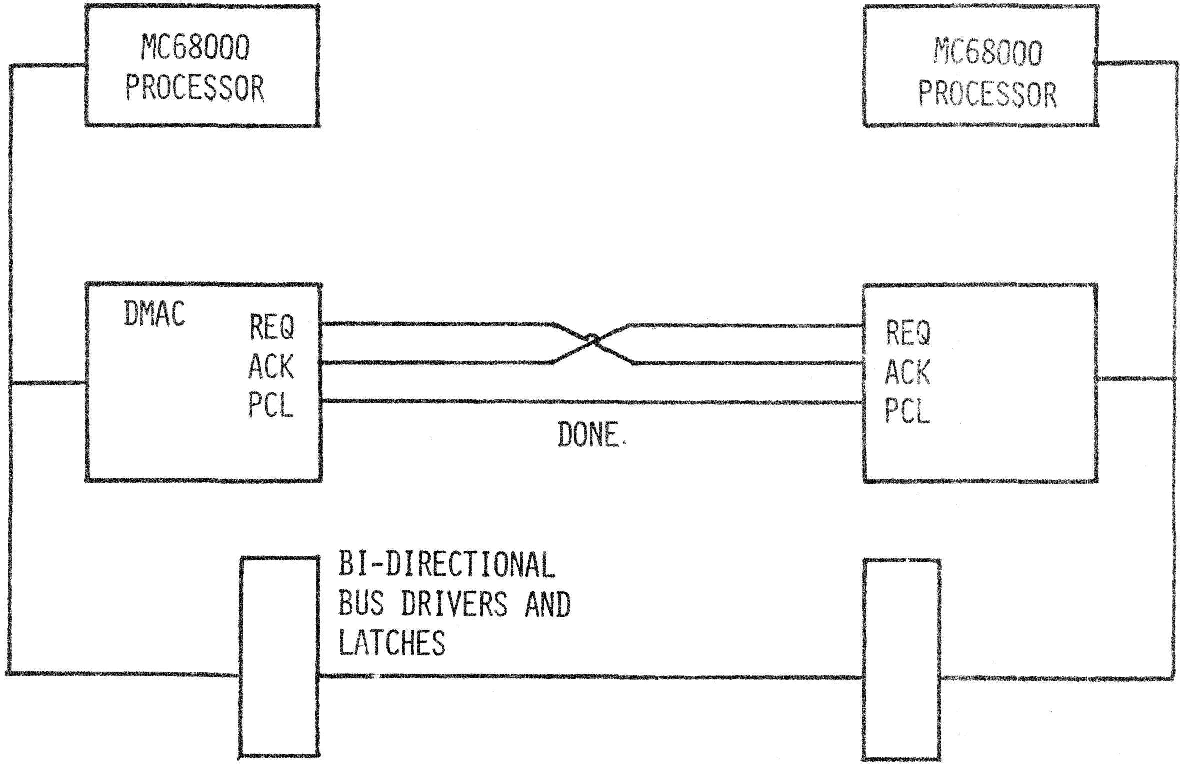
MMU - MEMORY MANAGEMENT UNIT

BAM - BUS ARBITRATION UNIT

SYSTEMS CONCEPTS



LOCAL/GLOBAL BUS SCHEME
ESTABLISHED USING BAMS



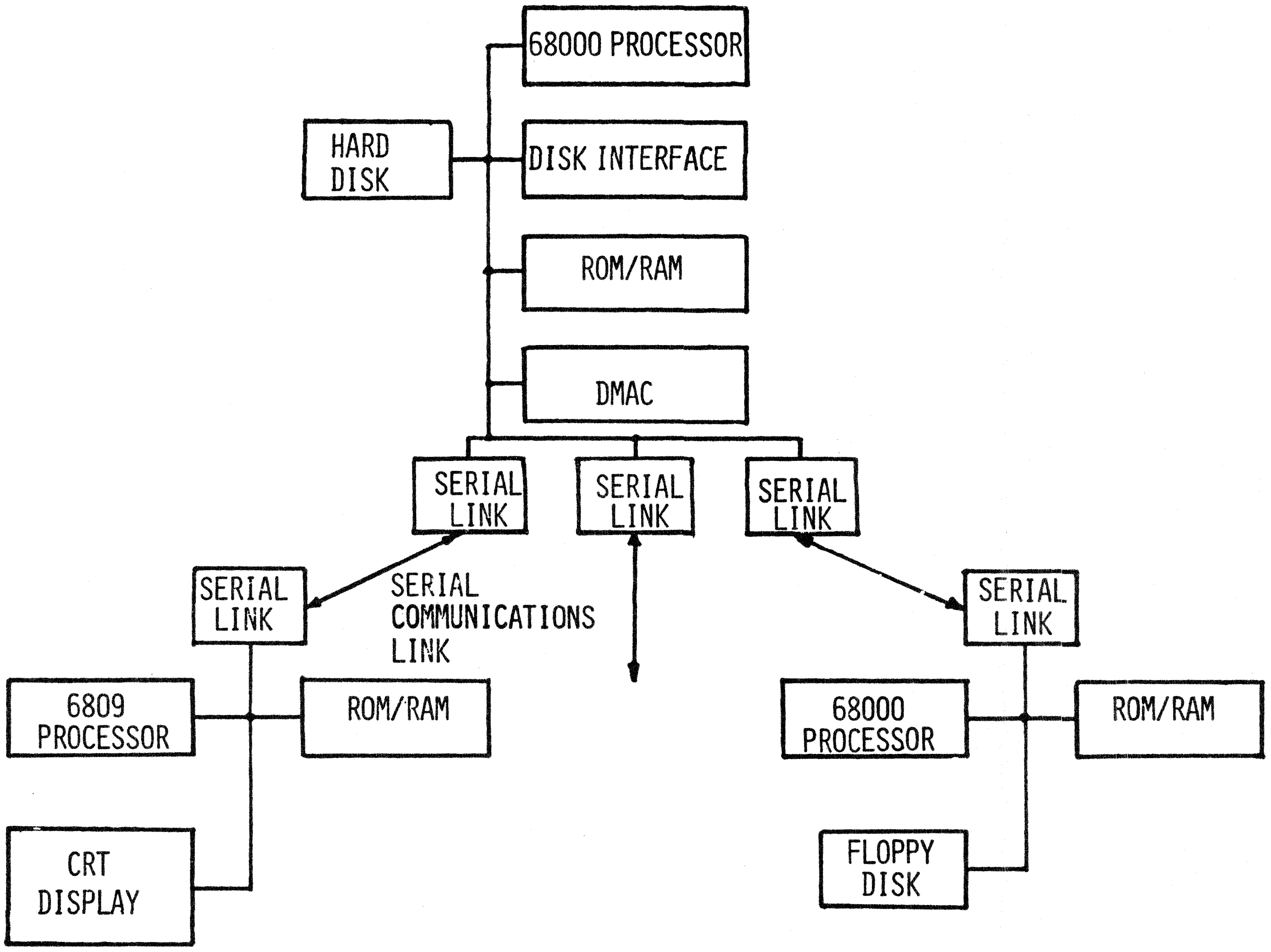
PARALLEL I/O SYSTEM
CONFIGURATION

ADVANTAGES OF PARALLEL I/O

- HIGH SPEED DATA TRANSFER
- LOW SOFTWARE OVERHEAD

DISADVANTAGES

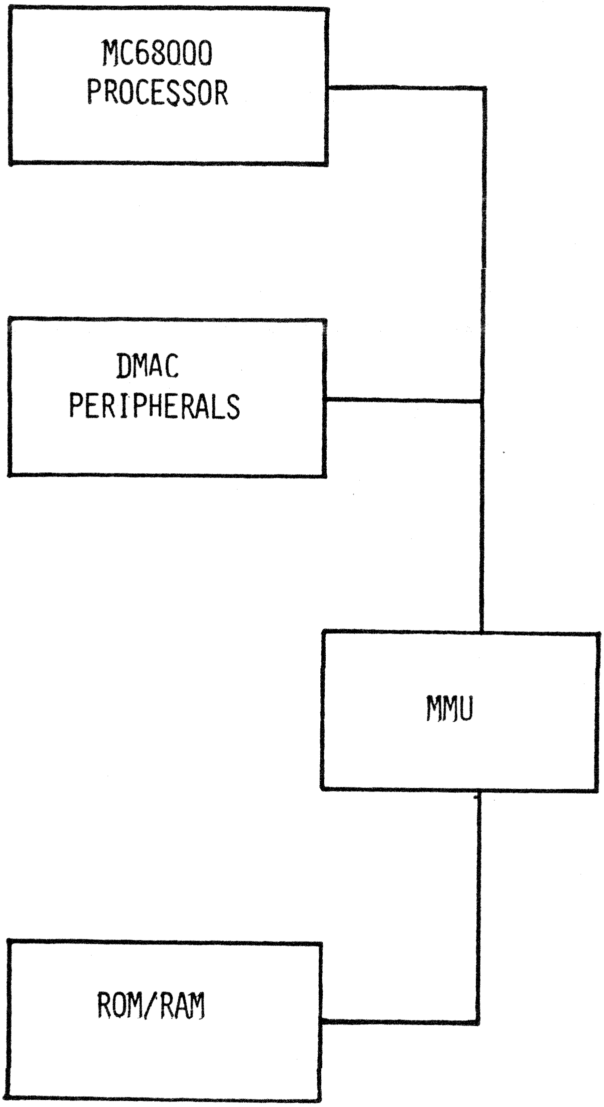
- DATA SKEWING PROBLEMS LIMIT BANDWIDTH
- MULTIPLE LINES FOR INTERCONNECTION
- HIGH CURRENT DRIVERS REQUIRED
- USUALLY USED ONLY FOR SHORT DISTANCES
- MULTIDROP CONFIGURATION IS HARD TO CONFIGURE
- FULL-DUPLEX OPERATION DOUBLES HARDWARE OVERHEAD



SERIAL I/O SYSTEMS
CONFIGURATIONS

ADVANTAGES OF SERIAL I/O

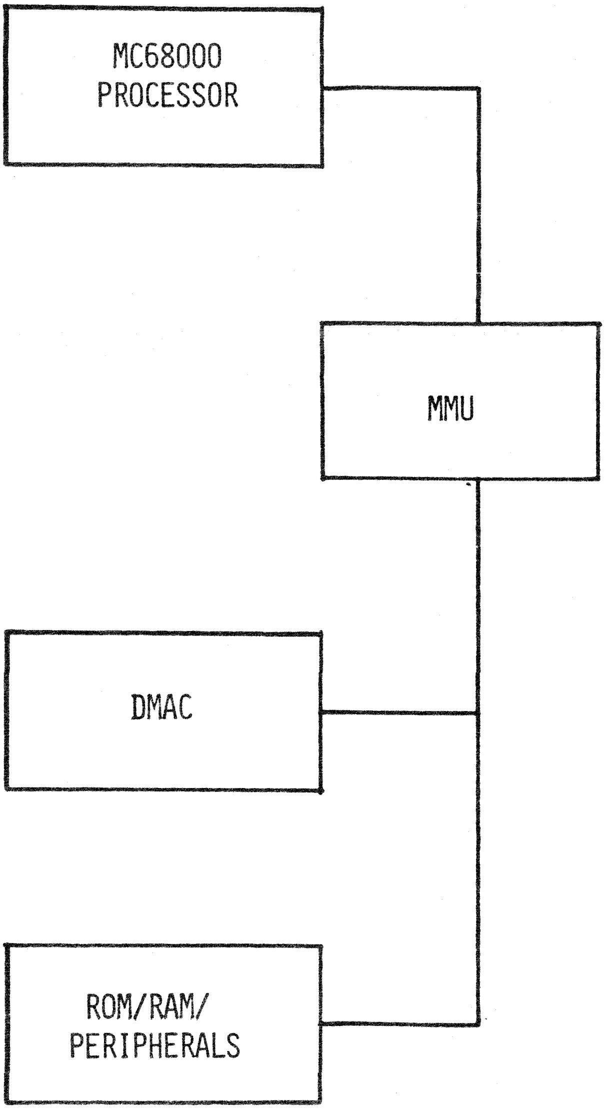
- SIMPLE INTERCONNECTION
- NO DATA SKEWING PROBLEMS
- MULTIDROP CONFIGURATIONS ARE SIMPLE TO CONNECT
- LONG LINES ARE EASY TO ACCOMMODATE.
- LOWER OVERHEAD FOR FULL-DUPLEX FOR SERIAL THAN PARALLEL I/O.
- LOWER BANDWIDTH.



DMA AND PERIPHERALS
UPSTREAM OF MMU

FEATURES OF "DOWNSTREAM" ORGANIZATION

- RELOCATABLE DMA CHANNELS AND PERIPHERALS
- OS CAN "GIVE AWAY" DMA CHANNELS AND PERIPHERALS TO USER TASKS.
- DMA CHANNELS AND PERIPHERALS CAN BE MAPPED SO THAT THEY ARE A CONTIGUOUS PART OF USER MEMORY MAP.
- DMA AND PERIPHERALS ARE PROTECTABLE RESOURCES.
- ONLY PHYSICAL ADDRESSES CAN BE TRANSFERRED WITH DMA-- LOGICAL ADDRESSES MUST FIRST BE CONVERTED TO PHYSICAL ADDRESSES.
- RESOURCE PROTECTION MUST BE DONE BY THE OS TO PROTECT AGAINST DMAC MIS-USE.



DMAC AND PERIPHERALS
DOWNSTREAM OF MMU

FEATURES OF "UPSTREAM" ORGANIZATION

- FIXED PERIPHERAL AND DMA CHANNEL LOCATION.

- DMA CAN BE SET UP WITH LOGICAL ADDRESSES RATHER THAN PHYSICAL ADDRESSES. USER TASKS DO NOT HAVE TO PERFORM ADDRESS CONVERSIONS.

- DMA TRANSFERS ARE CHECKED FOR VALIDITY BEFORE THEY TAKE PLACE.

- OS MUST PROTECT RESOURCES IN SOFTWARE.

THE MCXXX

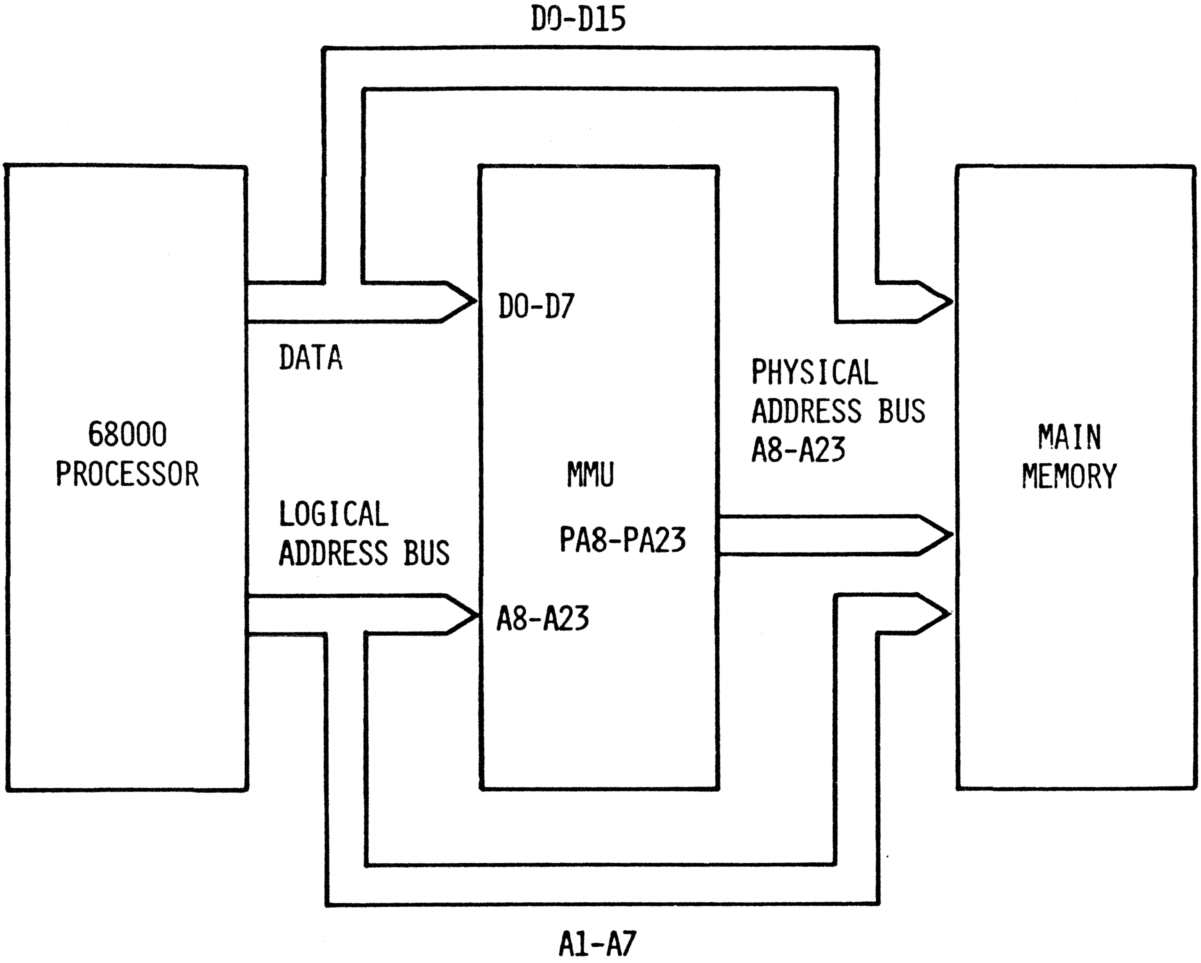
MEMORY MANAGEMENT UNIT

WHY MEMORY MANAGEMENT?

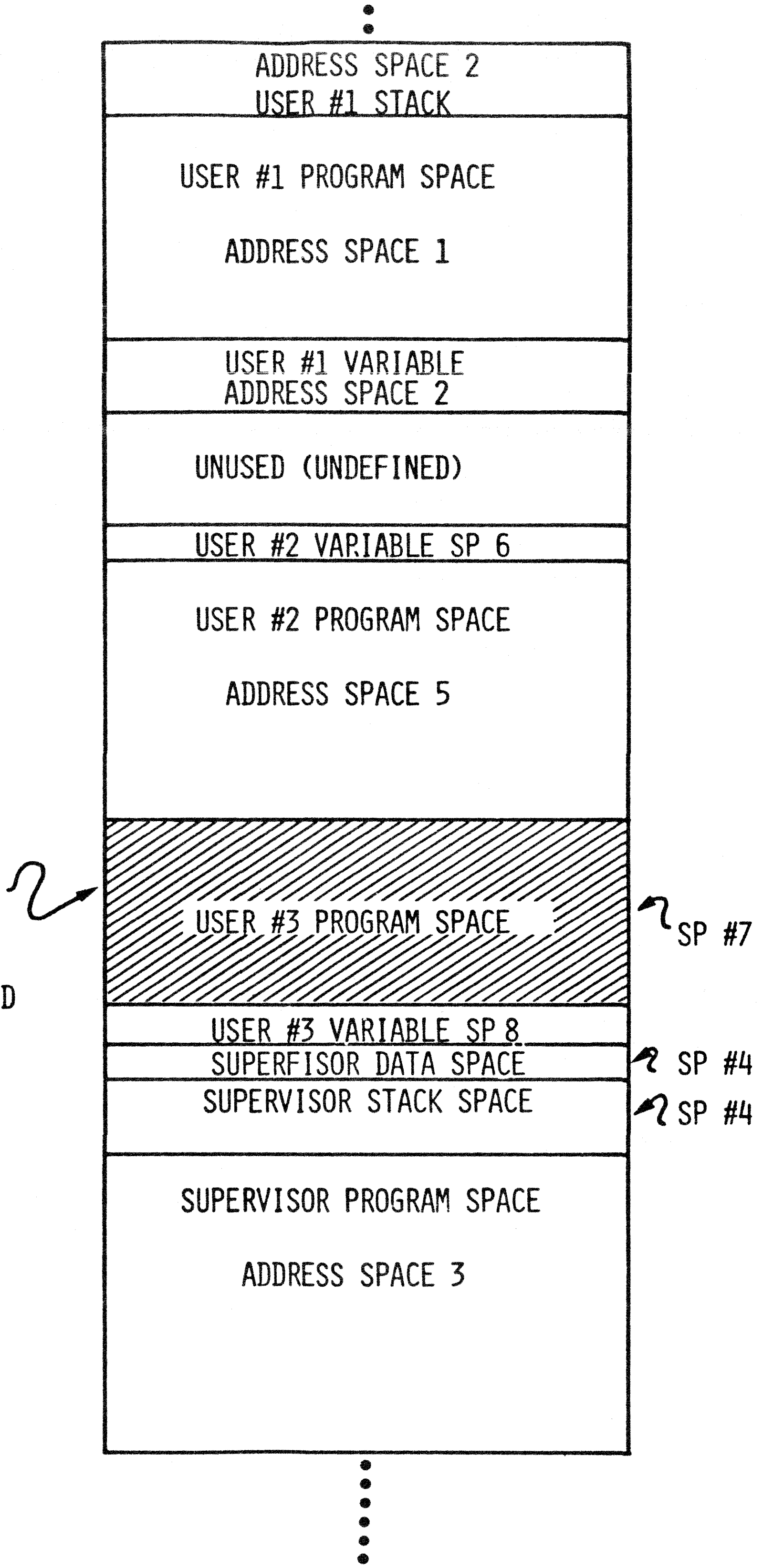
- PROVIDE ADDRESS TRANSLATION TO SIMPLIFY PROGRAMMING MODEL OF ADDRESS SPACE.
- PROVIDE PROTECTION OF DIFFERENT MEMORY AREAS FROM UNAUTHORIZED ACCESS.
- ALLOW FOR SEGMENTS OF MEMORY TO BE SHARED AMONG USERS (TASKS).
- PROVIDE TRANSLATION AND PROTECTION FOR USE WITH DMA OPERATIONS.
- PROVIDE SIMPLE CONTEXT SWITCHING.

MEMORY MANAGEMENT UNIT

- CREATES LOGICAL AND PHYSICAL ADDRESS BUS $\pm 2^{15}$
- PROVIDES INTERFACE NECESSARY FOR VIRTUAL MEMORY SYSTEMS
- SUPPORTS 8 SEGMENTS lower and upper pointer n (256 bytes)
- PROVIDES PROTECTION OF SEGMENTS
- PROVIDES FOR FAST CONTEXT SWITCHES
- PROVIDES ADDRESS TRANSLATION
- CASCADABLE TO ALLOW ANY NUMBER OF SEGMENTS
- BUS COMPATIBLE WITH THE MC68000 FAMILY
- 64-PIN FAMILY SYTLE PACKAGE



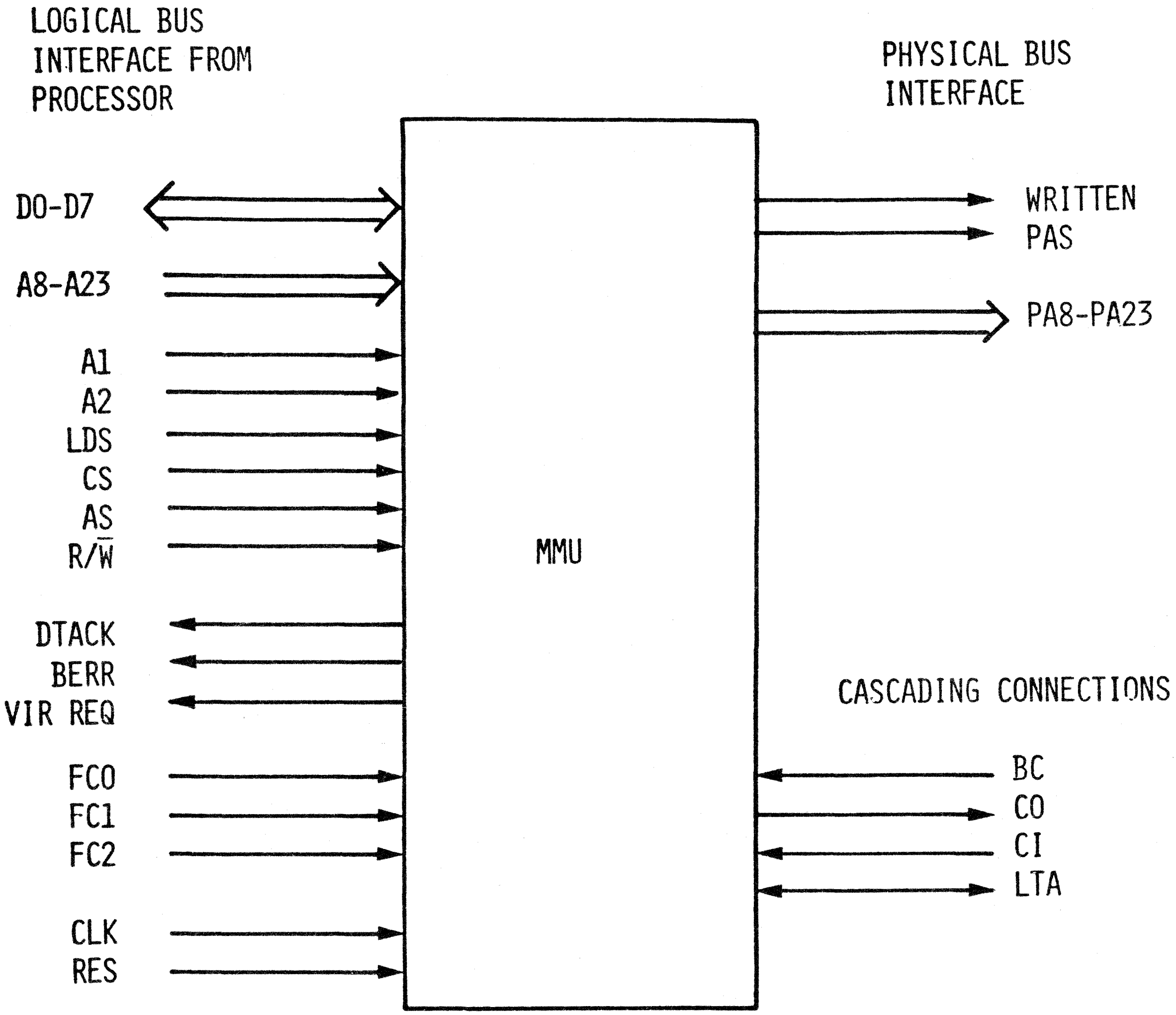
MMU SYSTEM CONFIGURATION



DEFINED BUT NOT
RESIDENT

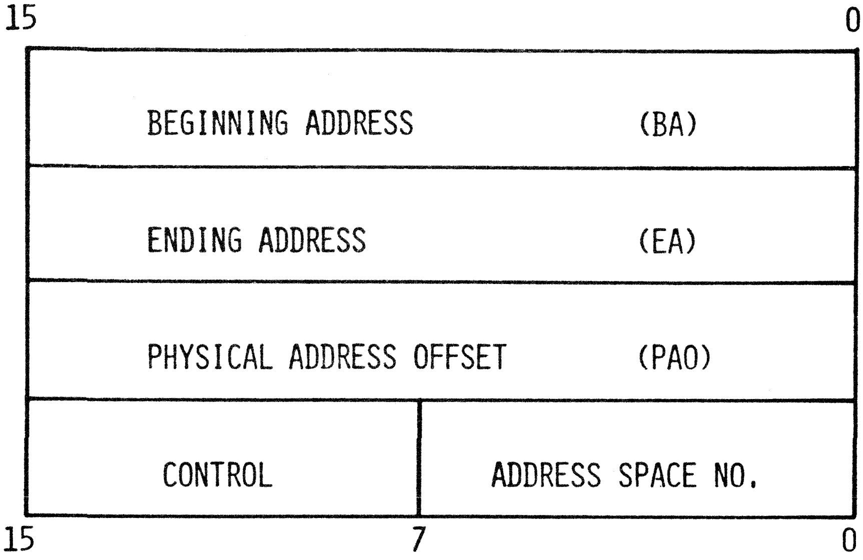
VIR-REQ GENERATED
WHEN ACCESSED

HYPOTHETICAL
MEMORY MAP


LOGICAL BUS
INTERFACE FROM
PROCESSOR
PHYSICAL BUS
INTERFACE
D0-D7
A8-A23
A1
A2
LDS
CS
AS
R/$\overline{W}$
DTACK
BERR
VIR REQ
FC0
FC1
FC2
CLK
RES
MMU
WRITTEN
PAS
PA8-PA23
CASCADING CONNECTIONS
BC
CO
CI
LTA

FUNCTION CODE	TASK TABLE
0 RESERVED	
1 USER PROGRAM	ACTIVE ADDRESS SPACE (S) FOR USER PROGRAM
2 USER DATA	ACTIVE ADDRESS SPACE (S) FOR USER DATA
3 RESERVED	
4 RESERVED	
5 SUPERVISOR PROGRAM	ACTIVE ADDRESS SPACE (S) FOR SUPERVISOR PROGRAM
6 SUPERVISOR DATA	ACTIVE ADDRESS SPACE (S) FOR SUPERVISOR DATA
7 INTERRUPT ACK	▨

ONE SEGMENT OUT OF EIGHT



1. AN ADDRESS SPACE NUMBER EQUAL TO ZERO DISABLES THE SEGMENT.

2. WRITING INTO BA, EA, PAO CLEARS THE AS NUMBER REGISTER.

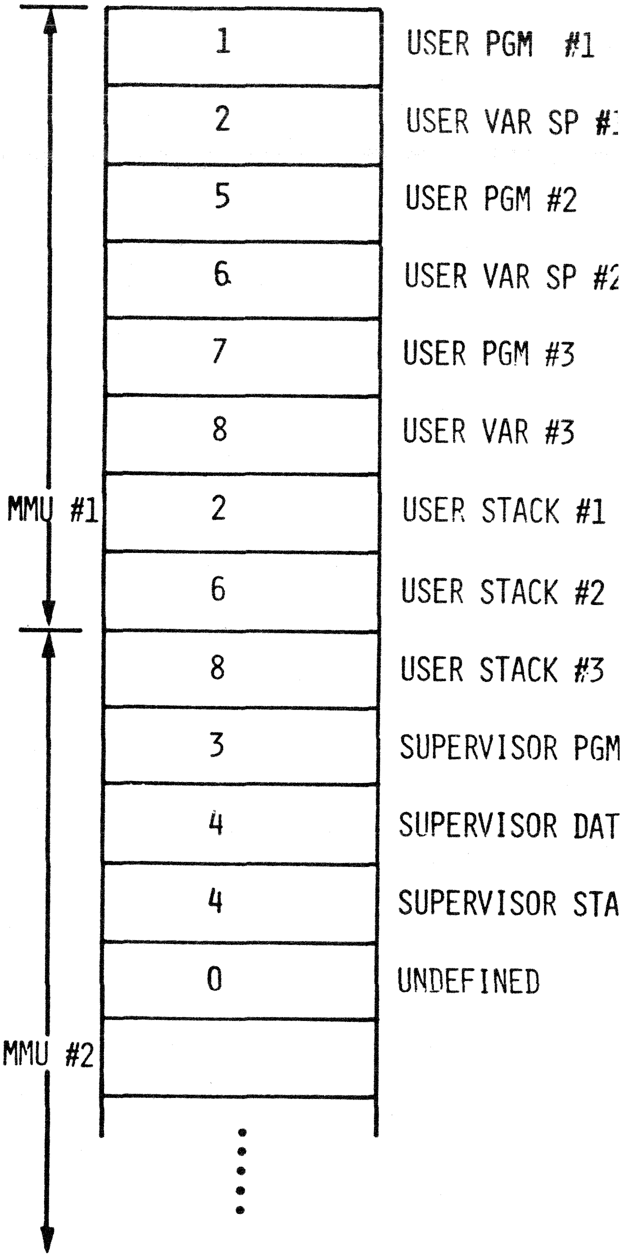
FUNCTION CODE TASK TABLE

0 -	
1 USER PGM	1
2 USER ~~PGM~~ DATA	2
3 -	
4 -	
5 SUP PGM	3
6 SUP DATA	4
7 IAK	▨

CURRENTLY EXECUTING USER #1's PROGRAM

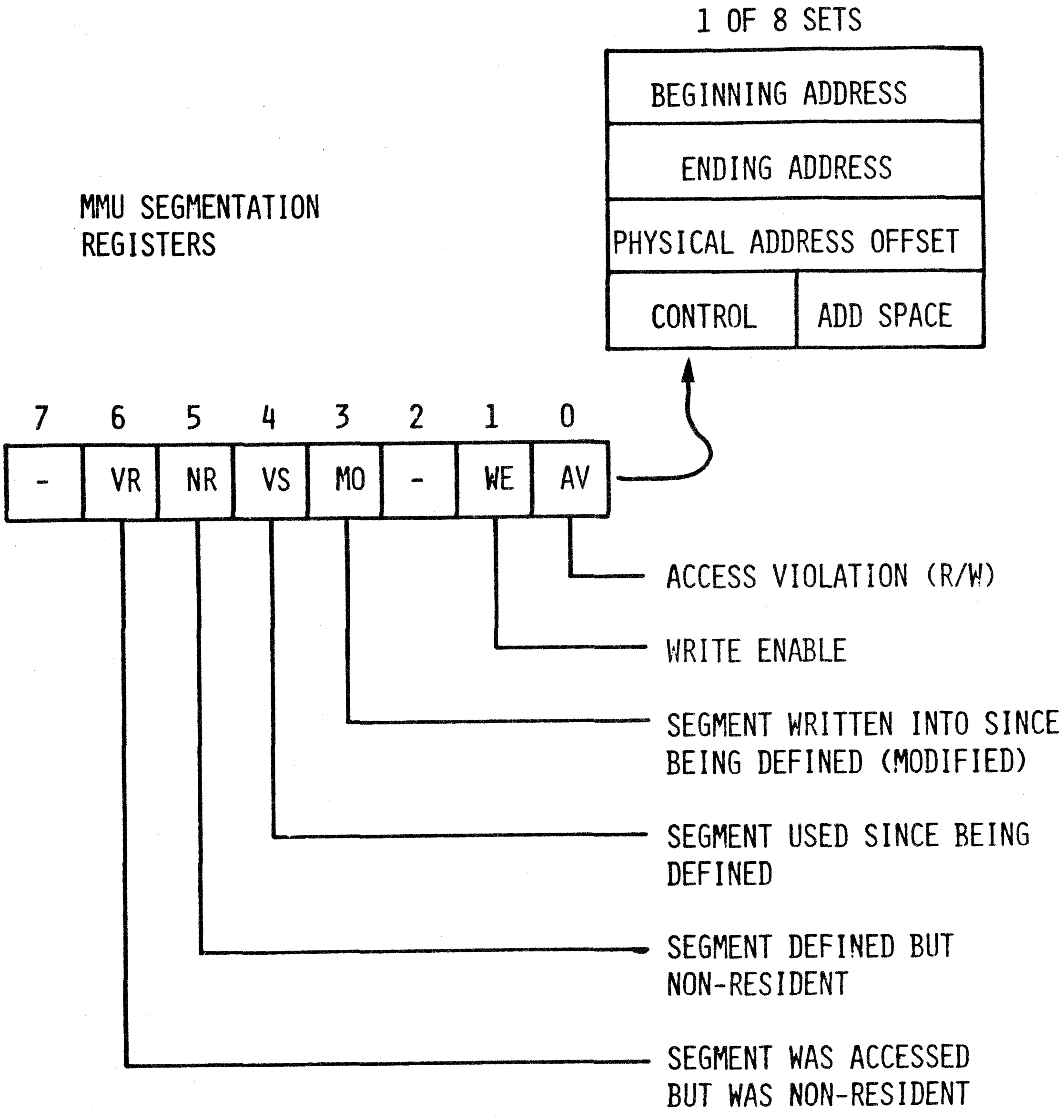
TO CHANGE CONTEXT FOR EXECUTION OF USER #2's PROGRAM,- USER PROGRAM TASK NUMBER WOULD BE SET TO 5, AND USER DATA TASK NUMBER WOULD BE SET TO 6.

ADDRESS SPACES
(MMU INTERNAL REGISTERS)



MMU #1	1	USER PGM #1
	2	USER VAR SP #1
	5	USER PGM #2
	6	USER VAR SP #2
	7	USER PGM #3
	8	USER VAR #3
	2	USER STACK #1
	6	USER STACK #2
MMU #2	8	USER STACK #3
	3	SUPERVISOR PGM
	4	SUPERVISOR DAT
	4	SUPERVISOR STA
	0	UNDEFINED

MMU SEGMENTATION REGISTERS


1 OF 8 SETS
BEGINNING ADDRESS
ENDING ADDRESS
PHYSICAL ADDRESS OFFSET
CONTROL
ADD SPACE
7
6
5
4
3
2
1
0
-
VR
NR
VS
MO
-
WE
AV
ACCESS VIOLATION (R/W)
WRITE ENABLE
SEGMENT WRITTEN INTO SINCE BEING DEFINED (MODIFIED)
SEGMENT USED SINCE BEING DEFINED
SEGMENT DEFINED BUT NON-RESIDENT
SEGMENT WAS ACCESSED BUT WAS NON-RESIDENT

- OPERATING SYSTEM ORIENTED FEATURES:

  DESCRIPTION OF STATUS BITS

- ACCESS VIOLATION:

  IS SET WHEN A WRITE IS ATTEMPTED IN A "READ-ONLY" SEGMENT.

- WRITE ENABLE:

  WHEN SET TO ZERO SEGMENT IS DEFINED AS "READ-ONLY".

  WHEN SET TO ONE SEGMENT IS DEFINED AS READ/WRITE

- MODIFIED:

  SET WHEN A SEGMENT HAS BEEN WRITTEN INTO.

  USED BY THE OPERATING SYSTEM TO KEEP TRACK OF WHICH SEGMENTS MUST BE UPDATED IN VIRTUAL MEMORY.

- USED:

  SET WHEN SEGMENT HAS BEEN ACCESSED.

  USED BY OPERATING SYSTEM FOR "GARBAGE COLLECTION", BY PURGING THE SEGMENTS WHICH HAVE NOT BEEN RECENTLY USED.

- NON-RESIDENT:

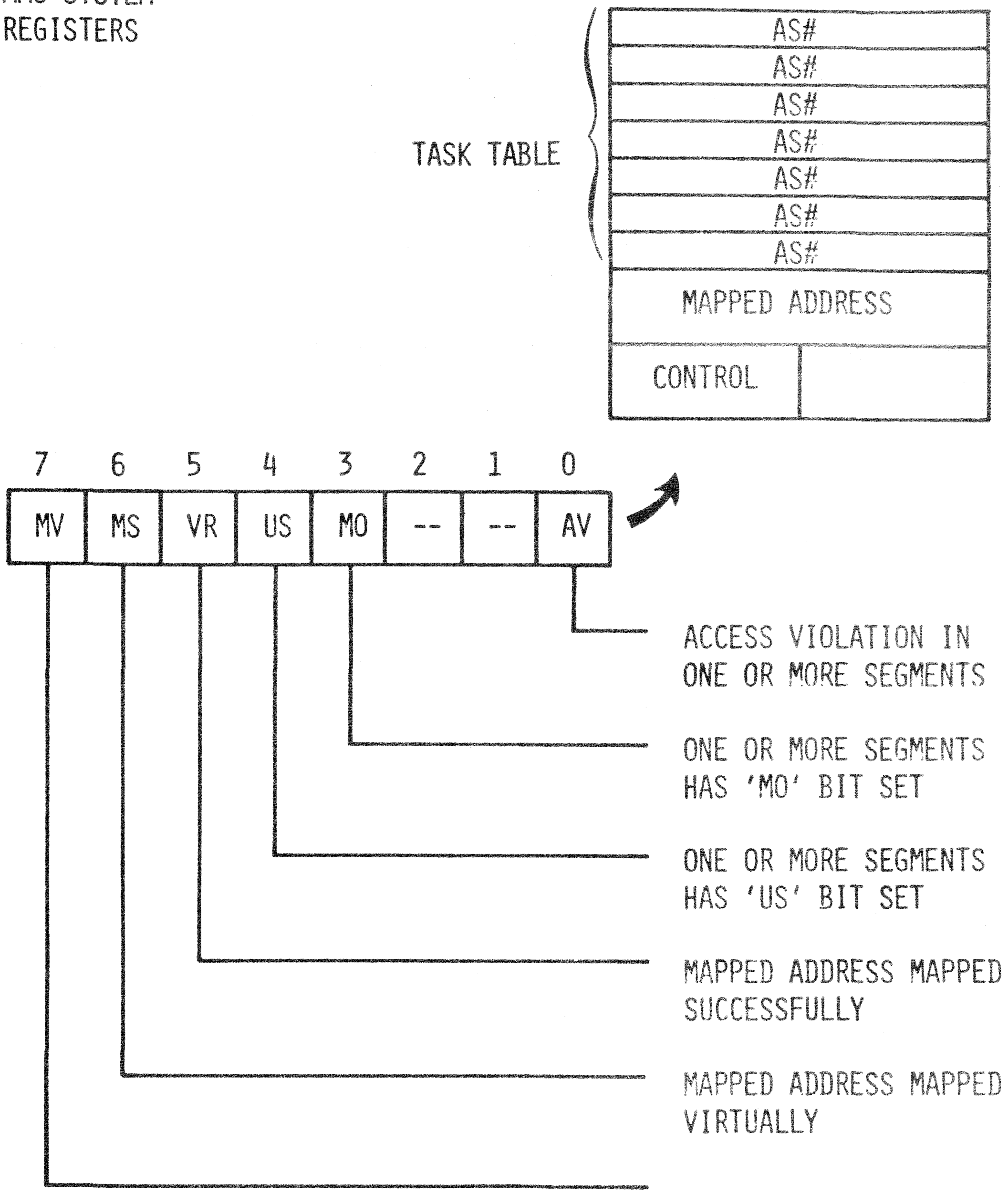
  SET WHEN THE OPERATING SYSTEM DEFINES THE SEGMENT.

  RESET BY OPERATING SYSTEM WHEN SEGMENT IS MADE RESIDENT.

- VIRTUAL-REQUEST:

  SET WHEN AN ACCESS IS MADE TO A SEGMENT WITH ITS 'NR' BIT SET. INDICATES THAT DMA ACTION SHOULD BRING THE SEGMENT IN.

MMU SYSTEM REGISTERS



TASK TABLE

AS#

AS#

AS#

AS#

AS#

AS#

AS#

MAPPED ADDRESS

CONTROL

7	6	5	4	3	2	1	0
MV	MS	VR	US	MO	--	--	AV

ACCESS VIOLATION IN ONE OR MORE SEGMENTS

ONE OR MORE SEGMENTS HAS 'MO' BIT SET

ONE OR MORE SEGMENTS HAS 'US' BIT SET

MAPPED ADDRESS MAPPED SUCCESSFULLY

MAPPED ADDRESS MAPPED VIRTUALLY

- MAPPED SUCCESSFULLY:

  SET WHEN LOGICAL ADDRESS HAS BEEN BOUND CHECKED AND HAS SUCCESSFULLY BEEN MAPPED INTO ONE SEGMENT. THE MAPPED ADDRESS IS VALID IN THE 'MAPPED ADDRESS' REGISTER.

- MAPPED VIRTUALLY:

  SET WHEN LOGICAL ADDRESS MAPPED INTO A SEGMENT WHICH IS DEFINED, BUT NON-RESIDENT.

THE MC68XXX

DIRECT MEMORY ACCESS CONTROLLER

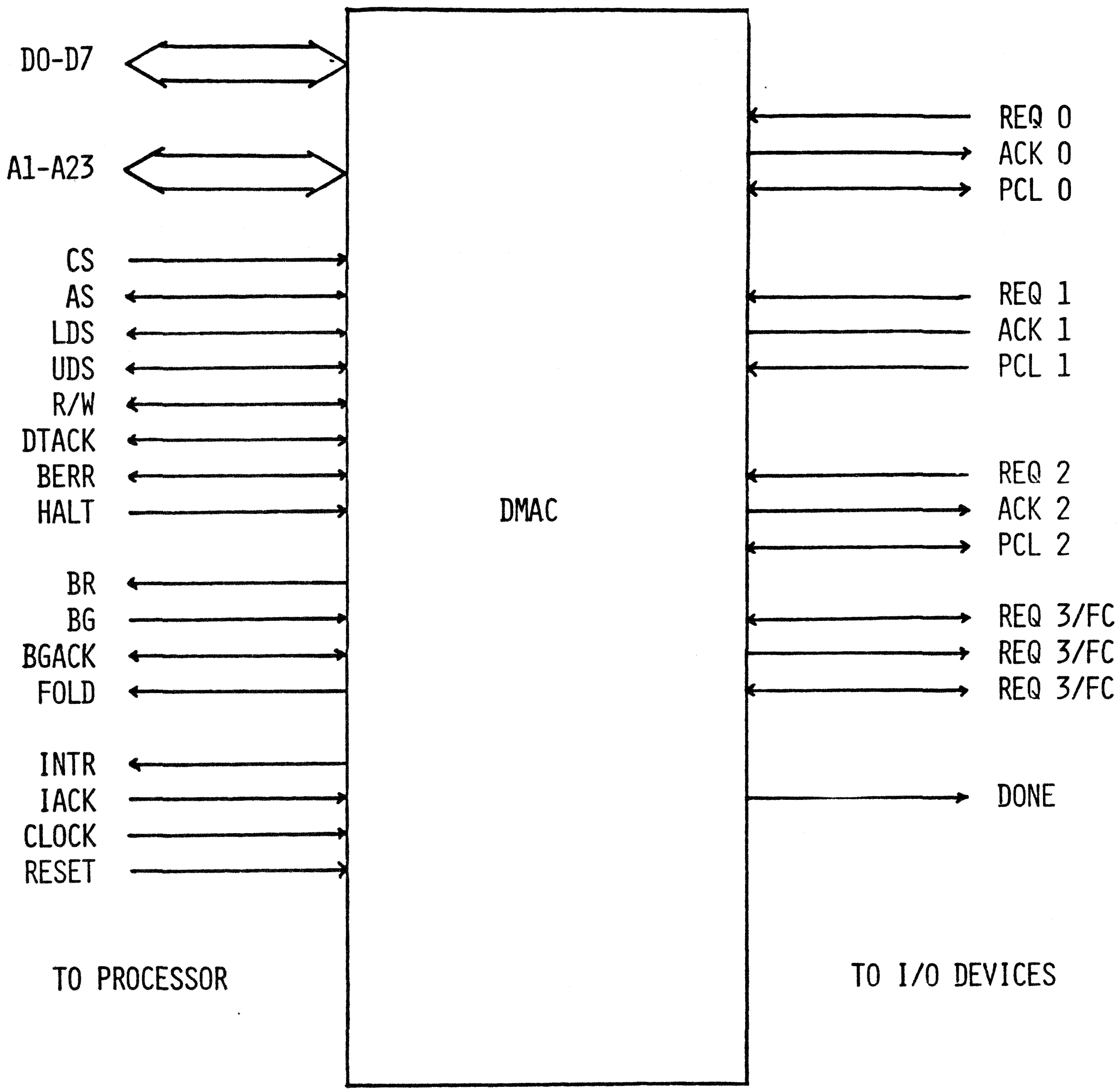
WHY HAVE DMA?

- WIDEN THE BANDWIDTH OF I/O CHANNELS.
- TO PERFORM MEMORY-TO-MEMORY BLOCK TRANSFERS FASTER AND WITH LESS OVERHEAD THAN THE PROCESSOR COULD.

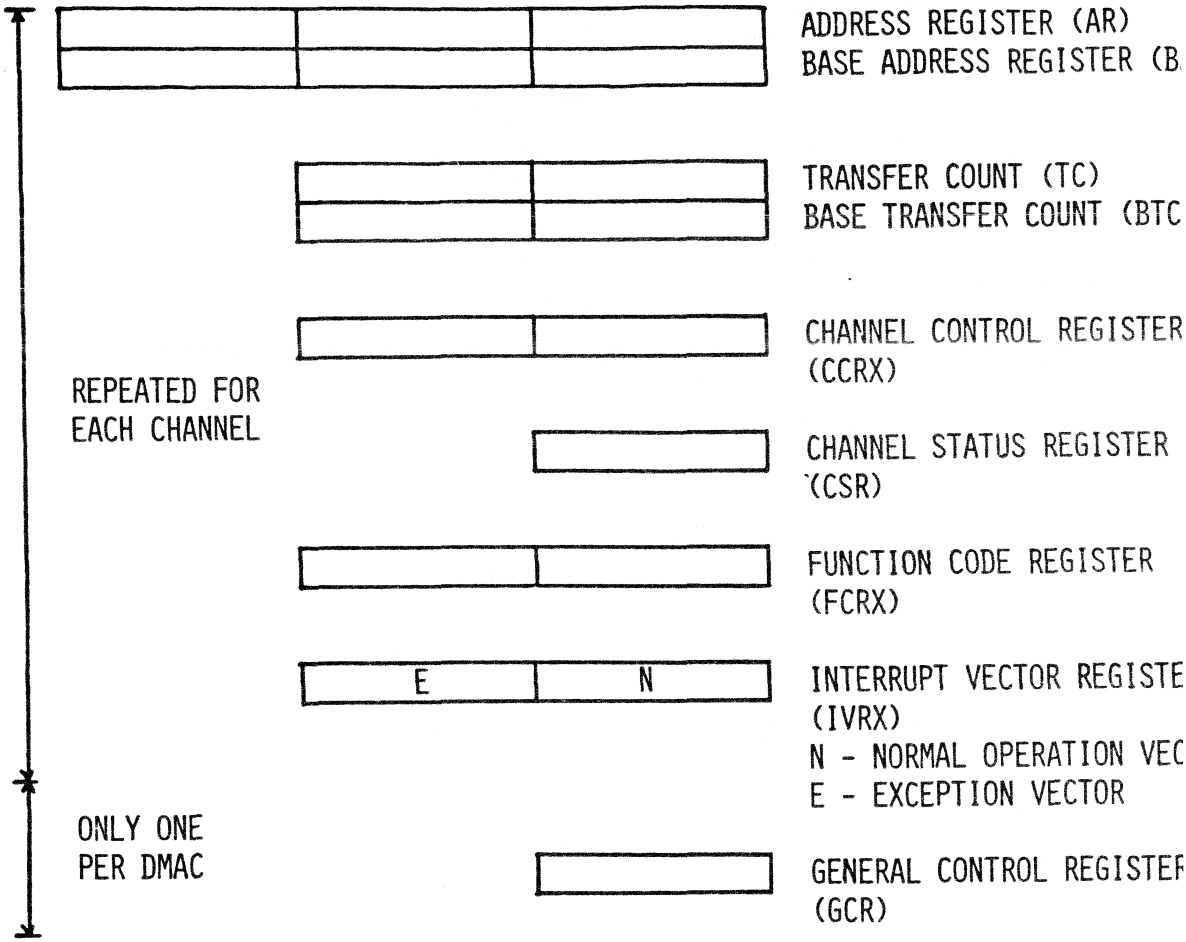
DIRECT MEMORY ACCESS CONTROLLER

FEATURES:

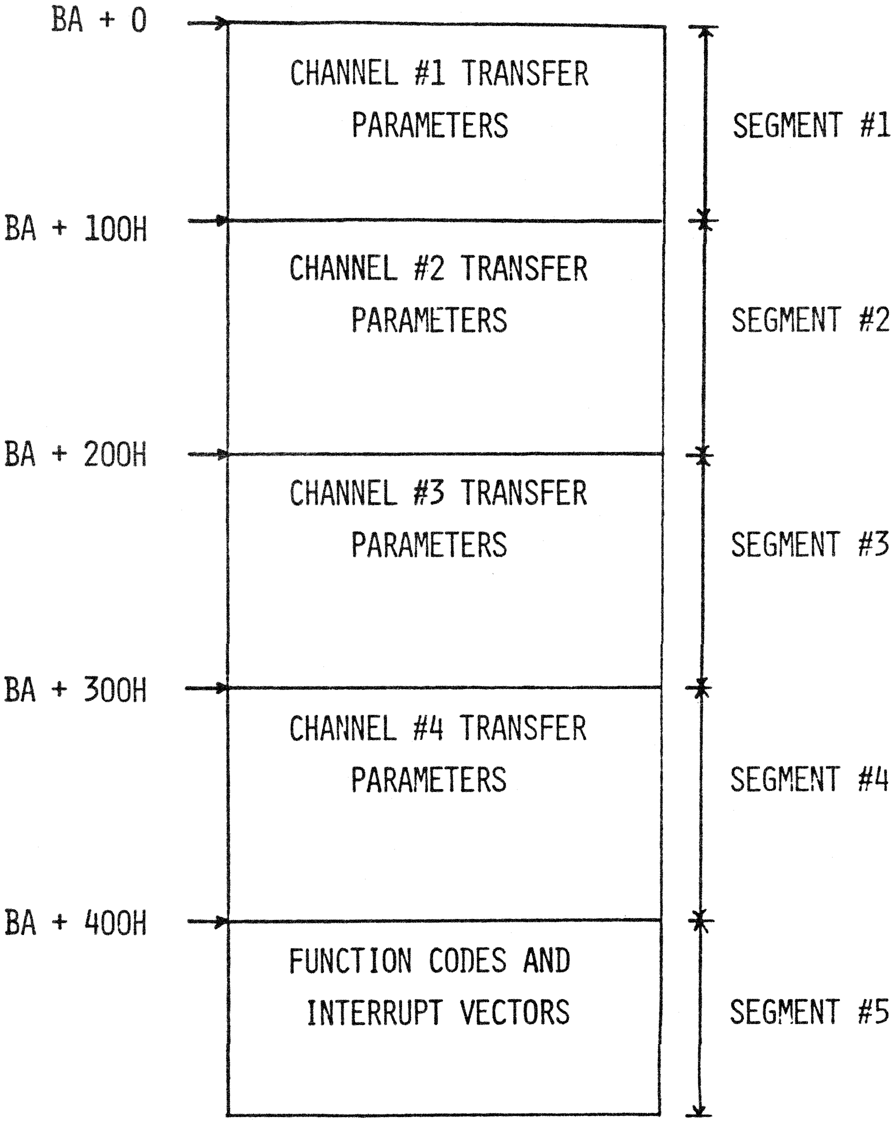
- FOUR CHANNEL DMA CONTROLLER
- 16 MBYTE ADDRESSING RANGE AND 65K BYTE COUNT RANGE
- MEMORY TO MEMORY BLOCK TRANSFERS
- FIXED AND ROTATING PRIORITIES
- 3 DATA CHAINING MODES AVAILABLE
- BUS COMPATIBLE WITH THE MC68000
- TWO INTERRUPT VECTORS FOR EACH CHANNEL
- BYTE, WORD, AND LONGWORD MODES
- 64-PIN FAMILY STYLE PACKAGE


D0-D7
A1-A23
CS
AS
LDS
UDS
R/W
DTACK
BERR
HALT
BR
BG
BGACK
FOLD
INTR
IACK
CLOCK
RESET
DMAC
REQ 0
ACK 0
PCL 0
REQ 1
ACK 1
PCL 1
REQ 2
ACK 2
PCL 2
REQ 3/FC
REQ 3/FC
REQ 3/FC
DONE
TO PROCESSOR
TO I/O DEVICES

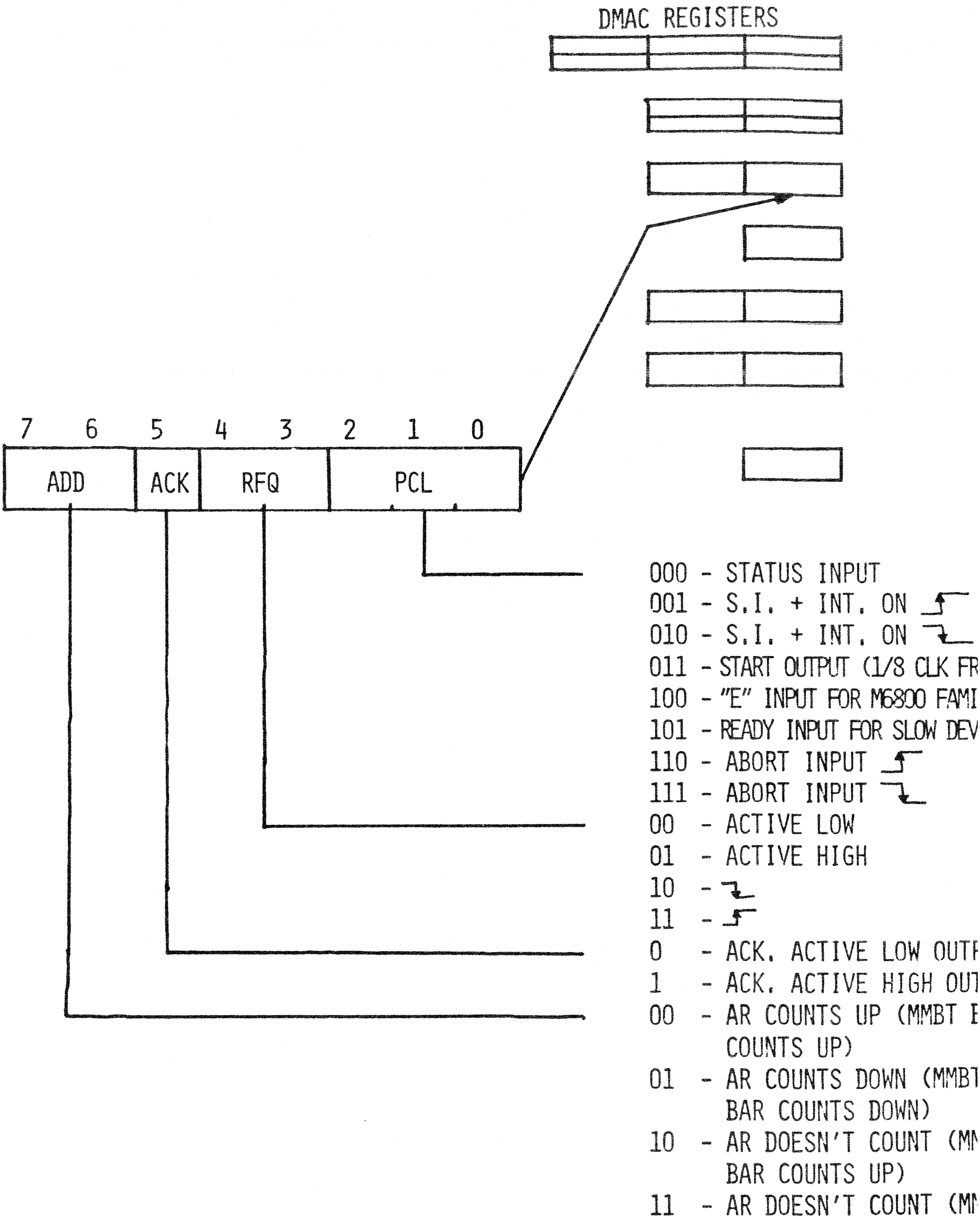
DMAC PIN-OUT



DMAC REGISTERS

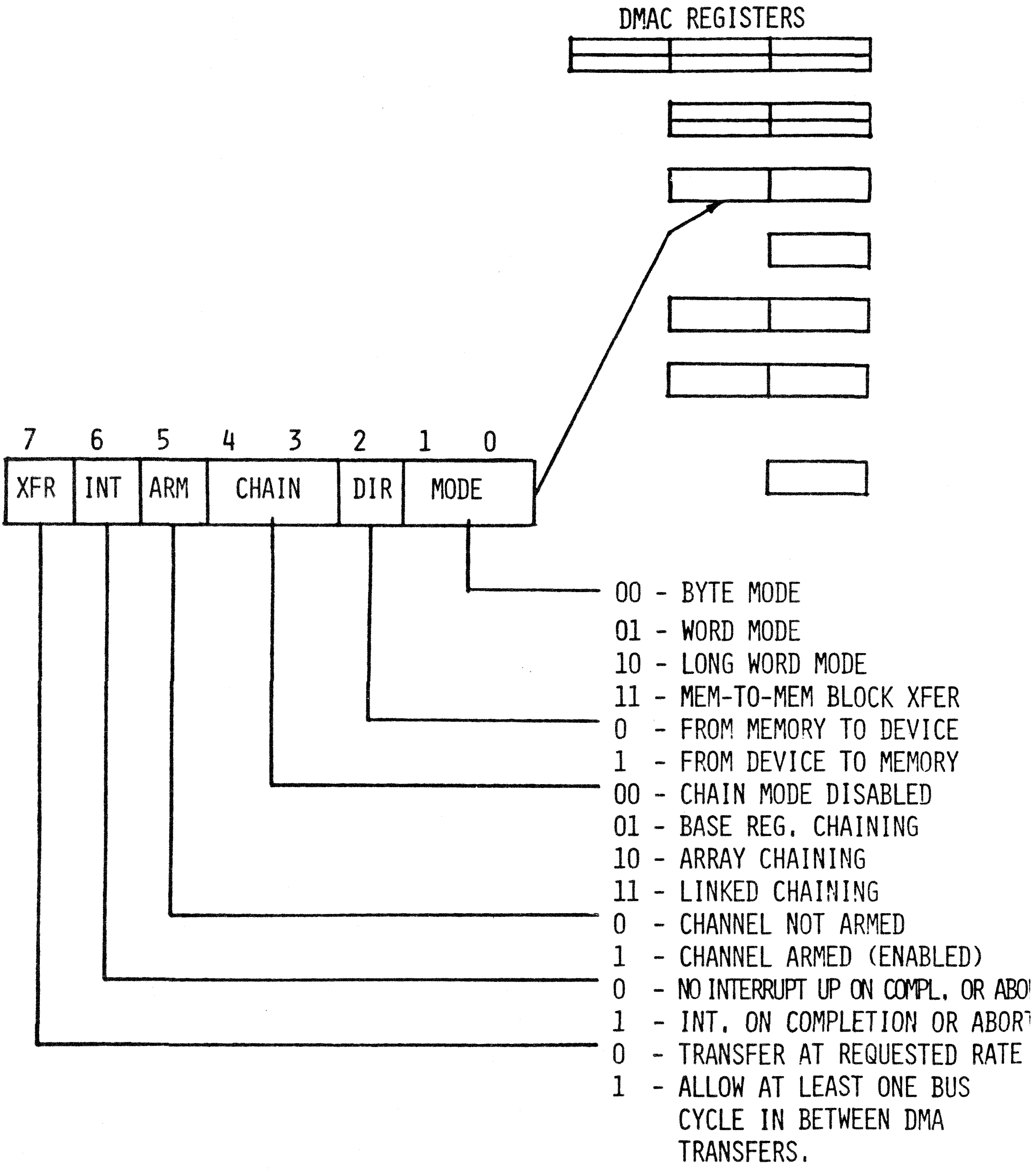

BA + 0
CHANNEL #1 TRANSFER PARAMETERS
SEGMENT #1
BA + 100H
CHANNEL #2 TRANSFER PARAMETERS
SEGMENT #2
BA + 200H
CHANNEL #3 TRANSFER PARAMETERS
SEGMENT #3
BA + 300H
CHANNEL #4 TRANSFER PARAMETERS
SEGMENT #4
BA + 400H
FUNCTION CODES AND INTERRUPT VECTORS
SEGMENT #5

DMAC MEMORY MAP



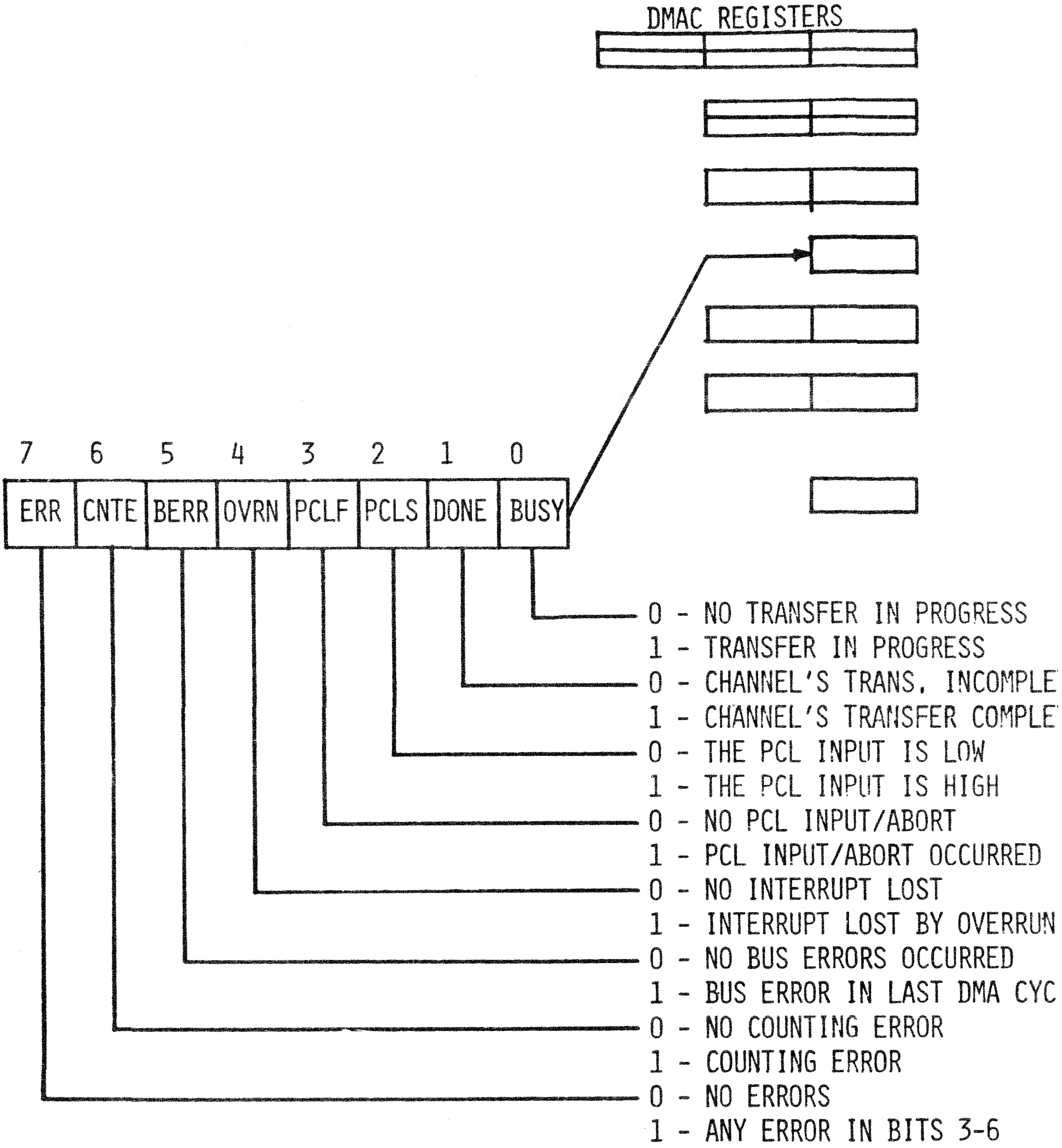
000 - STATUS INPUT
001 - S.I. + INT. ON ↑
010 - S.I. + INT. ON ↓
011 - START OUTPUT (1/8 CLK FR
100 - "E" INPUT FOR M6800 FAMI
101 - READY INPUT FOR SLOW DEV
110 - ABORT INPUT ↑
111 - ABORT INPUT ↓
00 - ACTIVE LOW
01 - ACTIVE HIGH
10 - ↓
11 - ↑
0 - ACK. ACTIVE LOW OUTP
1 - ACK. ACTIVE HIGH OUT
00 - AR COUNTS UP (MMBT I
COUNTS UP)
01 - AR COUNTS DOWN (MMBT
BAR COUNTS DOWN)
10 - AR DOESN'T COUNT (MM
BAR COUNTS UP)
11 - AR DOESN'T COUNT (MM
BAR DOESN'T COUNT)

CHANNEL CONTROL REGISTER 1 (CCR1)



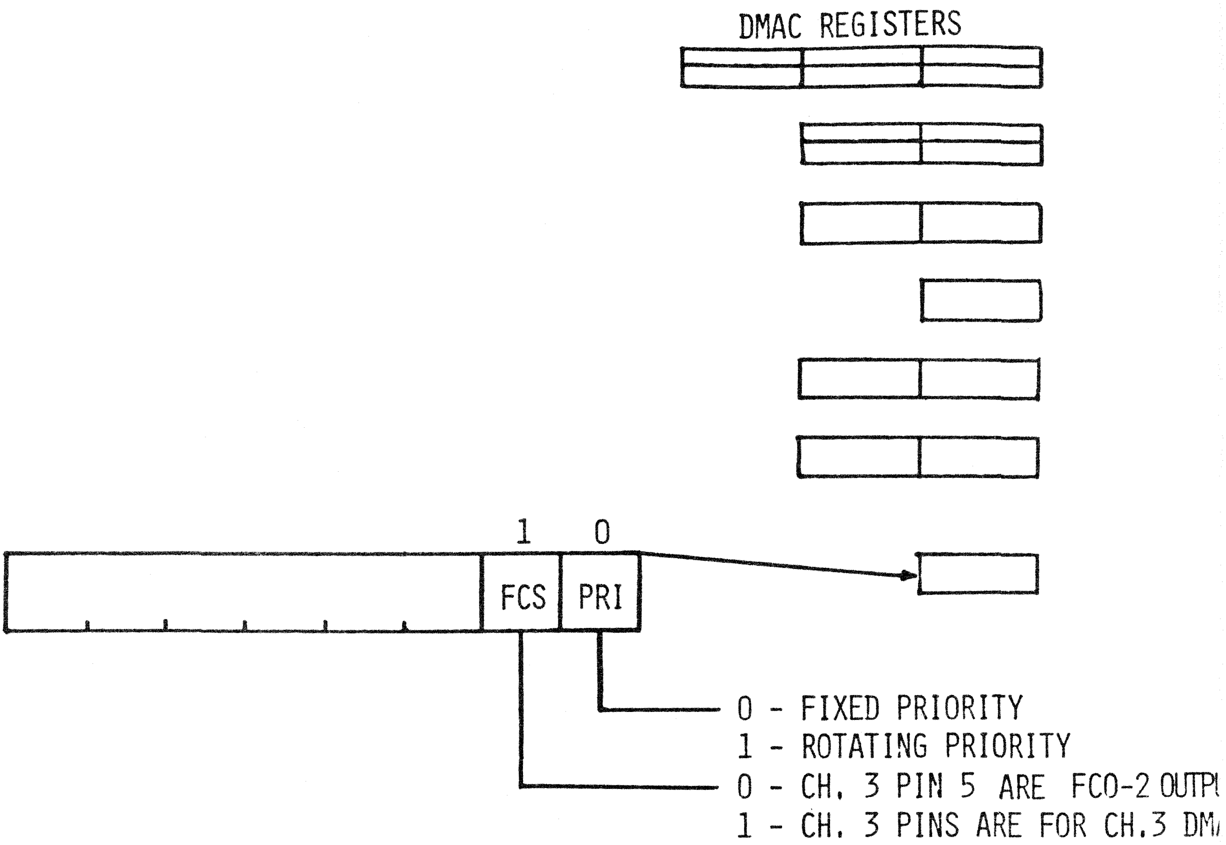
00 - BYTE MODE
01 - WORD MODE
10 - LONG WORD MODE
11 - MEM-TO-MEM BLOCK XFER
0 - FROM MEMORY TO DEVICE
1 - FROM DEVICE TO MEMORY
00 - CHAIN MODE DISABLED
01 - BASE REG. CHAINING
10 - ARRAY CHAINING
11 - LINKED CHAINING
0 - CHANNEL NOT ARMED
1 - CHANNEL ARMED (ENABLED)
0 - NO INTERRUPT UP ON COMPL. OR ABO
1 - INT. ON COMPLETION OR ABORT
0 - TRANSFER AT REQUESTED RATE
1 - ALLOW AT LEAST ONE BUS CYCLE IN BETWEEN DMA TRANSFERS.

CHANNEL CONTROL REGISTER 2 (CCR2)



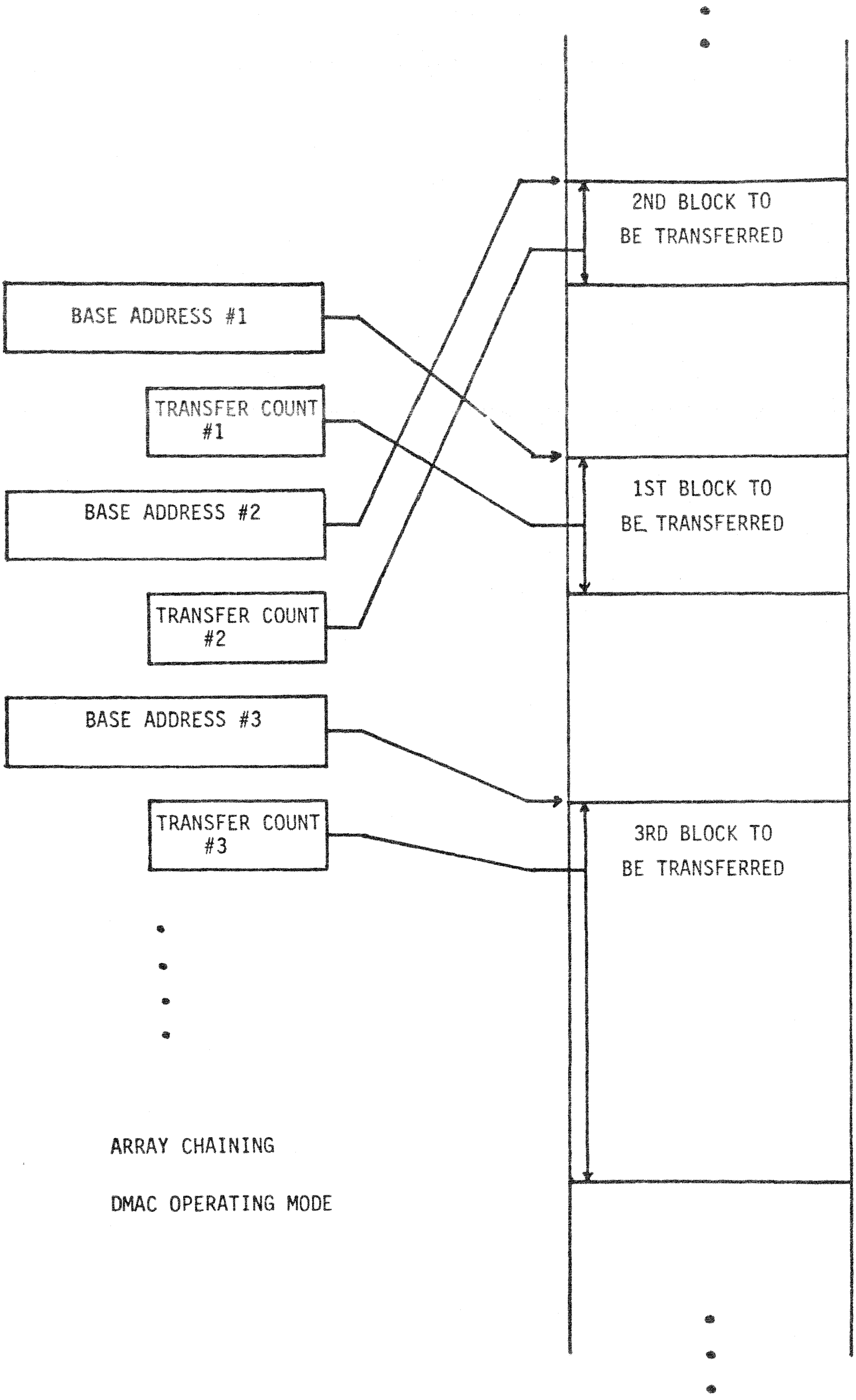
0 - NO TRANSFER IN PROGRESS
1 - TRANSFER IN PROGRESS
0 - CHANNEL'S TRANS. INCOMPLE
1 - CHANNEL'S TRANSFER COMPLE
0 - THE PCL INPUT IS LOW
1 - THE PCL INPUT IS HIGH
0 - NO PCL INPUT/ABORT
1 - PCL INPUT/ABORT OCCURRED
0 - NO INTERRUPT LOST
1 - INTERRUPT LOST BY OVERRUN
0 - NO BUS ERRORS OCCURRED
1 - BUS ERROR IN LAST DMA CYC
0 - NO COUNTING ERROR
1 - COUNTING ERROR
0 - NO ERRORS
1 - ANY ERROR IN BITS 3-6

CHANNEL STATUS REGISTER (CSR)



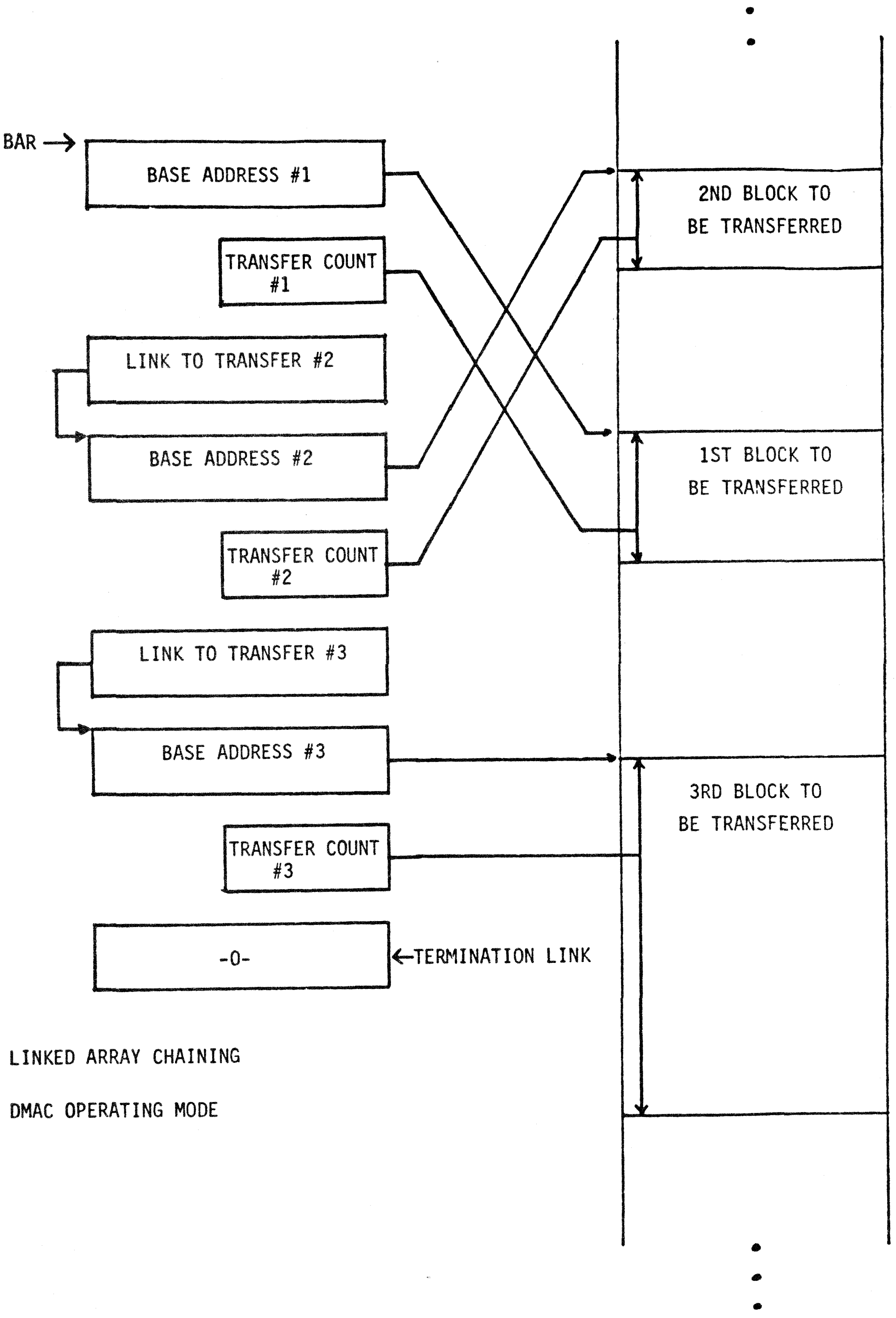
0 - FIXED PRIORITY
1 - ROTATING PRIORITY
0 - CH. 3 PIN 5 ARE FCO-2 OUTPU
1 - CH. 3 PINS ARE FOR CH.3 DM

GENERAL CONTROL REGISTER (GCR)



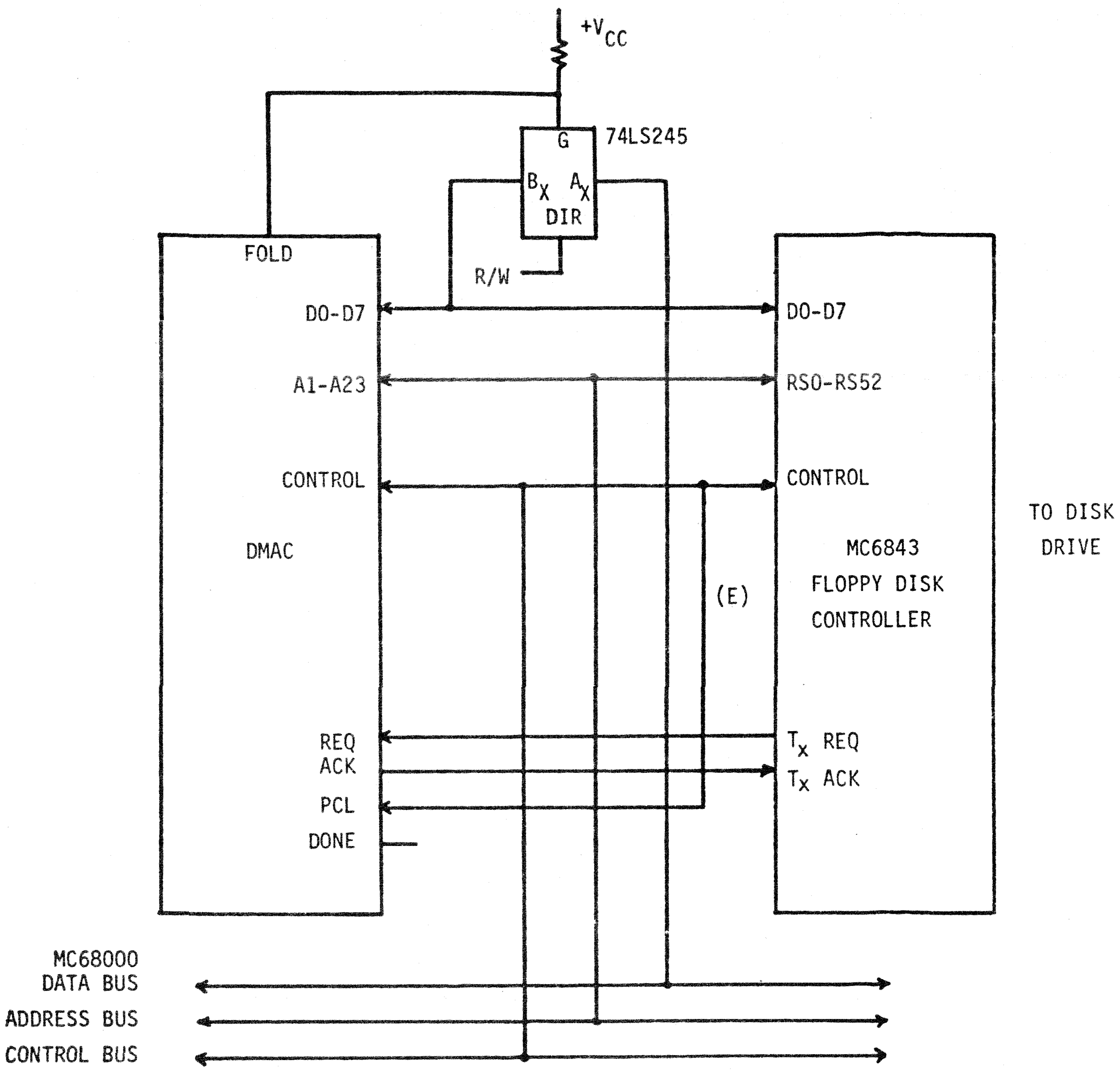
ARRAY CHAINING

DMAC OPERATING MODE

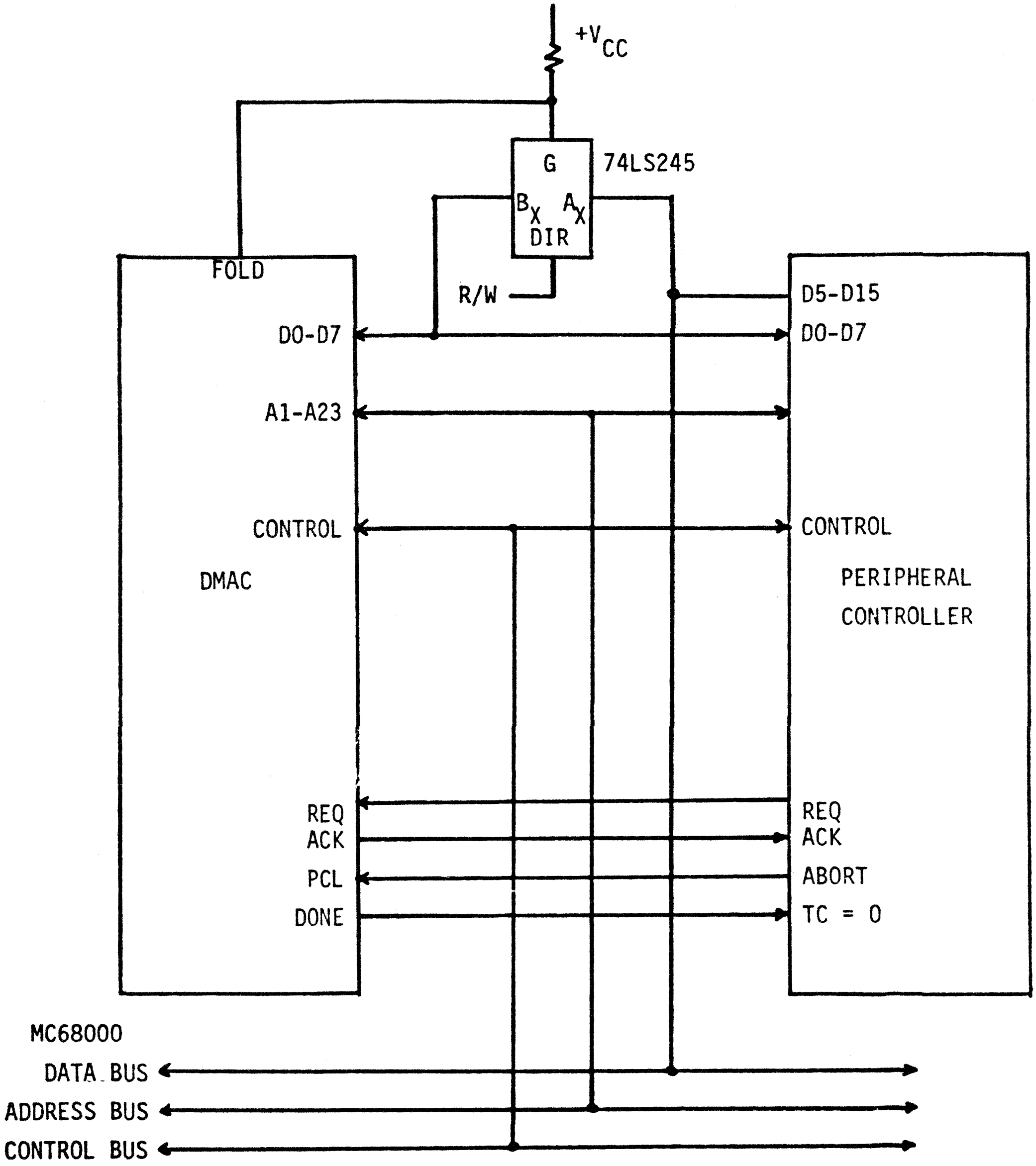


LINKED ARRAY CHAINING

DMAC OPERATING MODE



DMAC CONNECTED TO EXISTING M6800
FAMILY PERIPHERAL PART



DMAC CONNECTED TO FUTURE MC68000
PERIPHERAL PART

THE DMAC IS PERFORMANCE ORIENTED

- FOUR INDEPENDENT DMA CHANNELS.
- BANDWIDTH EQUAL OR GREATER THAN THE MC68000 PROCESSOR.
- DUAL INTERRUPT VECTORS ELIMINATES POLLING.
- DATA CHAINING LOWERS OS OVERHEAD FOR NON-CONTIGUOUS BLOCK TRANSFERS.

THE DMAC IS FAMILY COMPATIBLE

- COMPATIBLE WITH THE M6800 FAMILY OF PERIPHERAL PARTS.
- PLANNED WITH A 32-BIT STRUCTURE FOR FUTURE EXPANSION.
- UTILIZES FEATURES OFFERED BY THE MEMORY-MANAGEMENT-UNIT.
- BUS CYCLES CAN BE SUSPENDED AND RE-INITIATED FOR USE WITH THE BUS ARBITRATION MODULE.

THE DMAC IS FLEXIBLE

- SUPPORTS THREE CHAINING MODES FOR SUPPORTING OPTIMUM DATA STRUCTURES.
- ALLOWS FOR VIRTUAL MEMORY IMPLEMENTATION.
- SUPPORTS BYTE, WORD, AND LONG WORD MOVES.
- PERIPHERAL CONTROL LINES (PCL) REDUCE HARDWARE OVERHEAD.
- SUPPORTS 8-BIT M6800 FAMILY OR 16-BIT MC68000 PERIPHERALS.

THE DMAC LOWERS OPERATING SYSTEM OVERHEAD BY SUPPORTING:

- MEMORY-TO-MEMORY BLOCK TRANSFERS (MMBT).
- EVEN TO ODD AND ODD TO EVEN (MMBTS)
- MAPPED AND UNMAPPED OPERATION WHEN USED WITH THE MMU FOR MULTITASKING SYSTEMS.
- SEPERATE INTERRUPT VECTORS FOR NORMAL AND EXCEPTION PROCESSING TO ELIMINATE POLLING.
- SEGMENT PROTECTION UTILIZING MMU TO ELIMINATE PRIVILEGE CHECKING IN SOFTWARE.
- REGISTER MAP WHICH ALLOWS CHANNELS TO BE PROTECTED FROM EACH OTHER, AND FOR FUNCTION CODES AND INTERRUPT VECTORS TO BE TREATED AS A SEPERATE MEMORY SEGMENT.

THE MC68XXX

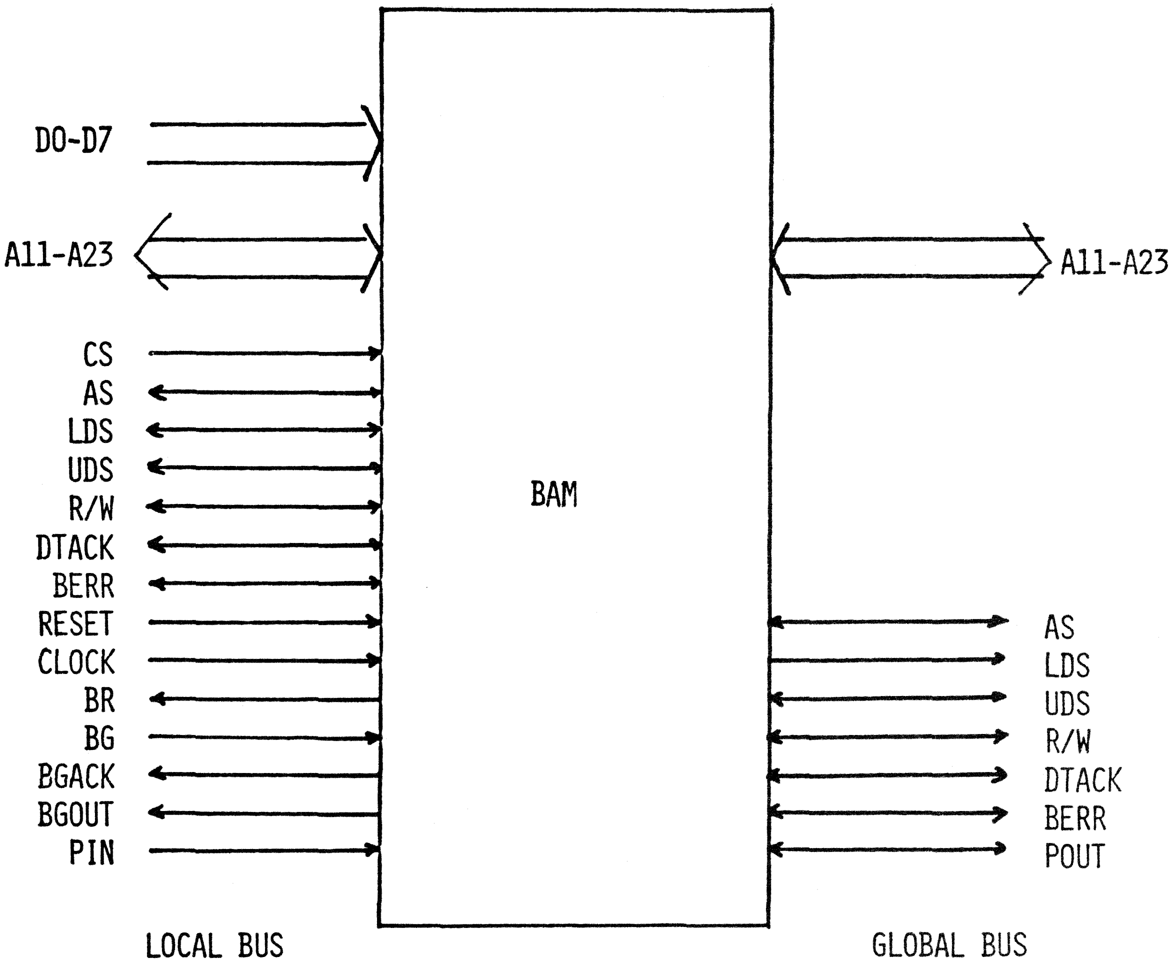
BUS ARBITRATION MODULE

WHY BUS ARBITRATION?

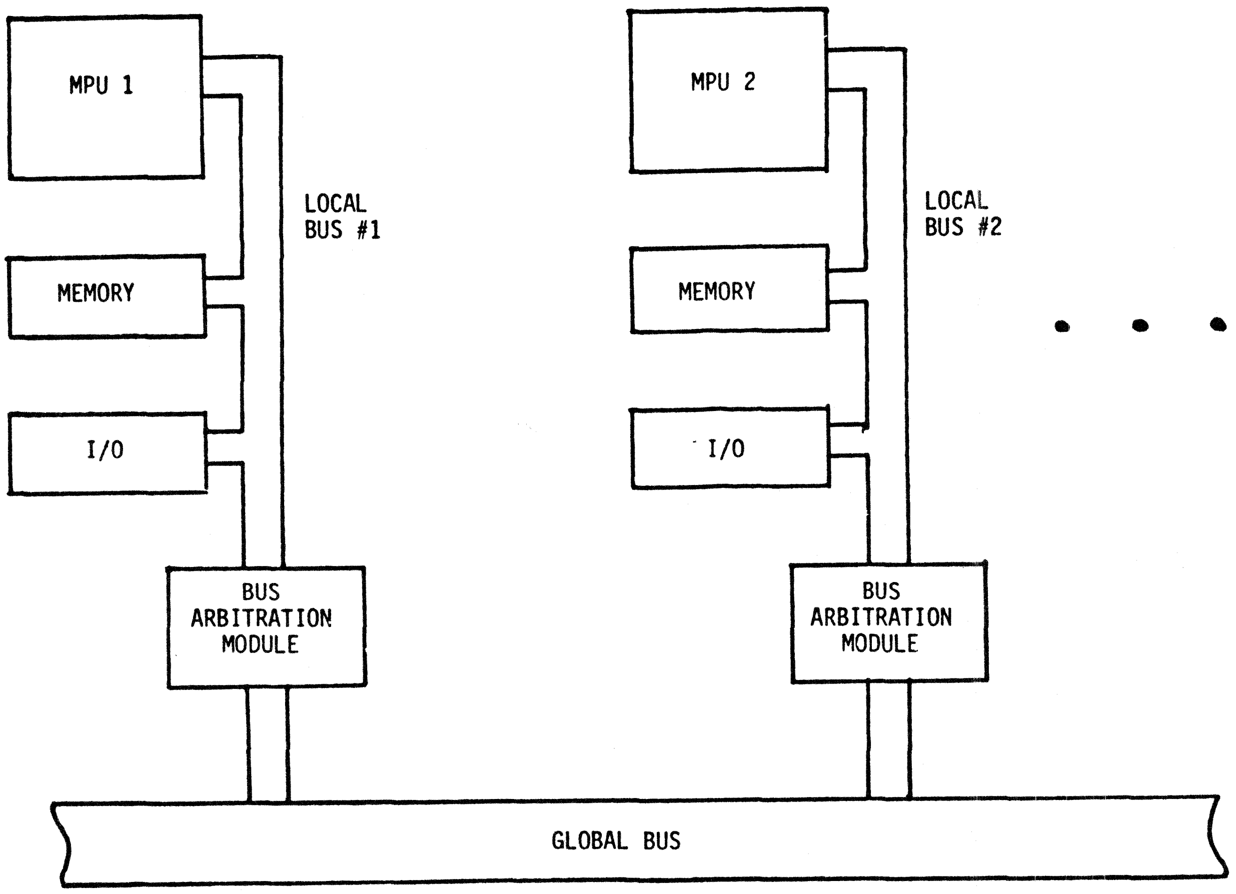
- ALLOW FOR INTERPROCESSOR COMMUNICATION.
- PROVIDE PRIORITIZATION FOR BUS USE.
- PREVENT "DEAD LOCKS".
- PROVIDE SYNCHRONIZATION BETWEEN TWO BUSSES WHICH MIGHT BE ASYNCHRONOUS TO EACH OTHER.
- ALLOW FOR DISTRIBUTED PROCESSING SYSTEMS

BUS ARBITRATION MODULE FEATURES

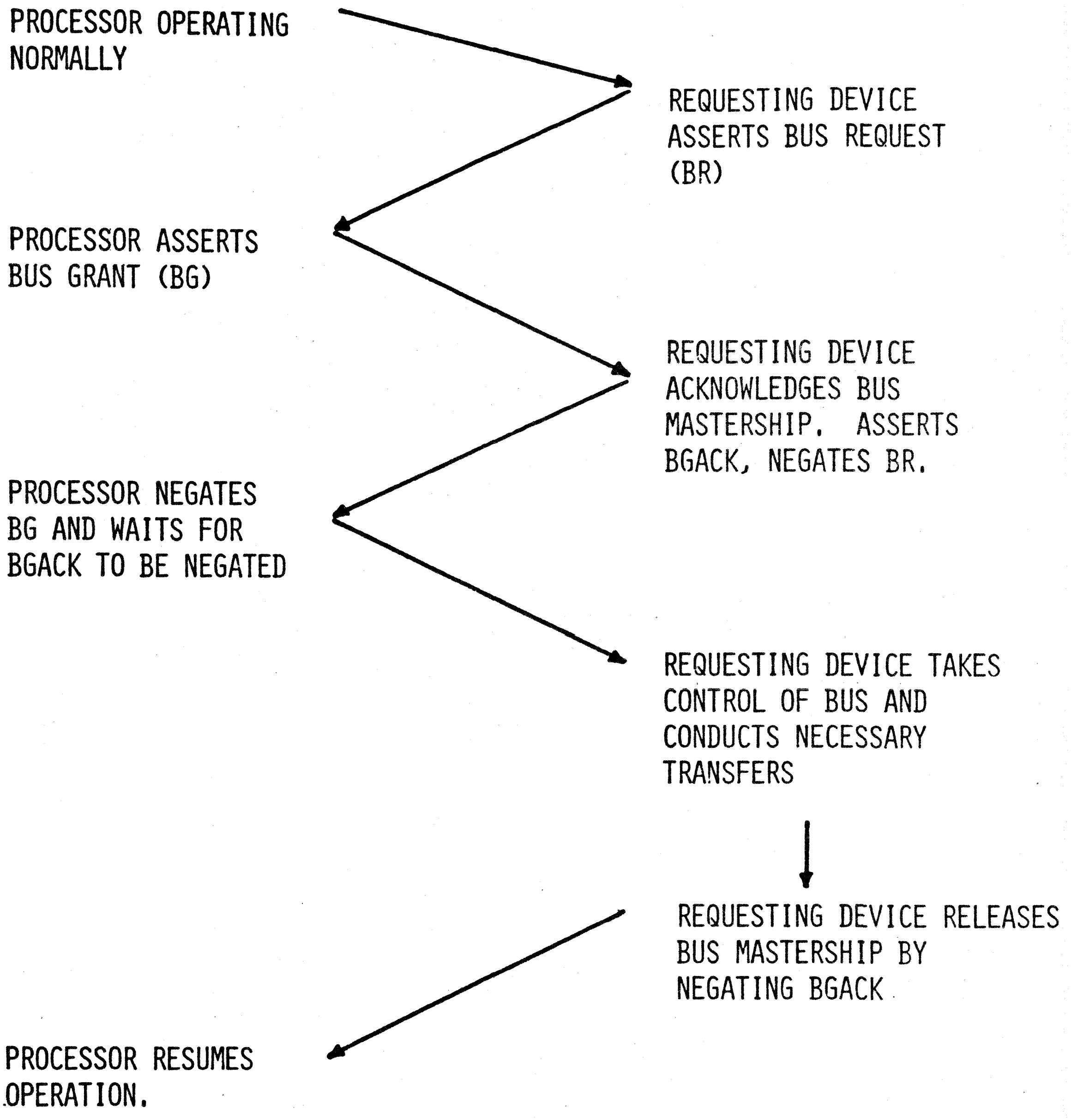
- ALLOWS FOR MULTI-PROCESSOR CONFIGURATIONS.
- PROGRAMMABLE FEATURE MAPS INTERPROCESSOR ADDRESSES.
- ESTABLISHES A GLOBAL BUS AND ALLOWS FOR GLOBAL RESOURCES.
- PROVIDES ADDRESS TRANSLATION FOR COMMUNICATION WITH GLOBAL BUS.
- DAISY CHAINABLE.
- MC68000 BUS COMPATIBLE.
- COMPATIBLE WITH DMAC AND MMU.
- SUPPORTS MC68000 BERR PHILOSOPHY.
- 1K BLOCK ACCESS SIZE.



PIN-OUT OF BUS ARBITRATION MODULE



LOCAL/GLOBAL BUS SCHEME
ESTABLISHED USING BAMS



PROCESSOR OPERATING NORMALLY

REQUESTING DEVICE ASSERTS BUS REQUEST (BR)

PROCESSOR ASSERTS BUS GRANT (BG)

REQUESTING DEVICE ACKNOWLEDGES BUS MASTERSHIP. ASSERTS BGACK, NEGATES BR.

PROCESSOR NEGATES BG AND WAITS FOR BGACK TO BE NEGATED

REQUESTING DEVICE TAKES CONTROL OF BUS AND CONDUCTS NECESSARY TRANSFERS

REQUESTING DEVICE RELEASES BUS MASTERSHIP BY NEGATING BGACK

PROCESSOR RESUMES OPERATION.

BUS ARBITRATION SEQUENCE

MOTOROLA

M68KEMLC(D1)

M68000 SIMULATOR REFERENCE MANUAL

Available now

MICROSYSTEMS

MOTOROLA

M68000XASM(D)

M68000
CROSS MACRO ASSEMBLER
PRELIMINARY SPECIFICATIONS

Available now

MICROSYSTEMS

MOTOROLA Semiconductors

MC6801

Advance Information

MOS

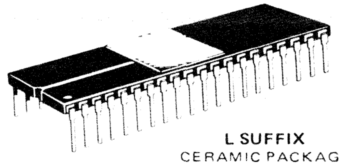
(N-CHANNEL, SILICON-GATE DEPLETION LOAD)

MICROCOMPUTER

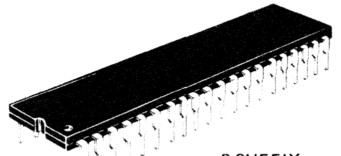
MICROCOMPUTER UNIT (MCU)

The MC6801 MCU is an 8-bit microcomputer system which is compatible with the M6800 family of parts. The MC6801 MCU is object code compatible with the MC6800 with improved execution times of key instructions plus several new 16-bit and 8-bit instructions including an 8 X 8 unsigned multiply with 16-bit result. The MC6801 MCU can operate as a single chip microcomputer or be expanded to 65K words. The MC6801 MCU is TTL compatible and requires one +5.0 volt power supply. The MC6801 MCU has 2K bytes of ROM and 128 bytes of RAM on chip, Serial Communications Interface (S.C.I.), and parallel I/O as well as a three function 16-bit timer. Block diagram is shown in Figure 1. Features of the MC6801 include the following:

- Expanded M6800 Instruction Set
- 8 X 8 Multiply
- On-Chip Serial Communications Interface (S.C.I.)
- Object Code Compatible With The MC6800 MPU
- 16-Bit Timer
- Single Chip Or Expandable To 65K Words
- 2K Bytes Of ROM
- 128 Bytes Of RAM (64 Bytes Retainable On Power Down)
- 31 Parallel I/O Lines
- Internal Clock/Divided-By-Four
- TTL Compatible Inputs And Outputs
- Interrupt Capability
- **External Clock/Divide-By-One Mask Option (MC6801E) And EPROM Versions MC68701 And MC68701E Available Soon.**



L SUFFIX
CERAMIC PACKAGE
CASE 715



P SUFFIX
PLASTIC PACKAGE
CASE 711

FIGURE 1 - SINGLE-CHIP MICROCOMPUTER BLOCK DIAGRAM

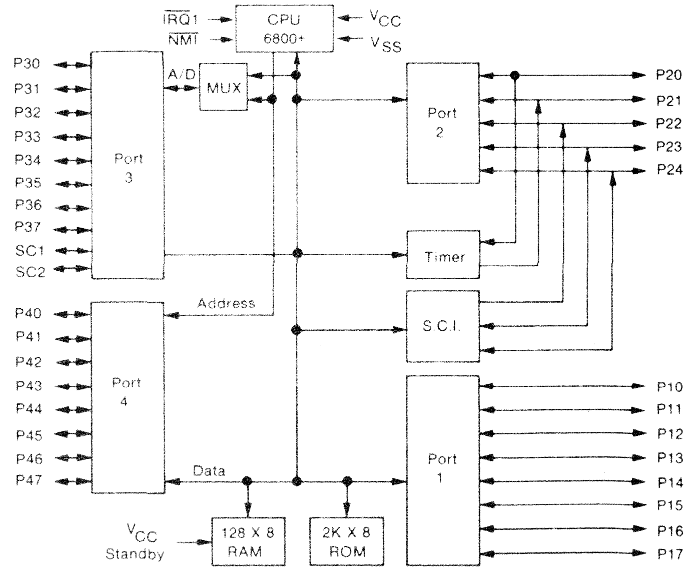
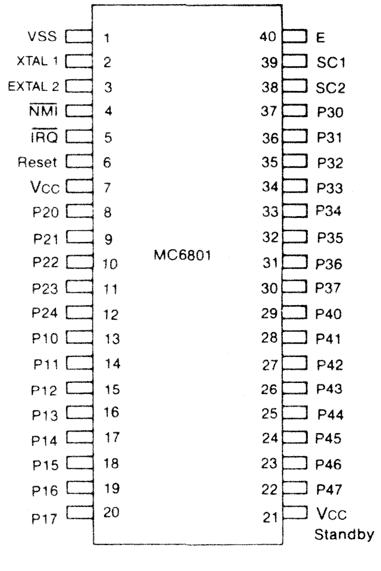


FIGURE 2 — PIN ASSIGNMENT



This is advance information and specifications are subject to change without notice.

ELECTRICAL CHARACTERISTICS (V_{CC} = 5.0V ±5%, V_{SS} = 0, T_A = T_L to T_H unless otherwise noted.)

Characteristic	Symbol	Min	Typ	Max	Unit
Input High Voltage Reset	V_{IH}	V_{SS} + 2.0 V_{SS} + 4.0	-	V_{CC} V_{CC}	Vdc
Input Low Voltage	V_{IL}	V_{SS} - 0.3	-	V_{SS} + 0.8	Vdc
Three-State (Off State) Input Current P10-P17 (Vin = 0.4 to 2.4 Vdc) P20-P24, P30-P37	I_{TSI} I_{TSI}	- -	2.0 2.0	10 10	μAdc μAdc
Output High Voltage All Outputs Except XTAL 1 and EXTAL 2 (I_{Load} = -200 μAdc)	V_{OH}	V_{SS} + 2.4	-	-	Vdc
Output Low Voltage All Outputs Except XTAL 1 and EXTAL 2 (I Load = 1.6 mAdc)	V_{OL}	-	-	V_{SS} +0.4	Vdc
Power Dissipation	P_D	-	-	1200	mW
Capacitance (Vin = 0, T_A = 25°C, f = 1.0 MHz) P10-P17, P20-P24, P40-P47 P30-P37 Reset SC1, SC2, IRQ	C_{in}	- - -	- - -	12.5 10 7.5	pF
Peripheral Data Setup Time (Figure 5)	$^{t}PDSU$	200	-	-	ns
Peripheral Data Hold Time (Figure 5)	^{t}PDH	0	-	-	ns
Delay Time, Enable negative transition to OS3 negative transition	$^{t}OSD1$	-	-	1.0	μs
Delay Time, Enable negative transition to OS3 positive transition	$^{t}OSD2$	-	-	1.0	μs
Delay Time, Enable negative transition to Peripheral Data Valid (Figure 6)	^{t}PWD	-	-	350	ns
Delay Time, Enable negative transition to Peripheral CMOS Data Valid (V_{CC} - 30% V_{CC}, P20-P24 (Figure 6)	$^{t}CMOS$	-	-	2.0	μs
Darlington Drive Current V_O = 1.5 Vdc P10-P17	I_{OH}	-1.0	-2.5	-10	mAdc
Standby Voltage (Not Operating) (Operating)	V_{SBB} V_{SB}	4.00 4.75	- -	5.25 5.25	Vdc

NOTE: The above electricals satisfy Ports 1 and 2 always, and Ports 3 and 4 in the single chip mode only.

BUS TIMING (Figure 9)

Characteristic	Symbol	Min	Typ	Max	Unit
Cycle Time	t_{CYC}	1000	-	-	ns
Address Strobe Pulse Width High	PW_{ASH}	220	-	-	ns
Address Strobe Rise Time	t_{ASR}	-	-	50	ns
Address Strobe Fall Time	t_{ASF}	-	-	50	ns
Address Strobe Delay Time	t_{ASD}	60	-	-	ns
Enable Rise Time	t_{ER}	-	-	50	ns
Enable Fall Time	t_{EF}	-	-	50	ns
Enable Pulse Width High Time	PW_{EH}	450	-	-	ns
Enable Pulse Width Low Time	PW_{EL}	450	-	-	ns
Address Strobe to Enable Delay Time	t_{ASED}	60	-	-	ns
Address Delay Time	t_{AD}	-	-	270	ns
Data Delay Write Time	t_{DDW}	-	-	225	ns
Data Set-up Time	t_{DSR}	100	-	-	ns
Hold Time } Read	t_{HR}	20	-	100	ns
Write	t_{HW}	20	-	-	ns
Address Delay Time for Latch	t_{ADL}	-	-	200	ns
Address Hold Time for Latch	t_{AHL}	20	-	-	ns
Pulse Width	PW_O	370	370	-	ns
Address Hold Time	t_{AH}	20	-	-	ns
Total Up Time	t_{UT}	750	-	-	ns

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	VCC	-0.3 to +7.0	Vdc
Input Voltage	Vin	-0.3 to +7.0	Vdc
Operating Temperature Range	TA	0 to 70	°C
Storage Temperature Range	Tstg	-55 to +150	°C
Thermal Resistance Plastic Package Ceramic Package	θ_{JA}	100 50	°C/W

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum rated voltages to this high impedance circuit. For proper operation it is recommended that V_{in} and V_{out} be constrained to the range $V_{SS} \leqq (V_{in}$ or $V_{out}) \leqq V_{DD}$.

TABLE 1 — MODE AND PORT SUMMARY

MCU SIGNAL DESCRIPTION

This section gives a description of the MCU signals for the various modes. Figure 2 shows the general pin assignments for the signals. SC1 and SC2 are signals which vary with the mode that the chip is in. Table 1 gives a summary of their function.

MODE	PORT 1 Eight Lines	PORT 2 Five Lines	PORT 3 Eight Lines	PORT 4 Eight Lines	SC1	SC2
SINGLE CHIP	I/O	I/O	I/O	I/O	$\overline{\text{IS3}}$(I)	$\overline{\text{OS3}}$(O)
EXPANDED MUX	I/O	I/O	ADDRESS BUS (A0-A7) DATA BUS (D0-D7)	ADDRESS BUS* (A8-A15)	AS(O)	R/$\overline{\text{W}}$(O)
EXPANDED NON-MUX	I/O	I/O	DATA BUS (D0-D7)	ADDRESS BUS* (A0-A7)	$\overline{\text{IOS}}$(O)	R/$\overline{\text{W}}$(O)

*These lines can be substituted for I/O (Input Only) starting with the most significant address line.

I = Input	IS = Input Strobe	SC = Strobe Control
O = Output	OS = Output Strobe	AS = Address Strobe
R/$\overline{\text{W}}$ = Read/$\overline{\text{Write}}$	IOS = I/O Select	

READ/WRITE TIMING FOR PORTS 3 AND 4 (Figures 3-4)

Characteristic	Symbol	Min	Typ	Max	Unit
Address Delay	t_{AD}	-	-	270	ns
Peripheral Read Access Time $t_{acc} = t_{ut} - (t_{AD} + t_{DSR})$	t_{acc}	-	-	530	ns
Data Setup Time (Read)	t_{DSR}	100	-	-	ns
Input Data Hold Time	t_{HR}	10	-	-	ns
Output Data Hold Time	t_{HW}	20	-	-	ns
Address Hold Time (Address, R/W)	t_{AH}	20	-	-	ns
Data Delay Time (Write)	t_{DDW}	-	165	225	ns
Processor Controls					
Processor Control Setup Time	t_{PCS}	200	-	-	ns
Processor Control Rise and Fall Time	t_{PCr}, t_{PCf}	-	-	100	ns
(Measured between 0.8V and 2.0V)				100	

PORT 3 STROBE TIMING (Figures 7-8)

Characteristic	Symbol	Min	Typ	Max	Unit
Output Strobe Delay 1	t_{DSD1}	-	-	1.0	μs
Output Strobe Delay 2	T_{OSD2}	-	-	1.0	μs
Input Strobe Pulse Width	PW_{IS}	200	-	-	ns
Input Data Hold Time	t_{IH}	20	-	-	ns
Input Data Setup Time	t_{IS}	100	-	-	ns

FIGURE 3 — READ DATA FROM MEMORY OR PERIPHERALS EXPANDED NON-MULTIPLEXED

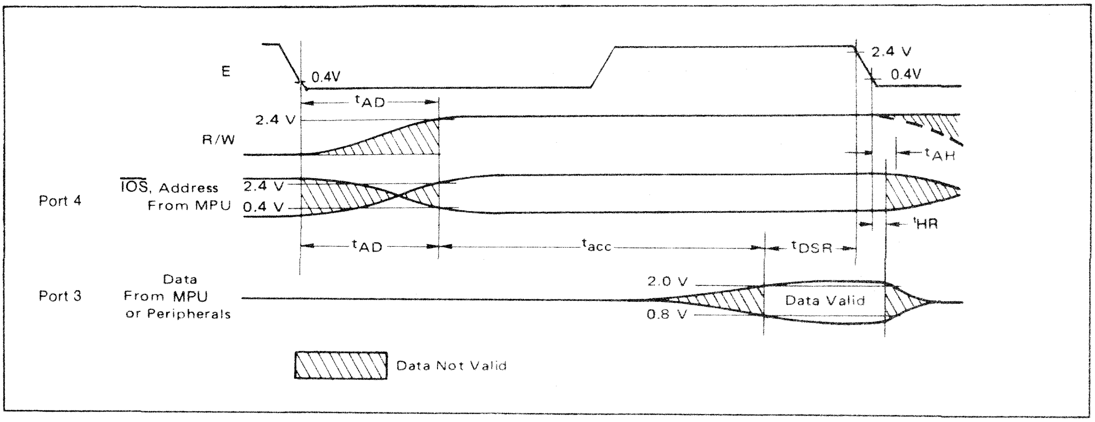
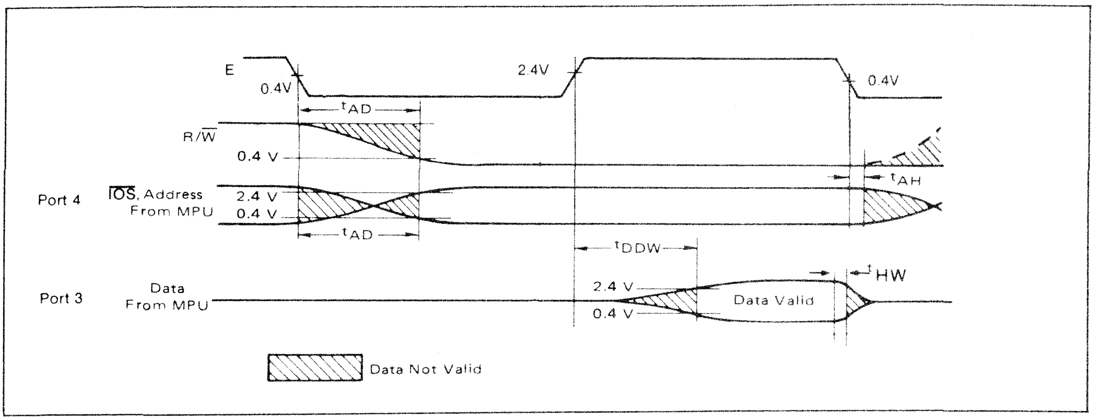


FIGURE 4 — WRITE DATA IN MEMORY OR PERIPHERALS EXPANDED NON-MULTIPLEXED



PORTS 1 AND 2 , AND PORTS 3 AND 4 IN THE SINGLE CHIP MODE

FIGURE 5 — PERIPHERAL DATA SETUP AND HOLD TIMES
(Read Mode)

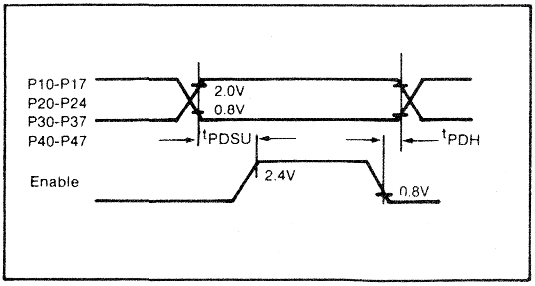
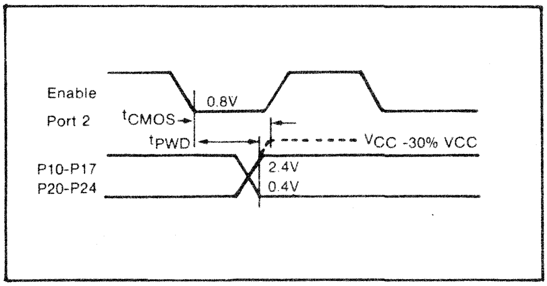


FIGURE 6 — PERIPHERAL CMOS DATA DELAY TIMES
(Write Mode)



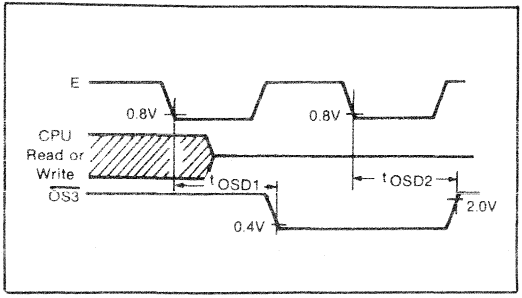


FIGURE 7 — OUTPUT STROBE TIMING — SINGLE CHIP MODE

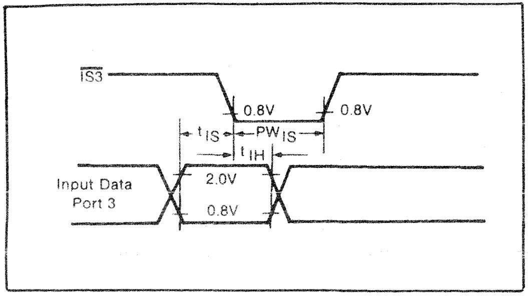


FIGURE 8 — INPUT STROBE TIMING — SINGLE CHIP MODE

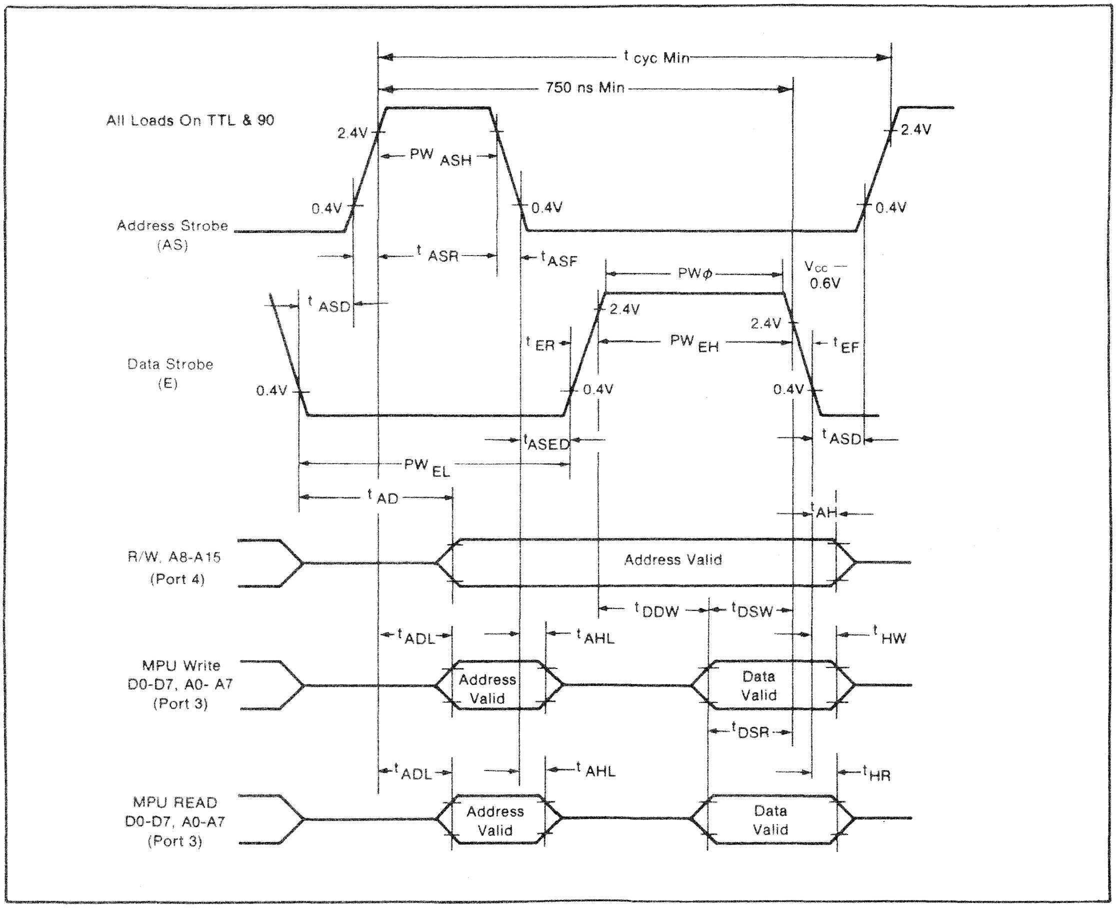


FIGURE 9 — MULTIPLEXED BUS TIMING

FIGURE 10—CMOS LOAD

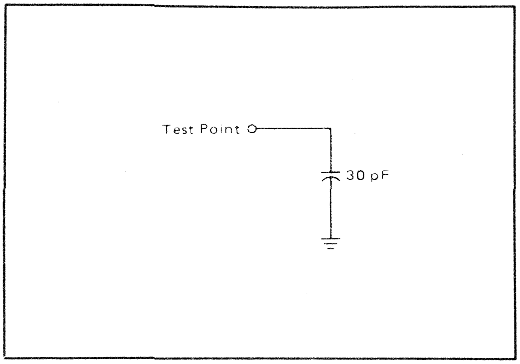
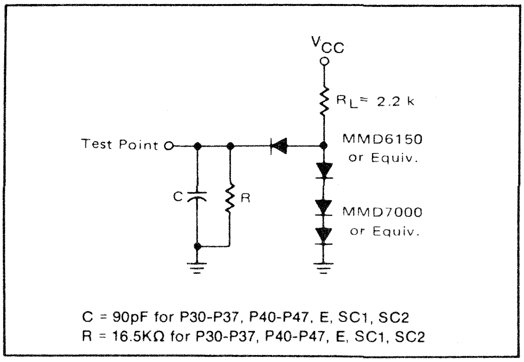
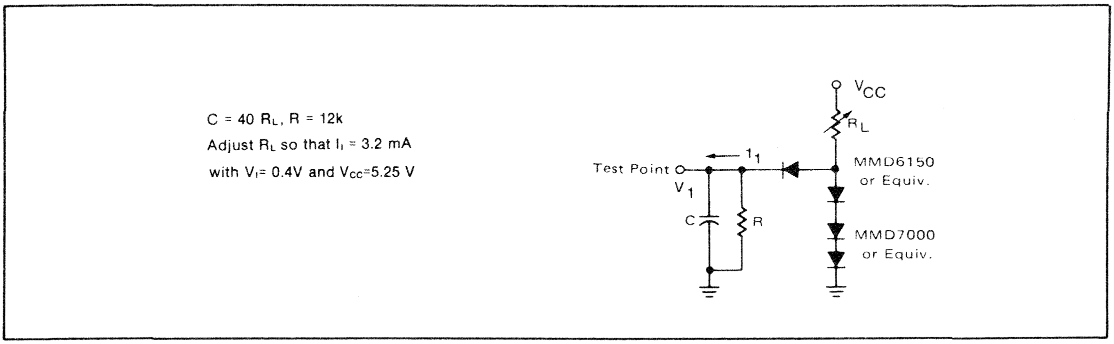


FIGURE 11 — BUS TIMING TEST LOAD AND PORTS 1, 3 AND 4 FOR SINGLE CHIP MODE



C = 90pF for P30-P37, P40-P47, E, SC1, SC2
R = 16.5KΩ for P30-P37, P40-P47, E, SC1, SC2

FIGURE 12 — TEST LOADS FOR PORT 1
Darlington Load
(P10-P17)



C = 40 R_L, R = 12k
Adjust R_L so that I_I = 3.2 mA
with V_I= 0.4V and V_{CC}=5.25 V

FIGURE 13 — TYPICAL DATA BUS OUTPUT DELAY versus CAPACITIVE LOADING

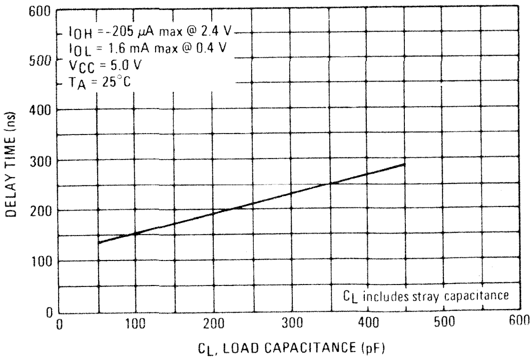
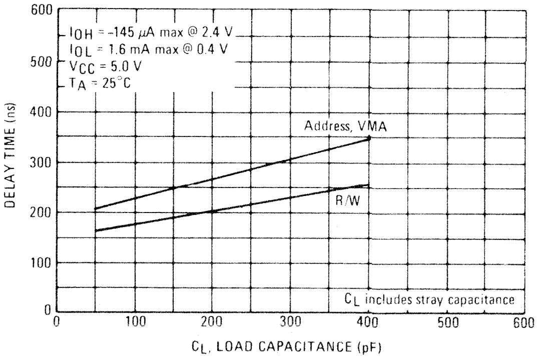


FIGURE 14 — TYPICAL READ/WRITE, VMA AND ADDRESS OUTPUT DELAY versus CAPACITIVE LOADING



SIGNAL DESCRIPTIONS

Vcc and Vss

These two pins are used to supply power and ground to the chip. The voltage supplied will be +5 volts ±5%.

XTAL 1 and EXTAL 2

These connections are for a parallel resonant fundamental crystal, AT cut. Divide by 4 circuitry is included with the internal clock, so a 4 MHz crystal may be used to run the system at 1 MHz. The divide by 4 circuitry allows for use of the inexpensive 3.56 MHz Color TV crystal for non-time critical applications. Two 27 pF capacitors are needed from the two crystal pins to ground to insure reliable operation. EXTAL2 may be driven by an external clock source at a 4 MHz rate to run at 1 MHz with a 40/60% duty cycle. It is not restricted to 4 MHz, as it will divide by 4 any frequency less than or equal to 4 MHz. XTAL1 must be grounded if an external clock is used. The following are the recommended crystal parameters:

AT = Cut Parallel Resonance Crystal
C_o = 7 pF MAX
FREQ = 4.0 MHz @ C_L = 24 pF
R_S = 50 ohms MAX.
Frequency Tolerance - ±5% to ±0.02%
The best E output "Worst Case Design" tolerance is ±0.05% (500 ppM) using A ±0.02% crystal.

Vcc Standby

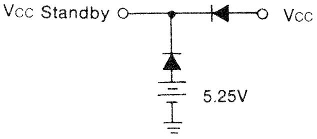
This pin will supply +5 volts ±5% to the standby RAM on the chip. The first 64 bytes of RAM will be maintained in the power down mode with 8 mA current max in the ROM version. The circuit of figure 15 can be utilized to assure that Vcc Standby does not go below VSBB during power down.

To retain information in the RAM during power down the following procedure is necessary:

1) Write "0" into the RAM enable bit, RAM E. RAM E is bit 6 of the RAM Control Register at location $0014. This disables the standby RAM, thereby protecting it at power down.

2) Keep Vcc Standby greater than VSBB.

FIGURE 15—BATTERY BACKUP FOR Vcc STANDBY



Reset

This input is used to reset and start the MPU from a power down condition, resulting from a power failure or an initial start-up of the processor. On power up, the reset must be held low for at least 20 ms. During operation, $\overline{\text{Reset}}$, when brought low, must be held low at least 3 clock cycles.

When a high level is detected, the MPU does the following:

a) All the higher order address lines will be forced high.

b) I/O Port 2 bits, 2, 1, and 0 are latched into programmed control bits PC2, PC1 and PC0.

c) The last two (FFFE, FFFF) locations in memory will be used to load the program addressed by the program counter.

d) The interrupt mask bit is set, must be cleared before the MPU can recognize maskable interrupts.

Enable (E)

This supplies the external clock for the rest of the system when the internal oscillator is used. It is a single phase, TTL compatible clock, and will be the divide by 4 result of the crystal frequency. It will drive one TTL load and 90 pF.

Non-Maskable Interrupt ($\overline{\text{NMI}}$)

A low-going edge on this input requests that a non-maskable-interrupt sequence be generated within the processor. As with the Interrupt Request signal, the processor will complete the current instruction that is being executed before it recognizes the $\overline{\text{NMI}}$ signal. The interrupt mask bit in the Condition Code Register has no effect on $\overline{\text{NMI}}$.

In response to an $\overline{\text{NMI}}$ interrupt, the Index Register, Program Counter, Accumulators, and Condition Code Register are stored on the stack. At the end of the sequence, a 16-bit address will be loaded that points to a vectoring address located in memory locations FFFC and FFFD. An address loaded at these locations causes the MPU to branch to a non-maskable interrupt service routine in memory.

A 3.3 kΩ external resistor to Vcc should be used for wire-OR and optimum control of interrupts.

Inputs $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$ are hardware interrupt lines that are sampled during E and will start the interrupt routine on the clock bar following the completion of an instruction.

Interrupt Request ($\overline{\text{IRQ}}$)

This level sensitive input requests that an interrupt sequence be generated within the machine. The processor will wait until it completes the current instruction that is being executed before it recognizes the request. At that time, if the interrupt mask bit in the Condition Code Register is not set, the machine will begin an interrupt sequence. The Index Register, Program Counter, Accumulators, and Condition Code Register are stored on the stack. Next the MPU will respond to the interrupt request by setting the interrupt mask bit high so that no further maskable interrupts may occur. At the end of the cycle, a 16-bit address will be loaded that points to a vectoring address which is located in memory locations FFF8 and FFF9. An address loaded at these locations causes the MPU to branch to an interrupt routine in memory.

The IRQ requires a 3.3 kΩ external resistor to Vcc which should be used for wire-OR and optimum control of interrupts. Internal Interrupts will use an internal interrupt line (IRQ2). This interrupt will operate the same as IRQ except that it will use the vector address of FFF0 and FFF7. IRQ1 will have priority over IRQ2 if both occur at the same time. The Interrupt Mask Bit in the condition mode register masks both interrupts. (See Figure 25).

The following pins are available in the Single Chip Mode, and are associated with Port 3 only.

Input Strobe ($\overline{\text{IS3}}$) (SC1)

This sets an interrupt for the processor when the IS3 Enable bit is set. As shown in Figure 8 Input Strobe Timing, IS3 will fall T_{IS} minimum after data is valid on Port 3. If IS3 Enable is set in the I/O Port Control/Status Register, an interrupt will occur. If the latch enable bit in the I/O Control Status Register is set, this strobe will latch the input data from another device when that device has indicated that it has valid data.

Output Strobe ($\overline{\text{OS3}}$) (SC2)

This signal is used by the processor to strobe an external device, indicating valid data is on the I/O pins. The timing for the Output Strobe is shown in Figure 7. I/O Port Control/Status Register is discussed in the following section.

The following pins are available in the Expanded Modes.

Read/Write (R/W) (SC2)

This TTL compatible output signals the peripherals and memory devices whether the MPU is in a Read (high) or a Write (low) state. The normal standby state of this signal is Read (high). This output is capable of driving one TTL load and 90 pF.

I/O Strobe ($\overline{IOS}$) (SC1)

In the expanded non-multiplexed mode of operation, $\overline{IOS}$ internally decodes A9 through A15 as zero's and A8 as a one. This allows external access of the 256 locations from $0100 to $01FF. The timing diagrams are shown as figures 3 and 4.

Address Strobe (AS) (SC1)

In the expanded multiplexed mode of operation address strobe is output on this pin. This signal is used to latch the 8 LSB's of address which are multiplexed with data on Port 3. An 8-bit latch is utilized in conjunction with Address Strobe, as shown in figure 29, Expanded Multiplexed Mode. Address Strobe signals the latch when it is time to latch the address lines so the lines can become data bus lines during the E pulse. The timing for this signal is shown in the MC6801 Bus Timing Figure 9. This signal is also used to disable the address from the multiplexed bus allowing a deselect time, T_{ASD} before the data is enabled to the bus.

MC6801 PORTS

There are four I/O ports on the MC6801 MCU; three 8-bit ports and one 5-bit port. There are two control lines associated with one of the 8-bit ports. Each port has an associated write only Data Direction Register which allows each I/O line to be programmed to act as an input or an output.* A "1" in the corresponding Data Direction Register bit will cause that I/O line to be an output. A "0" in the corresponding Data Direction Register bit will cause that I/O line to be an input. There are four ports: Port 1, Port 2, Port 3, and Port 4. Their addresses and the addresses of their Data Direction registers are given in Table 2.

*The only exception is bit 1 of Port 2, which can either be data input or Timer output.

TABLE 2 — PORT AND DATA DIRECTION REGISTER ADDRESSES

Ports	Port Address	Data Direction Register Address
I/O Port 1	$0002	$0000
I/O Port 2	$0003	$0001
I/O Port 3	$0006	$0004
I/O Port 4	$0007	$0005

I/O Port 1

This is an 8-bit port whose individual bits may be defined as inputs or outputs by the corresponding bit in its data direction register. The 8 output buffers have three-state capability, allowing them to enter a high impedance state when the peripheral data lines are used as inputs. In order to be read properly, the voltage on the input lines must be greater than 2.0 volts for a logic "1" and less than 0.8 volt for a logic "0". As outputs, these lines are TTL compatible and may also be used as a source of up to 1 mA at 1.5 volts to directly drive a Darlington base. After Reset, the I/O lines are configured as inputs. In all three modes, Port 1 is always parallel I/O.

I/O Port 2

This port has five lines that may be defined as inputs or outputs by its data direction register. The 5 output buffers have three-state capability, allowing them to enter a high impedance state when used as an input. In order to be read properly, the voltage on the input lines must be greater than 2.0 volts for a logic "1" and less than 0.8 volt for a logic "0". As outputs, this port has no internal pullup resistors but will drive TTL inputs directly. For driving CMOS inputs, external pullup resistors are required. After Reset, the I/O lines are configured as inputs. Three pins on Port 2 (pins 10, 9 and 8 of the chip) are used to program the mode of operation during reset. The values of these pins at reset are latched into the three MSB's (bits 7, 6, and 5) of Port 2 which are read only. This is explained in the Mode Selection Section.

In all three modes, Port 2 can be configured as I/O and provides access to the Serial Communications Interface and the Timer. Bit 1 is the only pin restricted to data input or Timer output.

I/O Port 3

This is an 8-bit port that can be configured as I/O, a data bus, or an address bus multiplexed with the data bus — depending on the mode of operation hardware programmed by the user at reset. As a data bus, Port 3 is bi-directional. As an input for peripherals, it must be supplied regular TTL levels, that is, greater than 2.0 volts for a logic "1" and less than 0.8 volt for a logic "0".

Its TTL compatible three-state output buffers are capable of driving one TTL load and 90 pf. In the Expanded Modes, after reset, the data direction register is inhibited and data flow depends on the state of the R/W line. The input strobe (IS3) and the output strobe (OS3) used for handshaking are explained later.

In the three modes Port 3 assumes the following characteristics:

Single Chip Mode: Parallel Inputs/Outputs as programmed by its associated Data Direction Register. There are two control lines associated with this port in this mode, an input strobe and an output strobe, that can be used for handshaking. They are controlled by the I/O Port Control/Status Register explained at the end of this section.

Expanded Non-Multiplexed Mode: In this mode Port 3 becomes the data bus (D7-D0).

Expanded Multiplexed Mode: In this mode Port 3 becomes both the data bus (D7-D0) and lower bits of the address bus (A7-A0). An address strobe output is true when the address is on the port.

I/O PORT 3 CONTROL/STATUS REGISTER

	7	6	5	4	3	2	1	0
$000F	IS3 FLAG	IS3 ENABLE	X	OSS	LATCH ENABLE	X	X	X

Bit 0 Not used.
Bit 1 Not used.
Bit 2 Not used.
Bit 3 **Latch Enable.** This controls the input latch for I/O Port 3. If this bit is set high the input data will be latched with the falling edge of the Input Strobe, IS3. This bit is cleared by reset, or CPU Read Port 3.

Bit 4 **(OSS) Output Strobe Select.** This bit will select if the Output Strobe should be generated by a write to I/O Port 3 or a read of I/O Port 3. When this bit is cleared the strobe is generated by a read Port 3. When this bit is set the strobe is generated by a write Port 3.

Bit 5 Not used.

Bit 6 **IS3 ENABLE.** This bit will be the interrupt caused by IS3. When set to a low level the IS3 FLAG will be set by input strobe but the interrupt will not be generated. This bit is cleared by reset.

Bit 7 **IS3 FLAG.** This is a read only status bit that is set by the falling edge of the input strobe, IS3. It is cleared by a read of the Control/Status Register followed by a read or write of I/O Port 3. Reset will clear this bit.

I/O Port 4

This is an 8-bit port that can be configured as I/O or as address lines depending on the mode of operation. In order to be read properly, the voltage on the input lines must be greater than 2.0 volts for a logic "1" and less than 0.8 volt for a logic "0" As outputs, each line is TTL compatible and can drive 1 TTL load and 90 pF. After reset, the lines are configured as inputs. To use the pins as addresses, therefore, they should be programmed as outputs. In the three modes, Port 4 assumes the following characteristics:

Single Chip Mode: Parallel Inputs/Outputs as programmed by its associated Data Direction Register.

Expanded Non-Multiplexed Mode: In this mode Port 4 is configured as the lower order address lines (A7-A0) by writing one's to the data direction register. When all eight address lines are not needed, the remaining lines, starting with the most significant bit, may be used as I/O (inputs only).

Expanded Multiplexed Mode: In this mode Port 4 is configured as the high order address lines (A15-A8) by writing one's to the data direction register. When all eight address lines are not needed, the remaining lines, starting with the most significant bit, may be used as I/O (inputs only).

MODE SELECTION

The mode of operation that 6801 will operate in after Reset is determined by hardware that the user must wire on pins 10, 9, and 8 of the chip. These pins are the three LSB's (I/O 2, I/O 1, and I/O 0 respectively) of Port 2. They are latched into programmed control bits PC2, PC1, and PC0 when reset goes high. I/O Port 2 Register is shown below.

	7	6	5	4	3	2	1	0
$0003	PC2	PC1	PC0	I/O 4	I/O 3	I/O 2	I/O 1	I/O 0

An example of external hardware that could be used in the Expanded Non-Multiplexed Mode is given in Figure 16. In the Expanded Non-Multiplexed Mode, pins 10, 9 and 8 are programmed Hi, Lo, Hi respectively as shown.

Couplers between the pins on Port 2 and the peripherals attached may be required. If the lines go to devices which require signals at power up differing from the signals needed to program the 6801's mode, couplers are necessary.

The MC14066B can be used to provide this isolation between the peripheral device and the MCU during reset. Figure 17 shows the logic diagram and truth table for the MC14066B. It is bidirectional and requires no external logic to determine the direction of the information flow. The logic shown insures that the data on the peripheral will not change before it is latched into the MCU and the MCU has started the reset sequence.

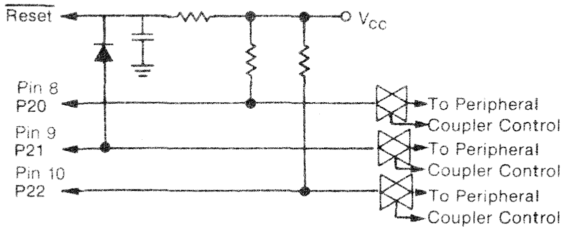
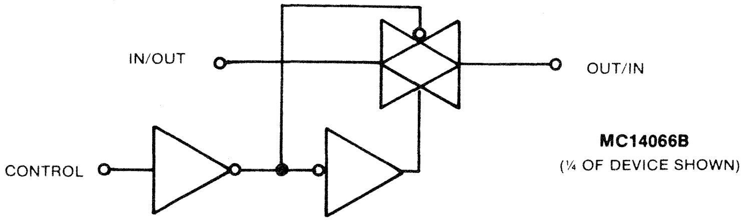


FIGURE 16 — DIODE CONFIGURATION FOR THE EXPANDED NON-MULTIPLEXED MODE

As bits 5, 6 and 7 of Port 2 are read only, the mode cannot be changed through software. The mode selections are shown in Table 3.

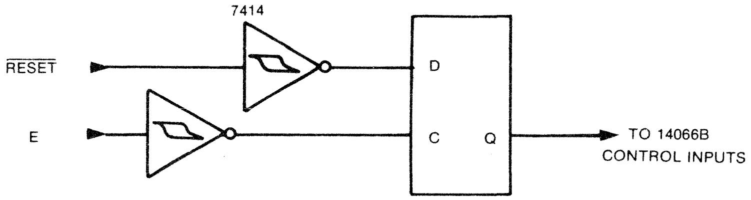
P20 refers to Port 2, bit 0.

FIGURE 17—MC14066B QUAD ANALOG, SWITCH/ MULTIPLEXER IN A TYPICAL MC6801 CIRCUIT



CONTROL	SWITCH
0	OFF
1	ON

V CONTROL	Vin TO Vout RESISTANCE
V_{SS}	> 10^9 OHMS TYP.
V_{DD}	300 OHMS TYP.



MC6801 BASIC MODES

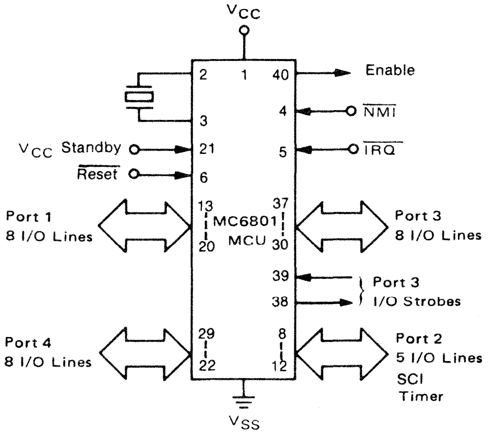
The MC6801 is capable of operating in three basic modes; (1) Single Chip Mode, (2) Expanded Multiplexed Mode (compatible with M6800 peripheral family) (3) Expanded Non-Multiplexed Mode.

SINGLE CHIP MODE

In the Single Chip Mode the Ports are configured for I/O.

This is shown in Figure 18 the single Chip Mode. In this mode, Port 3 will have two associated control lines, an input strobe and an output strobe for handshaking data.

FIGURE 18 — MC6801 MCU SINGLE-CHIP MODE



EXPANDED NON-MULTIPLEXED MODE

In this mode the MC6801 will directly address M6800 peripherals with no external logic. In this mode Port 3 becomes the data bus, Port 4 becomes the A7-A0 address bus or partial address and I/O (inputs only), Port 2 can be parallel I/O, serial I/O, Timer, or any combination thereof. Port 1 is parallel I/O only. In this mode the MC6801 is expandable to 256 locations. The eight address lines associated with Port 4 may be substituted for I/O (inputs only) if a fewer number of address lines will satisfy the application. (See Figure 19).

FIGURE 19 — MC6801 MCU EXPANDED NON-MULTIPLEXED MODE

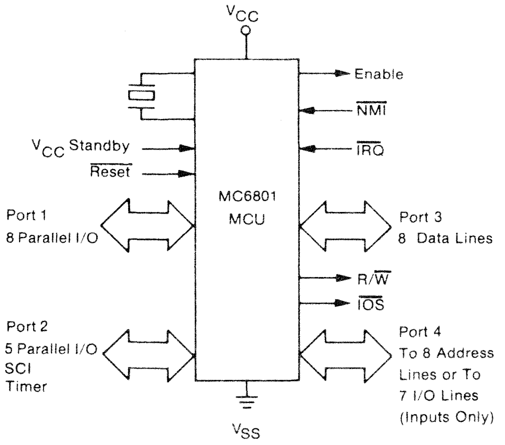
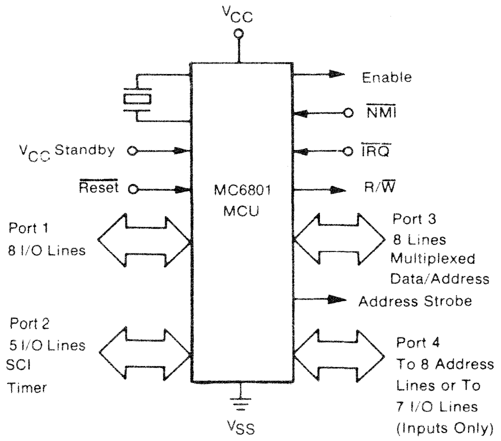


FIGURE 20 — MC6801 MCU EXPANDED MULTIPLEXED MODE



EXPANDED MULTIPLEXED MODE

In this mode Port 4 becomes higher order address lines with an alternative of substituting some of the address lines for I/O (inputs only). Port 3 is the data bus multiplexed with the lower order address lines differentiated by an output called Address Strobe. Port 2 is 5 lines of Parallel I/O, SCI, Timer, or any combination thereof. Port 1 is 8 Parallel I/O lines. In this mode it is expandable to 65K words. (See Figure 20).

TABLE 3 — MODE SELECTS

MODE		PROGRAM CONTROL			ROM	RAM	INTERRUPT VECTORS	BUS
7	SINGLE CHIP	Hi	Hi	Hi	I	I	I	I
6	EXPANDED MULTIPLEXED	Hi	Hi	Lo	I	I	I	Ep/M
5	EXPANDED NON-MULTIPLEXED	Hi	Lo	Hi	I	I	I	Ep
4	SINGLE CHIP TEST	Hi	Lo	Lo	I(2)	I(1)	I	I
3	64K ADDRESS I/O	Lo	Hi	Hi	E	E	E	Ep/M
2	PORTS 3 & 4 EXTERNAL	Lo	Hi	Lo	E	I	E	Ep/M
1		Lo	Lo	Hi	I	I	E	Ep/M
0	TEST-DATA OUTPUTTED FROM ROM & RAM TO I/O PORT 3	Lo	Lo	Lo	I	I	I*	Ep/M

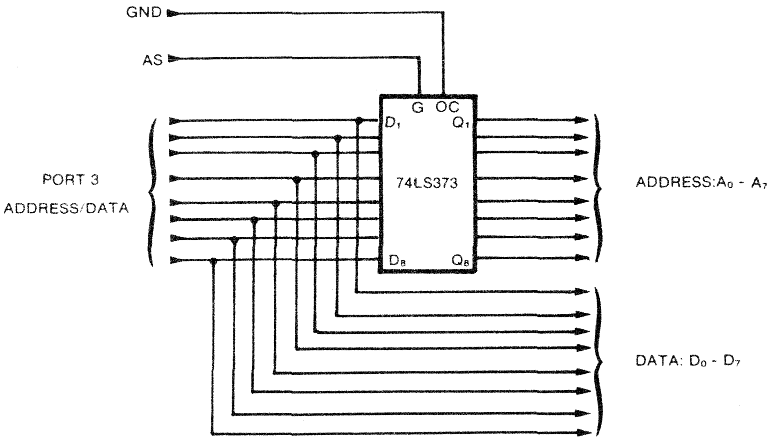
E — EXTERNAL all vectors are external
I — INTERNAL
Ep — EXPANDED
M — MULTIPLEXED

* First two addresses read from external after reset
(1) Address for RAM XX80-XXFF
(2) ROM disabled

Lower order Address Bus Latches

Since the data bus is multiplexed with the lower order address bus in Port 3, latches are required to latch those address bits. The SN74LS373 Transparent octal D-type latch can be used wtih the MC6801 to latch the least significant address byte. Figure 21 shows how to connect the latch to the MC6801. The output control to the LS373 may be connected to ground.

FIGURE 21 — LATCH CONNECTION



FUNCTION TABLE

OUTPUT CONTROL	ENABLE G	ENABLE D	OUTPUT Q
L	H	H	H
L	H	L	L
L	L	X	Q_0
H	X	X	Z

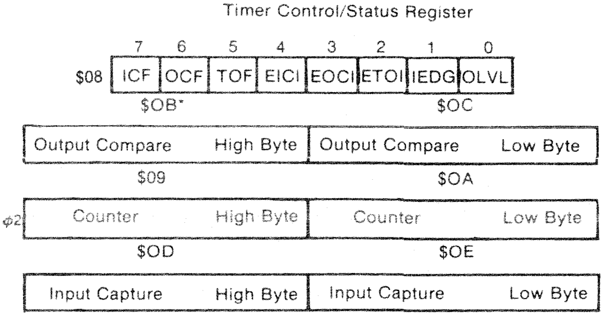
PROGRAMMABLE TIMER

The MC6801 contains an on-chip 16-bit programmable timer which may be used to perform measurements on an input waveform while independently generating an output waveform. Pulse widths for both input and output signals may vary from a few microseconds to many seconds. The timer hardware consists of

- an 8-bit control and status register,
- a 16-bit free running counter,
- a 16-bit output compare register, and
- a 16-bit input capture register

A block diagram of the timer registers is shown in Figure 22.

FIGURE 22 — BLOCK DIAGRAM OF TIMER REGISTERS



* The characters above the registers represent their address in Hex.

Free Running Counter ($0009:000A)

The key element in the programmable timer is a 16-bit free running counter which is driven to increasing values by the MPU ϕ. The counter value may be read by the MPU software at any time. The counter is cleared to zero on RESET and may be considered a read-only register with one exception. Any MPU write to the counter's address ($09) will always result in a preset value of $FFF8 being loaded into the counter regardless of the value involved in the write. This preset feature is intended for testing operation of the part, but may be of value in some applications.

Output Compare Register ($000B:000C)

The Output Compare Register is a 16-bit read/write register which is used to control an output waveform. The contents of this register are constantly compared with the current value of the free running counter. When a match is found, a flag is set (OCF) in the Timer Control and Status Register (TCSR) and the current value of the Output Level bit (OLVL) in the TCSR is clocked to the output level register. Providing the Data Direction Register for Port 2, Bit 1 contains a "1" (output), the output level register value will appear on the pin for Port 2 Bit 1. The values in the Output Compare Register and Output level bit may then be changed to control the output level on the next compare value. The Output Compare Register is set to $FFFF during RESET. The Compare function is inhibited for one cycle following a write to the high byte of the Output Compare Register to insure a valid 16-bit value is in the register before a compare is made.

Input Capture Register ($000D:000E)

The Input Capture Register is a 16-bit read-only register used to store the current value of the free running counter when the proper transition of an external input signal occurs. The input transition change required to trigger the counter transfer is controlled by the Input Edge bit (IEDG) in the TCSR. The Data Direction Register bit for Port 2 Bit 0, should* be clear (zero) in order to gate in the external input signal to the edge detect unit in the timer.

*With Port 2 Bit 0 configured as an output and set to "1", the external input will still be seen by the edge detect unit.

Timer Control and Status Register (TCSR) ($0008)

The Timer Control and Status Register consists of an 8-bit register of which all 8 bits are readable but only the low order 5 bits may be written. The upper three bits contain read-only timer status information and indicate that:

- a proper transition has taken place on the input pin with a subsequent transfer of the current counter value to the input capture register,
- a match has been found between the value in the free running counter and the output compare register, and
- when $0000 is in the free running counter.

Each of the flags may be enabled onto the MC6801 internal bus (IRQ2) with an individual Enable bit in the TCSR. If the I-bit in the MC6801 Condition Code register has been cleared, a prioriy vectored interrupt will occur corresponding to the flag bit(s) set. A description for each bit follows:

	7	6	5	4	3	2	1	0	
Timer Control and Status Register	ICF	OCF	TOF	EICI	EOCI	ETOI	IEDG	OLVL	$0008

Bit 0 **OLVL** Output Level — This value is clocked to the output level register on an output compare. If the DDR for Port 2 bit 1 is set, the value will appear on the output pin.

Bit 1 **IEDG** Input Edge — This bit controls which transition of an input will trigger a transfer of the counter to the input capture register. The DDR for Port 2 Bit 0 must be clear for this function to operate.
IEDG = 0 Transfer takes place on a negative (high-to-low transition).
IEDG = 1 Transfer takes place on a positive edge (low-to-high transition).

Bit 2 **ETOI** Enable Timer Overflow Interrupt — When **set,** this bit enables IRQ2 to occur on the internal bus for a TOF interrupt; when **clear** the interrupt is inhibited.

Bit 3 **EOCI** Enable Output Compare Interrupt — When **set**, this bit enables IRQ2 to appear on the internal bus for an input capture interrupt; when **clear** the interrupt is inhibited.

Bit 4 **EICI** Enable Input Capture Interrupt — When **set**, this bit enables IRQ2 to occur on the internal bus for an input capture interrupt; when **clear** the interrupt is inhibited.

Bit 5 **TOF** Timer Overflow Flag — This read-only bit is **set** when the counter contains $0000. It is **cleared** by a read of the TCSR (with TOF set) followed by an MPU read of the Counter ($09).

Bit 6 **OCF** Output Compare Flag — This read-only bit is **set** when a match is found between the output compare register and the free running counter. It is **cleared** by a read of the TCSR (with OCF set) followed by an MPU write to the output compare register ($0B or $0C).

Bit 7 **ICF** Input Capture Flag — This read-only status bit is **set** by a proper transition on the input to the edge detect unit; it is **cleared** by a read of the TCSR (with ICF set) followed by an MPU read of the Input Capture Register ($0D).

SERIAL COMMUNICATIONS INTERFACE

The MC6801 contains a full-duplex asynchronous serial communications interface (SCI) on board. Two serial data formats (standard mark/space (NRZ) or Bi-phase) are provided at several different data rates. The controller comprises a transmitter and a receiver which operate independently or each other but in the same data format and at the same data rate. Both transmitter and receiver communicate with the MPU via the data bus and with the outside world via pins 2, 3, and 4 of Port 2. The hardware, software, and registers are explained in the following paragraphs.

Wake-Up Feature

In a typical multi-processor application, the software protocol will usually contain a destination address in the initial byte(s) of the message. In order to permit non-selected MPU's to ignore the remainder of the message, a wake-up feature is included whereby all further interrupt processing may be optionally inhibited until the beginning of the next message. When the next message appears, the hardware re-enables (or "wakes-up") the for the next message. The "wake-up" is automatically triggered by a string of ten consecutive 1's which indicates an idle transmit line. The software protocol must provide for the short idle period between any two consecutive messages.

Programmable Options

The following features of the MC6801 serial I/O section are programmable:

- format — standard mark/space (NRZ) or Bi-phase
- clock — external or internal
- baud rate — one of 4 per given MPU $\phi2$ clock frequency or external clock X8 input
- wake-up feature — enabled or disabled
- interrupt requests — enabled or masked individually for transmitter and receiver data registers
- clock output — internal clock enabled or disabled to Port 2 (Bit 2)
- Port 2 (bits 3 and 4) — dedicated or not dedicated to serial I/O individually for transmitter and receiver.

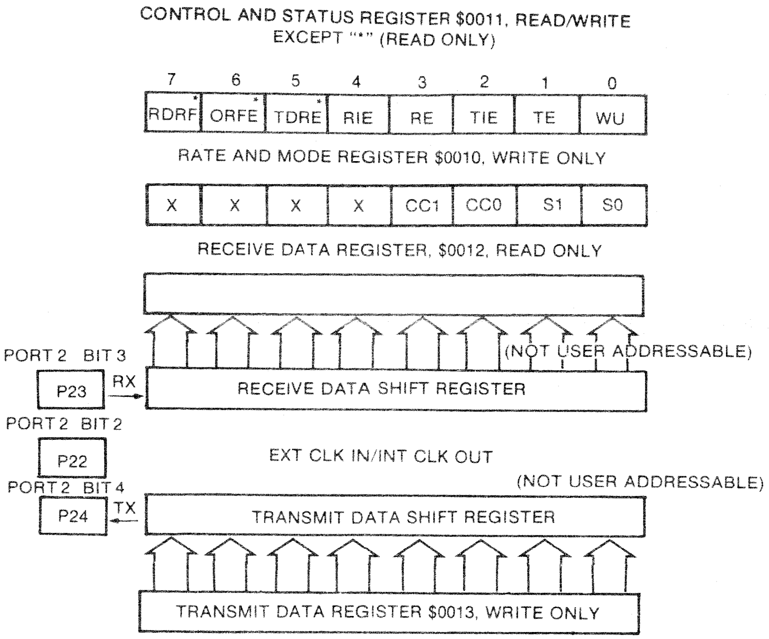
Serial Communications Hardware

The serial communications hardware is controlled by 4 registers as shown in Figure 23. The registers include:

- an 8-bit control and status register
- a 4-bit rate and mode control register (write only)
- an 8-bit read only receive data register and
- an 8-bit write only transmit data register.

In addition to the four registers, the serial I/O section utilizes bit 3 (serial input) and bit 4 (serial output) or Port 2. Bit 2 of Port 2 is utilized if the internal-clock-out or external-clock-in options are selected.

FIGURE 23 — SERIAL I/O REGISTERS


CONTROL AND STATUS REGISTER $0011, READ/WRITE
EXCEPT "*" (READ ONLY)
7
6
5
4
3
2
1
0
RDRF*
ORFE*
TDRE*
RIE
RE
TIE
TE
WU
RATE AND MODE REGISTER $0010, WRITE ONLY
X
X
X
X
CC1
CC0
S1
S0
RECEIVE DATA REGISTER, $0012, READ ONLY
PORT 2 BIT 3
P23
RX
RECEIVE DATA SHIFT REGISTER
(NOT USER ADDRESSABLE)
PORT 2 BIT 2
P22
EXT CLK IN/INT CLK OUT
PORT 2 BIT 4
P24
TX
(NOT USER ADDRESSABLE)
TRANSMIT DATA SHIFT REGISTER
TRANSMIT DATA REGISTER $0013, WRITE ONLY

Transmit/Receive Control and Status (TRCS) Register

The TRCS register consists of an 8-bit register of which all 8 bits may be read while only bits 0-4 may be written. The register is initialized to $20 on $\overline{\text{RESET}}$. The bits in the TRCS register are defined as follows:

7	6	5	4	3	2	1	0	
RDRE	ORFE	TDRE	RIE	RE	TIE	TE	WU	ADDR: $0011

Bit 0 **WU** "Wake-up" on Next Message — set by MC6801 software cleared by hardware on receipt of ten consecutive 1's.

Bit 1 **TE** Transmit Enable — set by MC6801/MC68701 to produce preamble of nine consecutive 1's and to enable gating of transmitter output to Port 2, bit 4 regardless of the DDR value corresponding to this bit; when clear, serial I/O has no effect on Port 2 bit 4.

Bit 2 **TIE** Transmit Interrupt Enable — when set, will permit an $\overline{\text{IRQ2}}$ interrupt to occur when bit 5 (TDRE) is set; when clear, the TDRE value is masked from the bus.

Bit 3 **RE** Receiver Enable — when set, gates Port 2 bit 3 to input of receiver regardless of DDR value for this bit; when clear, serial I/O has no effect on Port 2 bit 3.

Bit 4 **RIE** Receiver Interrpt Enable — when set, will permit an $\overline{\text{IRQ2}}$ interrupt to occur when bit 7 (RDRF) or bit 6 (OR) is set; when clear, the interrupt is masked.

Bit 5 **TDRE** Transmit Data Register Empty — set by hardware when a transfer is made from the transmit data register to the output shift register. The TDRE bit is cleared by reading the status register, then writing a new byte into the transmit data register, TDRE is initialized to 1 by $\overline{\text{RESET}}$.

Bit 6 **ORFE** Over-Run-Framing Error — set by hardware when an overrun or framing error occurs (receive only). An overrun is defined as a new byte received with last byte still in Data Register/Buffer. A framing error has occurred when the byte boundaries in bit stream are not synchronized to bit counter. The ORFE bit is cleared by reading the status register, then reading the Receive Data Register, or by $\overline{\text{RESET}}$.

Bit 7 **RDRF** Receiver Data Register Full — Set by hardware when a transfer from the input shift register to the receiver data register is made. The RDRF bit is cleared by reading the status register, then reading the Receive Data Register, or by $\overline{\text{RESET}}$.

Rate and Mode Control Register

The Rate and Mode Control register controls the following serial I/O variables:

- *Baud rate*
- format
- clocking source, and
- Port 2 bit 2 configuration

The register consists of 4 bits all of which are write-only and cleared on $\overline{\text{RESET}}$. The 4 bits in the register may be considered as a pair of 2-bit fields. The two low order bits control the bit rate for internal clocking and the remaining two bits control the format and clock select logic. The register definition is as follows:

7	6	5	4	3	2	1	0	
X	X	X	X	CC1	CC0	S1	S0	ADDR:$0010

Bit 0 **S0**
Bit 1 **S1**
Speed Select — These bits select the Baud rate for *the internal clock. The four rates which may be* selected are a function of the MPU $\phi 2$ clock frequency. Table 4 lists the available Baud rates.

Bit 2 **CC0**
Bit 3 **CC1**
Clock Control and Format Select — this 2-bit field controls the format and clock select logic. Table 5 *defines the bit field.*

TABLE 4 — SCI INTERNAL BAUD RATES

S1,S0	XTAL	4.0 MHz	4.9152 MHz	2.5476 MHz
	$\phi 2$	1.0 MHz	1.2288 MHz	0.6144 MHz
00	$\phi 2 \div 16$	62.5k Bits/s	76.8k Bits/s	38.4k Bits/s
01	$\phi 2 \div 128$	7,812.5 Bits/s	9,600 Bits/s	4,800 Bits/s
10	$\phi 2 \div 1024$	976.6 Bits/s	1,200 Bits/s	600 Bits/s
11	$\phi 2 \div 4096$	244.1 Bits/s	300 Bits/s	150 Bits/s

TABLE 5 — BIT FIELD

CC1, CC0	Format	Clock Source	Port 2 Bit 2	Port 2 Bit 3	Port 2 Bit 4
00	Bi-Phase	Internal	Not Used	**	**
01	NRZ	Internal	Not Used	**	**
10	NRZ	Internal	Output*	Serial Input	Serial Output
11	NRZ	External	Input	Serial Input	Serial Output

*Clock output is available regardless of values for bits RE and TE.

**Bit 3 is used for serial input if RE = "1" in TRCS; bit 4 is used for serial output if TE = "1" in TRCS.

Internally Generated Clock

If the user wishes for the serial I/O to furnish a clock, the following requirements are applicable:

the values of RE and TE are immaterial.

- the values of RE and TE are immaterial.
- CC1, CC0 must be set to 10
- the maximum clock rate will be $\phi \div 16$.
- the clock will be at 1X the bit rate and will have a rising edge at mid-bit.

Externally Generated Clock

If the user wishes to provide an external clock for the serial I/O, the following requirements are applicable:

- the CC1, CC0, field in the Rate and Mode Control Register must be set to 11,
- the external clock must be set to 8 times (X8) the desired baud rate and
- the maximum external clock frequency is 1.3 MHZ.

SERIAL OPERATIONS

The serial I/O hardware should be initialized by the MC6801 software prior to operation. This sequence will normally consist of:

- writing the desired operation control bits to the Rate and Mode Control Register and
- writing the desired operational control bits in the Transmit/ Receive Control and Status Register.

The Transmitter Enable (TE) and Receiver Enable (RE) bits may be left set for dedicated operations.

Transmit Operations

The transmit operation is enabled by the TE bit in the Transmit/Receive Control and Status Register. This bit when set, gates the output of the serial transmit shift register to Port 2 Bit 4 and takes unconditional control over the Data Direction Register value for Port 2, Bit 4.

Following a $\overline{\text{RESET}}$, the user should configure both the Rate and Mode Control Register and the Transmit/Receiver Control and Status Register for desired operation. Setting the TE bit during tnis procedure initiates the serial output by first transmitting a ten-bit preamble of 1's. Following the preamble, internal synchronization is established and the transmitter section is ready for operation.

At this point one of two situations exist:

a) if the Transmit Data Register is empty (TDRE = 1), a continuous string of ones will be sent indicating an idle line, or

b) if data has been loaded into the Transmit Data Register (TDRE = 0), the word is transferred to the output shift register and transmission of the data word will begin.

During the transfer itself, the 0 start bit is first transmitted. Then the 8 data bits (beginning with bit 0) followed by the stop bit, *are transmitted. When the Transmitter Data Register has* been emptied, the hardware sets the TDRE flag bit.

If the MC6801 fails to respond to the flag within the proper time, (TDRE is still set when the next normal transfer from the parallel data register to the serial output register should occur) then a 1 will be sent (instead of a 0) at "Start" bit time, followed by more 1's until more data is supplied to the data register. No 0's will be sent while TDRE remains a 1.

The Bi-phase mode operates as described above except that the serial output toggles each bit time, and on 1/2 bit times when a 1 is sent.

Receive Operation

The receive operation is enabled by the RE bit which gates in the serial input through Port 2 Bit 3. The receiver section operation is conditioned by the contents of the Transmit/ Receive Control and Status Register and the Rate and Mode *Control Register.*

The receiver bit interval is divided into 8 sub-intervals for internal synchronization. In the standard, non-Bi-phase mode, the received bit stream is synchronized by the first 0 (space) encountered.

The approximate center of each bit time is strobed during the next 10 bits. If the tenth bit is not a 1 (stop bit) a framing error is assumed, and bit ORFE is set. If the tenth bit is a 1, the data is transferred to the Receiver Data Register, and interrupt flag RDRF is set. If RDRF is still set at the next tenth bit time, ORFE will be set, indicating an over-run has occurred. When the MC6801 responds to either flag (RDRF or ORFE) by reading the status register followed by reading the Data Register, RDRF (or ORFE) will be cleared.

RAM CONTROL REGISTER

This register, which is addressed at $0014, gives status information about the standby RAM. A 0 in the RAM enable bit (RAM E) will disable the standby RAM, thereby protecting it at power down if V_{CC} is held greater than V_{SBB} volts, as explained previously in the signal description for V_{CC} Standby.

$0014	STANDBY BIT	RAME	X	X	X	X	X	X

Bit 0 Not Used.
Bit 1 Not Used.
Bit 2 Not used.
Bit 3 Not used.
Bit 4 Not used.
Bit 5 Not used.
Bit 6 The RAM ENABLE control bit allows the user the ability to disable the standby RAM. This bit is set to a logic "one" by reset which enables the standby RAM and can be written to one or zero under program control. When the RAM is disabled, logic "zero", data is read from external memory.
Bit 7 The STANDBY BIT of the control register, $0014, is cleared when the standby voltage is removed. This bit is a read/write status flag that the user can read which indicates that the standby RAM voltage has been applied, and the data in the standby RAM is valid.

FIGURE 24 — MEMORY MAP

The MC6801 provides up to 65k bytes of memory for program and/or data storage. The memory map is shown in Figure 24.

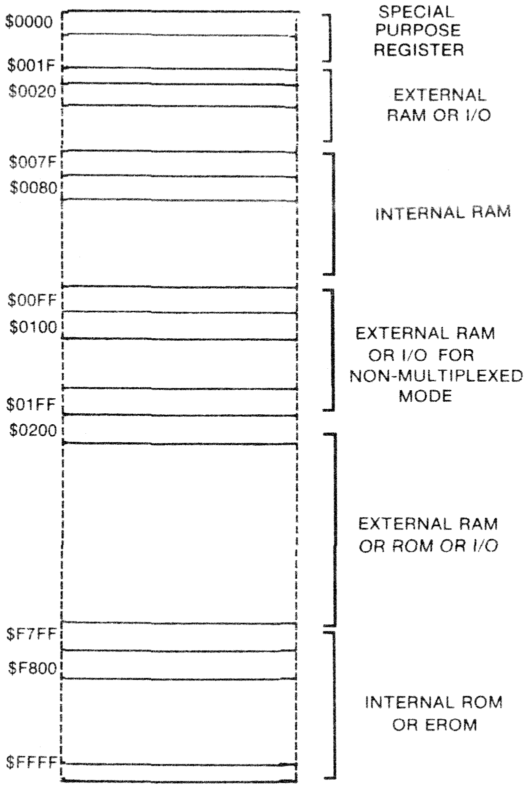


TABLE 6 — SPECIAL REGISTERS

The first 32 bytes are for the special purpose registers as shown in Table 6.

Hex Address	Register
00	Data Direction 1
01	Data Direction 2
02	I/O Port 1
03	I/O Port 2
04	Data Direction 3
05	Data Direction 4
06	I/O Port 3
07	I/O Port 4
08	TCSR
09	Counter High Byte
0A	Counter Low Byte
0B	Output Compare High Byte
0C	Output Compare Low Byte
0D	Input Capture High Byte
0E	Input Capture Low Byte
0F	I/O Port 3 C/S Register
10	Serial Rate and Mode Register
11	Serial Control and Status Register
12	Serial Receiver Data Register
13	Serial Transmit Data Register
14	RAM/EROM Control Register
15-1F	Reserved

FIGURE 25 — MEMORY MAP FOR INTERRUPT VECTORS

	Vector		Description
	MS	LS	
Highest Priority	FFFE,	FFFF	Restart
	FFFC,	FFFD	Non-Maskable Interrupt
	FFFA,	FFFB	Software Interrupt
	FFF8,	FFF9	IRQ1/Interrupt Strobe 3
	FFF6,	FFF7	IRQ2/Timer Input Capture
	FFF4,	FFF5	IRQ2/Timer Output Compare
	FFF2,	FFF3	IRQ2/Timer Overflow
Lowest Priority	FFF0,	FFF1	IRQ2/Serial I/O Interrupt

Locations $0020 through $007F access external RAM or I/O. Internal RAM is accessed at $0080 through $00FF. The RAM may be alternately selected by mask programming at location $A080. However, if the user desires to access external RAM at those locations he may do so by clearing the RAM ENABLE control bit of the RAM Control Register. In this way an extra 128 bytes of external RAM are available. The first 64 bytes of the 128 bytes of on-chip RAM are provided with a separate power supply. This will maintain the 64 bytes of RAM in the power down mode as explained in the pin description for Vcc Standby.

Locations $0100 through $01FF are available in the Expanded Non-Multiplexed Mode. The eight address lines of Port 4 make this 256 word expandability possible. Those not needed for address lines can be used as input lines instead.

The full range of addresses available to the user is in the Expanded Multiplexed Mode. Locations $0200 through $F7FF can be used as external RAM, external ROM, or I/O. Any higher order bits not required for addressing can be used as I/O as in the Expanded Non-Multiplexed Mode.

The internal ROM is located at $F800 through $FFFF. The decoder for the ROM may be mask programmed on A12, and A13 as zeros or one's to provide for $C800, $D800, $E800 for the ROM address. A12 and A13 may also be don't care in this decoder. The primary address for the ROM will be $F800.

GENERAL DESCRIPTION OF INSTRUCTION SET

The MC6801 is upward object code compatible with the MC6800 as it implements the full M6800 instruction set. The execution times of key instructions have been reduced to increase throughput. In addition, new instructions have been added; these include 16-bit operations and a hardware multiply.

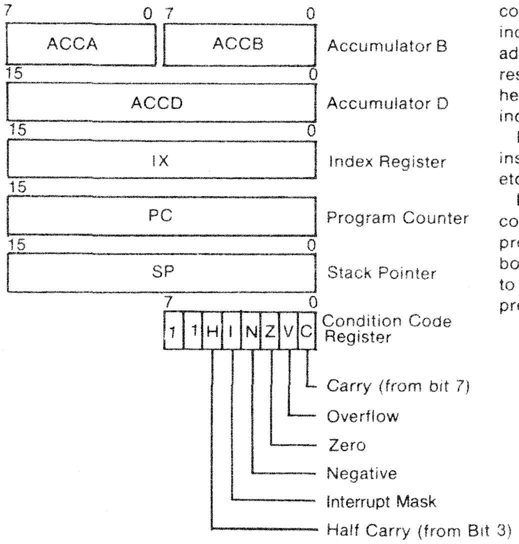
Included in the instruction set section are the following:

- MPU Programming Model (Figure 26)
- Addressing modes
- Accumulator and memory instructions — Table 7
- New instructions
- Index register and stack manipulations — Table 8
- Jump and branch instructions — Table 9
- Special operations — Figure 27
- Condition code register manipulation instructions — Table 10
- Instruction Execution times in machine cycles — Table 11
- Summary of cycle by cycle operation — Table 12

MPU PROGRAMMING MODEL

The programming model for the MC6801 is shown in Figure 26. The double (D) accumulator is physically the same as the A Accumulator concatenated with the B Accumulator so that any operation using accumulator D will destroy information in A and B.

FIGURE 26 — MCU PROGRAMMING MODEL



MPU ADDRESSING MODES

The MC6801 eight-bit microcomputer unit has seven address modes that can be used by a programmer, with the addressing mode a function of both the type of instruction and the coding within the instruction. A summary of the addressing modes for a particular instruction can be found in Table 11 along with the associated instruction execution time that is given in machine cycles. With a clock frequency of 4 MHz, these times would be microseconds.

Accumulator (ACCX) Addressing — In accumulator only addressing, either accumulator A or accumulator B is specified. These are one-byte instructions.

Immediate Addressing — In immediate addressing, the operand is contained in the second byte of the instruction except LDS and LDX which have the operand in the second and third bytes of the instruction. The MCU addresses this location when it fetches the immediate instruction for execution. These are two or three-byte instructions.

Direct Addressing — In direct addressing, the address of the operand is contained in the second byte of the instruction. Direct addressing allows the user to directly address the lowest 256 bytes in the machine i.e., locations zero through 255. Enhanced execution times are achieved by storing data in these locations. In most configurations, it should be a random access memory. These are two-byte instructions.

Extended Addressing — In extended addressing, the address contained in the second byte of the instruction is used as the higher eight-bits of the address of the operand. The third byte of the instruction is used as the lower eight-bits of the address for the operand. This is an absolute address in memory. These are three-byte instructions.

Indexed Addressing — In indexed addressing, the address contained in the second byte of the instruction is added to the index register's lowest eight bits in the MCU. The carry is then added to the higher order eight bits of the index register. This result is then used to address memory. The modified address is held in a temporary address register so there is no change to the index register. These are two-byte instructions.

Implied Addressing — In the implied addressing mode the instruction gives the address (i.e., stack pointer, index register, etc.). These are one-byte instructions.

Relative Addressing — In relative addressing, the address contained in the second byte of the instruction is added to the program counter's lowest eight bits plus two. The carry or borrow is then added to the high eight bits. This allows the user to address data within a range of -125 to +129 bytes of the present instruction. These are two-byte instructions.

TABLE 7—ACCUMULATOR & MEMORY INSTRUCTIONS

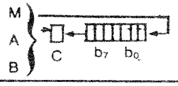
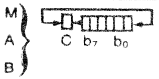
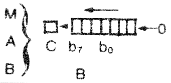
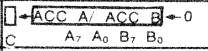
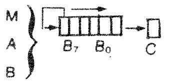
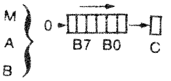
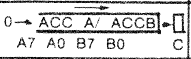
ACCUMULATOR AND MEMORY		ADDRESSING MODES																				
		IMMED.			DIRECT			INDEX			EXTEND			INHERENT				5	4	3	2	1	0
Operations	MNEMONIC	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	Boolean/Arithmetic Operation	H	I	N	Z	V	C
Add	ADDA	8B	2	2	9B	3	2	AB	4	2	BB	4	3				A + M → A	↕	•	↕	↕	↕	↕
	ADDB	CB	2	2	DB	3	2	EB	4	2	FB	4	3				B + M → B	↕	•	↕	↕	↕	↕
Add Double	ADDD	C3	4	3	D3	5	2	E3	6	2	F3	6	3				A:B+M:M+1 → A:B	•	•	↕	↕	↕	↕
Add Accumulators	ABA													1B	2	1	A + B → A	↕	•	↕	↕	↕	↕
Add With Carry	ADCA	89	2	2	99	3	2	A9	4	2	B9	4	3				A + M + C → A		•	↕	↕	↕	↕
	ADCB	C9	2	2	D9	3	2	E9	4	2	F9	4	3				B + M + C → B	↕	•	↕	↕	↕	↕
AND	ANDA	84	2	2	94	3	2	A4	4	2	B4	4	3				A M → A	•	•	↕	↕	R	•
	ANDB	C4	2	2	D4	3	2	E4	4	2	F4	4	3				B M → B	•	•	↕	↕	R	•
Bit Test	BIT A	85	2	2	95	3	2	A5	4	2	B5	4	3				A M	•	•	↕	↕	R	•
	BIT B	C5	2	2	D5	3	2	E5	4	2	F5	4	3				B M	•	•	↕	↕	R	•
Clear	CLR							6F	6	2	7F	6	3				00 → M	•	•	R	S	R	R
	CLRA													4F	2	1	00 → A	•	•	R	S	R	R
	CLRB													5F	2	1	00 → B	•	•	R	S	R	R
Compare	CMPA	81	2	2	91	3	2	A1	4	2	B1	4	3				A - M	•	•	↕	↕	↕	↕
	CMPB	C1	2	2	D1	3	2	E1	4	2	F1	4	3				B - M	•	•	↕	↕	↕	↕
Compare Accumulators	CBA													11	2	1	A - B	•	•	↕	↕	↕	↕
Complement, 1's	COM							63	6	2	73	6	3				M → M	•	•	↕	↕	R	S
	COMA													43	2	1	A → A	•	•	↕	↕	R	S
	COMB													53	2	1	B → B	•	•	↕	↕	R	S
Complement, 2's	NEG							60	6	2	70	6	3				OC - M → M	•	•	↕	↕	①	②
(Negate)	NEGA													40	2	1	00 - A → A	•	•	↕	↕	①	②
NEGB														50	2	1	00 - B → B	•	•	↕	↕	①	②
Decimal Adjust, A	DAA													19	2	1	Converts binary add of BCD characters into BCD format	•	•	↕	↕	↕	③
Decrement	DEC							6A	6	2	7A	6	3				M - 1 → M	•	•	↕	↕	④	•
	DECA													4A	2	1	A - 1 → A	•	•	↕	↕	④	•
	DECB													5A	2	1	B - 1 → B	•	•	↕	↕	④	•
Exclusive OR	EORA	88	2	2	98	3	2	A8	4	2	B8	4	3				A ⊕ M → A	•	•	↕	↕	R	•
	EORB	C8	2	2	D8	3	2	E8	4	2	F8	4	3				B ⊕ M → B	•	•	↕	↕	R	•
Increment	INC							6C	6	2	7C	6	3				M + 1 → M	•	•	↕	↕	⑤	•
	INCA													4C	2	1	A + 1 → A	•	•	↕	↕	⑤	•
	INCB													5C	2	1	B + 1 → B	•	•	↕	↕	⑤	•
Load Accumulator	LDAA	86	2	2	96	3	2	A6	4	2	B6	4	3				M → A	•	•	↕	↕	R	•
	LDAB	C6	2	2	D6	3	2	E6	4	2	F6	4	3				M → B	•	•	↕	↕	R	•
Load Double Accumulator	LDAD	CC	3	3	DC	4	2	EC	5	2	FC	5	3				M → A M + 1 → B	•	•	↕	↕	R	•

The Condition Code Register notes are listed after Table 10.

(Continued)

TABLE 7 — Continued

ADDRESSING MODES

ACCUMULATOR AND MEMORY		IMMED.			DIRECT			INDEX			EXTEND			INHERENT				5	4	3	2	1	0
Operations	MNEMONIC	OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	Boolean/ Arithmetic Operation	H	I	N	Z	V	C
Multiply Unsigned	MUL													3D	10	1	A X B → A:B	•	•	•	•	•	⑬
OR, Inclusive	ORAA	8A	2	2	9A	3	2	AA	4	2	BA	4	3				A + M → A	•	•	↕	↕	R	•
	ORAB	CA	2	2	DA	3	2	EA	4	2	FA	4	3				B + M → B	•	•		↕	R	•
Push Data	PSHA													36	3	1	A → M_{sp}, SP - 1 → SP	•	•	•	•	•	•
	PSHB													37	3	1	B → M_{sp}, SP - 1 → SP	•	•	•	•	•	•
Pull Data	PULA													32	4	1	SP + 1 → SP, M_{sp} → A	•	•	•	•	•	•
	PULB													33	4	1	SP + 1 → SP, M_{sp} → B	•	•	•	•	•	•
Rotate Left	ROL							69	6	2	79	6	3					•	•	↕	↕	⑥	↕
	ROLA													49	2	1		•	•	↕	↕	⑥	↕
	ROLB													59	2	1		•	•	↕	↕	⑥	↕
Rotate Right	ROR							66	6	2	76	6	3					•	•	↕	↕	⑥	↕
	RORA													46	2	1		•	•	↕	↕	⑥	↕
	RORB													56	2	1		•	•	↕	↕	⑥	↕
Shift Left Arithmetic	ASL							68	6	2	78	6	3					•	•	↕	↕	⑥	↕
	ASLA													48	2	1		•	•	↕	↕	⑥	↕
	ASLB													58	2	1		•	•	↕	↕	⑥	↕
Double Shift Left, Arithmetic	ASLD													05	3	1		•	•	↕	↕	⑥	↕
Shift Right Arithmetic	ASR							67	6	2	77	6	3					•	•	↕	↕	⑥	↕
	ASRA													47	2	1		•	•	↕	↕	⑥	↕
	ASRB													57	2	1		•	•	↕	↕	⑥	↕
Shift Right, Logical	LSR							64	6	2	74	6	3					•	•	↕	↕	⑥	↕
	LSRA													44	2	1		•	•	↕	↕	⑥	↕
	LSRB													54	2	1				↕	↕		↕
Double Shift Right Logical	LSRD													04	3	1		• •	• •	R R	↕	⑥ 6	↕
Store Accumulator	STAA				97	3	2	A7	4	2	B7	4	3				A → M	•	•	↕	↕	R	•
	STAB				D7	3	2	E7	4	2	F7	4	3				B → M	•	•	↕	↕	R	•
Store Double Accumulator	STAD				DD	4	2	ED	5	2	FD	5	3				A → M B → M + 1	•	•	↕	↕	R	•
Subtract	SUBA	80	2	2	90	3	2	A0	4	2	B0	4	3				A - M → A	•	•	↕	↕	↕	↕
	SUBB	C0	2	2	D0	3	2	E0	4	2	F0	4	3				B - M B	•	•	↕	↕	↕	↕
Double Subtract	SUBD	83	4	3	93	5	2	A3	6	2	B3	6	3				A:B - M:M + 1 → A:B	•	•	↕	↕	↕	↕
Subtract Accumulators	SBA													10	2	1	A - B → A	•	•	↕	↕	↕	↕
Subtract With Carry	SBCA	82	2	2	92	3	2	A2	4	2	B2	4	3				A - M - C → A	•	•	↕	↕	↕	↕
	SBCB	C2	2	2	D2	3	2	E2	4	2	F2	4	3				B - M - C → B	•	•	↕	↕	↕	↕
Transfer Accumulators	TAB													16	2	1	A → B	•	•	↕	↕	R	•
	TBA													17	2	1	B → A	•	•	↕	↕	R	•
Test Zero or Minus	TST							6D	6	2	7D	6	3				M - 00	•	•	↕	↕	RR	R
	TSTB													5D	2	1	B - 00	•	•	↕	↕	R	R

The Condition Code Register notes are listed after Table 10.

ADDED INSTRUCTIONS

In addition to the existing M6800 Instruction Set, the following new instructions are incorporated in the MC6801 Microcomputer.

ABX Adds the 8-bit unsigned accumulator B to the 16-bit X-Register taking into account the possible carry out of the low order byte of the X-Register. IX ← IX + ACCB

ADDD Adds the double precision ACCD* to the double precision value M:M+1 and places the results in ACCD. ACCD ← (ACCD) + (M:M+1)

ASLD Shifts all bits of ACCAB one place to the left. Bit 0 is loaded with zero. The C bit is loaded from the most significant bit of ACCD.

LDD Loads the contents of double precision memory location into the double accumulator A:B. The condition codes are set according to the data. ACCD ← (M:M+1)

LSRD Shifts all bits of ACCD one place to the right. Bit 15 is loaded with zero. The C bit is loaded from the least significant bit to ACCD.

MUL Multiplies the 8 bits in accumulator A with the 8 bits in accumulator B to obtain a 16-bit unsigned number in A:B. ACCA contains MSB of result. ACCD ← ACCA * ACCB

PSHX The contents of the index register is pushed onto the stack at the address contained in the stack pointer. The stack pointer is decremented by 2. ↓(IXL), SP ← (SP) - 1
↓(IXL), SP ← (SP) - 1

PULX The index register is pulled from the stack beginning at the current address contained in the stack pointer +1. The stack pointer is incremented by 2 in total. SP ← (SP) + 1; IXH
SP ← (SP) + 1; IHL

STD Stores the contents of double accumulator A:B in memory. The contents of ACCD remain unchanged. M:M + 1 ← (ACCD)

SUBD Subtracts the contents of M:M + 1 from the contents of double accumulator AB and places the result in ACCD. ACCAB ← (ACCD) - (M:M + 1)

*ACCD is the 16 bit register (A:B) formed by concatenating the A and B accumulators. The A-accumulator is the most significant byte.

TABLE 8 — INDEX REGISTER AND STACK MANIPULATION INSTRUCTIONS

POINTER OPERATIONS	MNEMONIC	IMMED			DIRECT			INDEX			EXTND			IMPLIED				COND. CODE REG. 5	4	3	2	1	0
		OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#	BOOLEAN/ARITHMETIC OPERATION	H	I	N	Z	V	C
Compare Index Reg	CPX	8C	4	3	9C	5	2	AC	6	2	BC	6	3				$X_H - M, X_L - (M + 1)$	•	•	⑦	↕	⑧	•
Decrement Index Reg	DEX													09	3	1	X − 1 → X	•	•	•	↕	•	•
Decrement Stack Pntr	DES													34	3	1	SP − 1 → SP	•	•	•	•	•	•
Increment Index Reg	INX													08	3	1	X + 1 → X	•	•	•	↕	•	•
Increment Stack Pntr	INS													31	3	1	1 SP + 1 → SP	•	•	•	•	•	•
Load Index Reg	LDX	CE	3	3	DE	4	2	EE	5	2	FE	5	3				$M \rightarrow X_H, (M + 1) \rightarrow X_L$	•	•	⑨	↕	R	•
Load Stack Pntr	LDS	8E	3	3	9E	4	2	AE	5	2	BE	5	3				$M \rightarrow SP_H, (M + 1) \rightarrow SP_L$	•	•	⑨	↕	R	•
Store Index Reg	STX				DF	4	2	EF	5	2	FF	5	3				$X_H \rightarrow M, X_L \rightarrow (M + 1)$	•	•	⑨	↕	R	•
Store Stack Pntr	STS				9F	5	2	AF	7	2	BF	6	3				$SP_H \rightarrow M, SP_L \rightarrow (M + 1)$	•	•	⑨	↕	R	•
Index Reg → Stack Pntr	TXS													35	3	1	X − 1 → SP	•	•	•	•	•	•
Stack Pntr → Index Reg	TSX													30	3	1	SP + 1 → X	•	•	•	•	•	•
Add	ABX													3A	3	1	B + X → X	•	•	•	•	•	•
Push Data	PSHX													3C	4	1	$X_L \rightarrow M_{SP}, SP - 1 \rightarrow SP$ $X_H \rightarrow H_{SP}, SP - 1 \rightarrow SP$	•	•	•	•	•	•
Pull Data	PULX													38	5	1	$SP + 1 \rightarrow SP, M_{SP} \rightarrow X_H$ $SP + 1 \rightarrow SP, M_{SP} \rightarrow X_L$	•	•	•	•	•	•

The Condition Code Register notes are listed after Table 10.

TABLE 9 — JUMP AND BRANCH INSTRUCTIONS

		RELATIVE			INDEX			EXTND			IMPLIED				COND. CODE REG.					
															5	4	3	2	1	0
OPERATIONS	MNEMONIC	OP	~	#	OP	~	#	OP	~	#	OP	~	#	BRANCH TEST	H	I	N	Z	V	C
Branch Always	BRA	20	4	2										None	●	●	●	●	●	●
Branch If Carry Clear	BCC	24	4	2										C = 0	●	●	●	●	●	●
Branch If Carry Set	BCS	25	4	2										C = 1	●	●	●	●	●	●
Branch If = Zero	BEQ	27	4	2										Z = 1	●	●	●	●	●	●
Branch If ≥ Zero	BGE	2C	4	2										N ⊕ V = 0	●	●	●	●	●	●
Branch If > Zero	BGT	2E	4	2										Z + (N ⊕ V) = 0	●	●	●	●	●	●
Branch If Higher	BHI	22	4	2										C + Z = 0	●	●	●	●	●	●
Branch If ≤ Zero	BLE	2F	4	2										Z + (N ⊕ V) = 1	●	●	●	●	●	●
Branch If Lower Or Same	BLS	23	4	2										C + Z = 1	●	●	●	●	●	●
Branch If < Zero	BLT	2D	4	2										N ⊕ V = 1	●	●	●	●	●	●
Branch If Minus	BMI	2B	4	2										N = 1	●	●	●	●	●	●
Branch If Not Equal Zero	BNE	26	4	2										Z = 0	●	●	●	●	●	●
Branch If Overflow Clear	BVC	28	4	2										V = 0	●	●	●	●	●	●
Branch If Overflow Set	BVS	29	4	2										V = 1	●	●	●	●	●	●
Branch If Plus	BPL	2A	4	2										N = 0	●	●	●	●	●	●
Branch To Subroutine	BSR	8D	8	2										See Special Operations	●	●	●	●	●	●
Jump	JMP				6E	4	2	7E	3	3					●	●	●	●	●	●
Jump To Subroutine	JSR				AD	8	2	BD	9	3					●	●	●	●	●	●
No Operation	NOP										01	2	1	Advances Prog. Cntr. Only	●	●	●	●	●	●
Return From Interrupt	RTI										3B	10	1		⑩					
Return From Subroutine	RTS										39	5	1	See Special Operations	●	●	●	●	●	●
Software Interrupt	SWI										3F	12	1		●	●	●	●	●	●
Wait for Interrupt*	WAI										3E	9	1		●	⑪	●	●	●	●

TABLE 10 — CONDITION CODE REGISTER MANIPULATION INSTRUCTIONS

		IMPLIED				COND. CODE REG.					
						5	4	3	2	1	0
OPERATIONS	MNEMONIC	OP	~	#	BOOLEAN OPERATION	H	I	N	Z	V	C
Clear Carry	CLC	0C	2	1	0 → C	●	●	●	●	●	R
Clear Interrupt Mask	CLI	0E	2	1	0 → I	●	R	●	●	●	●
Clear Overflow	CLV	0A	2	1	0 → V	●	●	●	●	R	●
Set Carry	SEC	0D	2	1	1 → C	●	●	●	●	●	S
Set Interrupt Mask	SEI	0F	2	1	1 → I	●	S	●	●	●	●
Set Overflow	SEV	0B	2	1	1 → V	●	●	●	●	S	●
Accumulator A → CCR	TAP	06	2	1	A → CCR	⑫					
CCR → Accumulator A	TPA	07	2	1	CCR → A	●	●	●	●	●	●

CONDITION CODE REGISTER NOTES: (Bit set if test is true and cleared otherwise)

1	(Bit V)	Test: Result = 10000000?
2	(Bit C)	Test: Result = 00000000?
3	(Bit C)	Test: Decimal value of most significant BCD Character greater than nine? (Not cleared if previously set.)
4	(Bit V)	Test: Operand = 10000000 prior to execution?
5	(Bit V)	Test: Operand = 01111111 prior to execution?
6	(Bit V)	Test: Set equal to result of N⊕C after shift has occurred.
7	(Bit N)	Test: Sign bit of most significant (MS) byte = 1?
8	(Bit V)	Test: 2's complement overflow from subtraction of MS bytes?
9	(Bit N)	Test: Result less than zero? (Bit 15 = 1)
10	(All)	Load Condition Code Register from Stack. (See Special Operations)
11	(Bit I)	Set when interrupt occurs. If previously set, a Non-Maskable Interrupt is required to exit the wait state.
12	(All)	Set according to the contents of Accumulator A.

FIGURE 27 — SPECIAL OPERATIONS

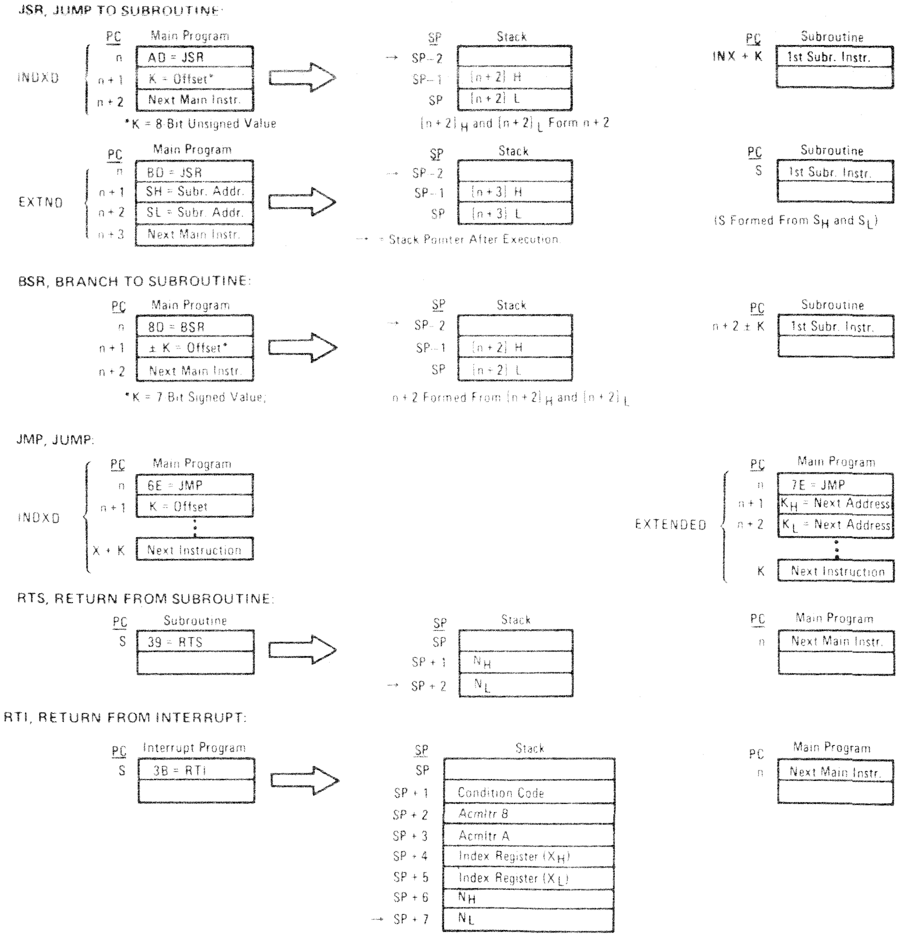


TABLE 11 — INSTRUCTION EXECUTION TIMES IN MACHINE CYCLE

	ACCX	Immediate	Direct	Extended	Indexed	Inherent	Relative
ABA	•	•	•	•	•	2	•
ABX	•	•	•	•	•	3	•
ADC	•	2	3	4	4	•	•
ADD	•	2	3	4	4	•	•
ADDD	•	4	5	6	6	•	•
AND	•	2	3	4	4	•	•
ASL	2	•	•	6	6	•	•
ASLD	•	•	•	•	•	3	•
ASR	2	•	•	6	6	•	•
BCC	•	•	•	•	•	•	3
BCS	•	•	•	•	•	•	3
BEQ	•	•	•	•	•	•	3
BGE	•	•	•	•	•	•	3
BGT	•	•	•	•	•	•	3
BHI	•	•	•	•	•	•	3
BIT	•	2	3	4	4	•	•
BLE	•	•	•	•	•	•	3
BLS	•	•	•	•	•	•	3
BLT	•	•	•	•	•	•	3
BMI	•	•	•	•	•	•	3
BNE	•	•	•	•	•	•	3
BPL	•	•	•	•	•	•	3
BRA	•	•	•	•	•	•	3
BSR	•	•	•	•	•	•	6
BVC	•	•	•	•	•	•	3
BVS	•	•	•	•	•	•	3
CBA	•	•	•	•	•	2	•
CLC	•	•	•	•	•	2	•
CLI	•	•	•	•	•	2	•
CLR	2	•	•	6	6	•	•
CLV	•	•	•	•	•	2	•
CMP	•	2	3	4	4	•	•
COM	2	•	•	6	6	•	•
CPX	•	4	5	6	6	•	•
DAA	•	•	•	•	•	2	•
DEC	2	•	•	6	6	•	•
DES	•	•	•	•	•	3	•
DEX	•	•	•	•	•	3	•
EOR	•	2	3	4	4	•	•
INC	2	•	•	6	6	•	•
INS	•	•	•	•	•	3	•

	ACCX	Immediate	Direct	Extended	Indexed	Inherent	Relative
INX	•	•	•	•	•	3	•
JMP	•	•	•	3	3	•	•
JSR	•	•	5	6	6	•	•
LDA	•	2	3	4	4	•	•
LDD	•	3	4	5	5	•	•
LDS	•	3	4	5	5	•	•
LDX	•	3	4	5	5	•	•
LSR	2	•	•	6	6	•	•
LSRD	•	•	•	•	•	3	•
MUL	•	•	•	•	•	10	•
NEG	2	•	•	6	6	•	•
NOP	•	•	•	•	•	2	•
ORA	•	2	3	4	4	•	•
PSH	3	•	•	•	•	•	•
PSHX	•	•	•	•	•	4	•
PUL	4	•	•	•	•	•	•
PULX	•	•	•	•	•	5	•
ROL	2	•	•	6	6	•	•
ROR	2	•	•	6	6	•	•
RTI	•	•	•	•	•	10	•
RTS	•	•	•	•	•	5	•
SBA	•	•	•	•	•	2	•
SBC	•	2	3	4	4	•	•
SEC	•	•	•	•	•	2	•
SEI	•	•	•	•	•	2	•
SEV	•	•	•	•	•	2	•
STA	•	•	3	4	4	•	•
STD	•	•	4	5	5	•	•
STS	•	•	4	5	5	•	•
STX	•	•	4	5	5	•	•
SUB	•	2	3	4	4	•	•
SUBD	•	4	5	6	6	•	•
SWI	•	•	•	•	•	12	•
TAB	•	•	•	•	•	2	•
TAP	•	•	•	•	•	2	•
TBA	•	•	•	•	•	2	•
TPA	•	•	•	•	•	2	•
TST	2	•	•	6	6	•	•
TSX	•	•	•	•	•	3	•
TXS	•	•	•	•	•	3	•
WAI	•	•	•	•	•	9	•

Summary of Cycle by Cycle Operation

Table 12 provides a detailed description of the information present on the Address Bus, Data Bus, and the Read/Write line (R/W) during each cycle for each instruction.

This information is useful in comparing actual with expected results during debug of both software and hardware as the control program is executed. The information is categorized in groups according to addressing mode and number of cycles per instruction. (In general, instructions with the same addressing mode and number of cycles execute in the same manner; exceptions are indicated in the table).

TABLE 12 — CYCLE BY CYCLE OPERATION

ADDRESS MODE & INSTRUCTIONS	CYCLE	CYCLE #	ADDRESS BUS	R/W LINE	DATA BUS
IMMEDIATE					
ADC EOR	2	1	Op Code Address	1	Op Code
ADD LDA		2	Op Code Address + 1	1	Operand Data
AND ORA					
BIT SBC					
CMP SUB					
LDS	3	1	Op Code Address	1	Op Code
LDX		2	Op Code Address + 1	1	Operand Data (High Order Byte)
		3	Op Code Address + 2	1	Operand Data (Low Order Byte)
CPX	4	1	Op Code Address	1	Op Code
SUBD		2	Op Code Address + 1	1	Operand Data (High Order Byte)
ADDD		3	Op Code Address + 2	1	Operand Data (Low Order Byte)
		4	Address Bus FFFF	1	Low Byte of Restart Vector
DIRECT					
ADC EOR	3	1	Op Code Address	1	Op Code
ADD LDA		2	Op Code Address + 1	1	Address of Operand
AND ORA		3	Address of Operand	1	Operand Data
BIT SBC					
CMP SUB					
STA	3	1	Op Code Address	1	Op Code
		2	Op Code Address + 1	1	Destination Address
		3	Destination Address	0	Data from Accumulator
LDS	4	1	Op Code Address	1	Op Code
LDX		2	Op Code Address + 1	1	Address of Operand
LDD		3	Address of Operand	1	Operand Data (High Order Byte)
		4	Operand Address + 1	1	Operand Data (Low Order Byte)
STS	4	1	Op Code Address	1	Op Code
STX		2	Op Code Address + 1	1	Address of Operand
STD		3	Address of Operand	0	Register Data (High Order Byte)
		4	Address of Operand + 1	0	Register Data (Low Order Byte)
CPX	5	1	Op Code Address	1	Op Code
SUBD		2	Op Code Address + 1	1	Address of Operand
ADDD		3	Operand Address	1	Operand Data (High Order Byte)
		4	Operand Address + 1	1	Operand Data (Low Order Byte)
		5	Address Bus FFFF	1	Low Byte of Restart Vector
JSR	5	1	Op Code Address	1	Op Code
		2	Op Code Address + 1	1	Irrelevant Data
		3	Subroutine Address	1	First Subroutine Op Code
		4	Stack Pointer	0	Return Address (Low Order Byte)
		5	Stack Pointer + 1	0	Return Address (High Order Byte)

(continued)

TABLE 12 — CYCLE BY CYCLE OPERATION
(cont)

ADDRESS MODE & INSTRUCTIONS	CYCLES	CYCLE #	ADDRESS BUS	R/W LINE	DATA BUS
EXTENDED					
JMP	3	1	Op Code Address	1	Op Code
		2	Op Code Address + 1	1	Jump Address (High Order Byte)
		3	Op Code Address + 2	1	Jump Address (Low Order Byte)
ADC EOR	4	1	Op Code Address	1	Op Code
ADD LDA		2	Op Code Address + 1	1	Address of Operand
AND ORA		3	Op Code Address + 2	1	Address of Operand (Low Order Byte)
BIT SBC		4	Address of Operand	1	Operand Data
CMP SUB					
STA A	4	1	Op Code Address	1	Op Code
STA B		2	Op Code Address + 1	1	Destination Address (High Order Byte)
		3	Op Code Address + 2	1	Destination Address (Low Order Byte)
		4	Operand Destination Address	0	Data from Accumulator
LDS	5	1	Op Code Address	1	Op Code
LDX		2	Op Code Address + 1	1	Address of Operand (High Order Byte)
LDD		3	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	Address of Operand	1	Operand Data (High Order Byte)
		5	Address of Operand + 1	1	Operand Data (Low Order Byte)
STS	5	1	Op Code Address	1	Op Code
STX		2	Op Code Address + 1	1	Address of Operand (High Order Byte)
STD		3	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	Address of Operand	0	Operand Data (High Order Byte)
		5	Address of Operand + 1	0	Operand Data (Low Order Byte)
ASL LSR	6	1	Op Code Address	1	Op Code
ASR NEG		2	Op Code Address + 1	1	Address of Operand (High Order Byte)
CLR ROL		3	Op Code Address + 2	1	Address of Operand (Low Order Byte)
COM ROR		4	Address of Operand	1	Current Operand Data
DEC TST (1)		5	Address Bus FFFF	1	Low Byte of Restart Vector
INC		6	Address of Operand	0	New Operand Data
CPX	6	1	Op Code Address	1	Op Code
SUBD		2	Op Code Address + 1	1	Operand Address (High Order Byte)
ADDD		3	Op code Address + 2	1	Operand Address (Low Order Byte)
		4	Operand Address	1	Operand Data (High Order Byte)
		5	Operand Address + 1	1	Operand Data (Low Order Byte)
		6	Address Bus FFFF	1	Low Byte of Restart Vector
JSR	6	1	Op Code Address	1	Op Code
		2	Op Code Address + 1	1	Address of Subroutine (High Order Byte)
		3	Op Code Address + 2	1	Address of Subroutine (Low Order Byte)
		4	Subroutine Starting Address	1	Op Code of Next Instruction
		5	Stack Pointer	0	Return Address (Low Order Byte)
		6	Stack Pointer - 1	0	Address of Operand (High Order Byte)

(continued)

TABLE 12 — CYCLE BY CYCLE OPERATION
(cont)

ADDRESS MODE & INSTRUCTIONS	CYCLES	CYCLE #	ADDRESS BUS	R/W LINE	DATA BUS
INDEXED					
JMP	3	1	Op Code Address	1	Op Code
		2	Op Code Address + 1	1	Offset
		3	Address Bus FFFF	1	Low Byte of Restart Vector
ADC EOR	4	1	Op Code Address	1	Op Code
ADD LDA		2	Op Code Address + 1	1	Offset
AND ORA		3	Address Bus FFFF	1	Low Byte of Restart Vector
BIT SBC		4	Index Register Plus Offset	1	Operand Data
CMP SUB					
STA	4	1	Op Code Address	1	Op Code
		2	Op Code Address + 1	1	Offset
		3	Address Bus FFFF	1	Low Byte of Restart Vector
		4	Index Register Plus Offset	0	Operand Data
LDS	5	1	Op Code Address	1	Op Code
LDX		2	Op Code Address + 1	1	Offset
LDD		3	Address Bus FFFF	1	Low Byte of Restart Vector
		4	Index Register Plus Offset	1	Operand Data (High Order Byte)
		5	Index Register Plus Offset + 1	1	Operand Data (Low Order Byte)
STS	5	1	Op Code Address	1	Op Code
STX		2	Op Code Address + 1	1	Offset
STD		3	Address Bus FFFF	1	Low Byte of Restart Vector
		4	Index Register Plus Offset	0	Operand Data (High Order Byte)
		5	Index Register Plus Offset + 1	0	Operand Data (Low Order Byte)
ASL LSR	6	1	Op Code Address	1	Op Code
ASR NEG		2	Op Code Address + 1	1	Offset
CLR ROL		3	Address Bus FFFF	1	Low Byte of Restart Vector
COM ROR		4	Index Register Plus Offset	1	Current Operand Data
DEC TST (1)		5	Address Bus FFFF	1	Low Byte of Restart Vector
INC		6	Index Register Plus Offset	0	New Operand Data
CPX	6	1	Op Code Address	1	Op Code
SUBD		2	Op Code Address + 1	1	Offset
ADDD		3	Address Bus FFFF	1	Low Byte of Restart Vector
		4	Index Register + Offset	1	Operand Data (High Order Byte)
		5	Index Register + Offset + 1	1	Operand Data (Low Order Byte)
		6	Address Bus FFFF		Low Byte of Restart Vector
JSR	6	1	Op Code Address	1	Op Code
		2	Op Code Address + 1	1	Offset
		3	Address Bus FFFF	1	Low Byte of Restart Vector
		4	Index Register + Offset	1	First Subroutine Op Code
		5	Stack Pointer	0	Return Address (Low Order Byte)
		6	Stack Pointer - 1	0	Return Address (High Order Byte)

(continued)

TABLE 12 — CYCLE BY CYCLE OPERATION
(cont)

ADDRESS MODE & INSTRUCTIONS	CYCLES	CYCLES #	ADDRESS BUS	R/W LINE	DATA BUS
INHERENT					
ABA DAA SEC	2	1	Op Code Address	1	Op Code
ASL DEC SEI		2	Op Code Address +1	1	Op Code of Next Instruction
ASR INC SEV					
CBA LSR TAB					
CLC NEG TAP					
CLI NOP TBA					
CLR ROL TPA					
CLV ROR TST					
COM SBA					
ABX	3	1	Op Code Address	1	Op Code
		2	Op Code Address +1	1	Irrelevent Data
		3	Address Bus FFFF	1	Low Byte of Restart Vector
ASLD	3	1	Op Code Address	1	Op Code
LSRD		2	Op Code Address +1	1	Irrevelant Data
		3	Address Bus FFFF	1	Low Byte of Restart Vector
DES	3	1	Op Code Address	1	Op Code
INS		2	Op Code Address +1	1	Op Code of Next Instruction
		3	Previous Register Contents	1	Irrelevant Data
INX	3	1	Op Code Address	1	Op Code
DEX		2	Op Code Address +1	1	Op Code of Next Instruction
		3	Address Bus FFFF	1	Low Byte of Restart Vector
PSHA	3	1	Op Code Address	1	Op Code
PSHB		2	Op Code Address +1	1	Op Code of Next Instruction
		3	Stack Pointer	0	Accumulator Data
ISX	3	1	Op Code Address	1	Op Code
		2	Op Code Address +1	1	Op Code of Next Instruction
		3	Stack Pointer	1	Irrelevant Data
TXS	3	1	Op Code Address	1	Op Code
		2	Op Code Address +1	1	Op Code of Next Instruction
		3	Address Bus FFFF	1	Low Byte of Restart Vector
PULA	4	1	Op Code Address	1	Op Code
PULB		2	Op Code Address +1	1	Op Code of Next Instruction
		3	Stack Pointer	1	Irrelevant Data
		4	Stack Pointer +1	1	
PSHX	4	1	Op Code Address	1	Op Code
		2	Op Code Address +1	1	Irrelevant Data
		3	Stack Pointer	0	Index Register (Low Order Byte)
		4	Stack Pointer -1	0	Index Register (High Order Byte)
PULX	5	1	Op Code Address	1	Op Code
		2	Op Code Address +1	1	Irrelevant Data
		3	Stack Pointer	1	Irrelevant Data
		4	Stack Pointer +1	1	Index Register (High Order Byte)
5		5	Stack Pointer +2	1	Index Register (Low Order Byte)
BCC BHT BNE	3	1	Op Code Address	1	Op Code
BCS BLE BPL		2	Op Code Address +1	1	Branch Offset
BEQ BLS BRA		3	Address Bus FFFF	1	Low Byte of Restart Vector
BGE BLT BVC					
BGT BMT BVS					
BSR	6	1	Op Code Address	1	Op Code
		2	Op Code Address +1	1	Branch Offset
		3	Address Bus FFFF	1	Low Byte of Restart Vector
		4	Subroutine Starting Address	1	Op Code of Next Instruction
		5	Stack Pointer	0	Return Address (Low Order Byte)
		6	Stack Pointer -1	0	Return Address (High Order Byte)

FIGURE 28 — MC6801 MCU SINGLE-CHIP DUAL PROCESSOR CONFIGURATION

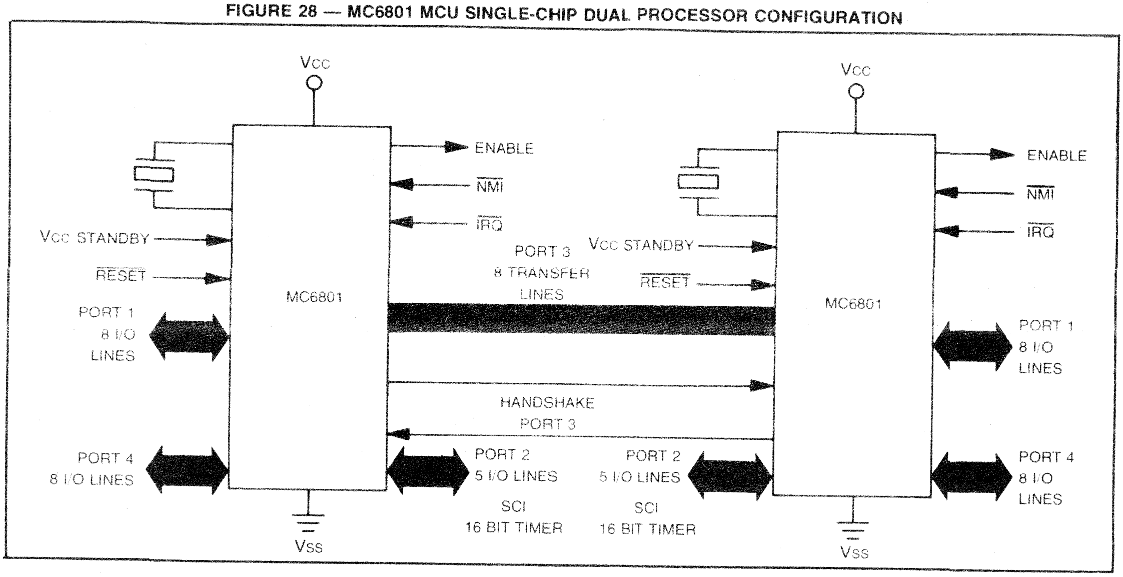


FIGURE 29 — MC6801 MCU
EXPANDED NON-MULTIPLEXED MODE

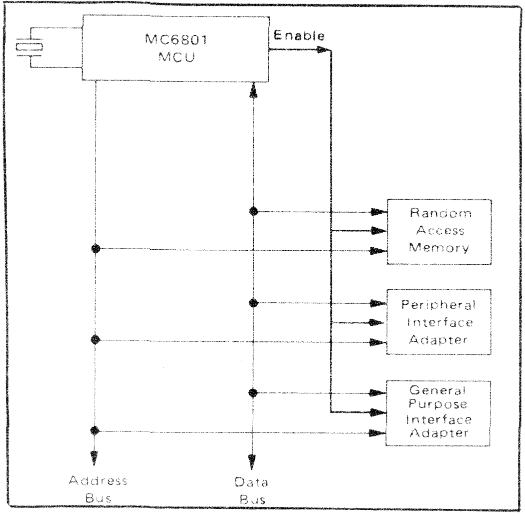
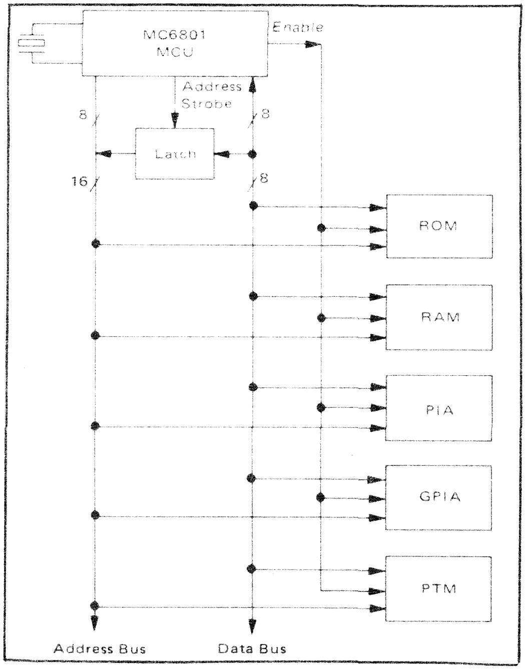
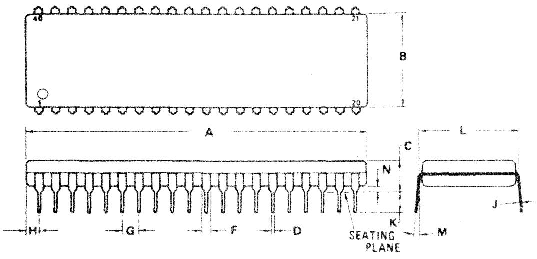
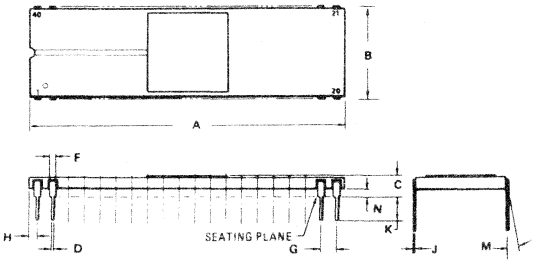


FIGURE 30 — MC6801 MCU
EXPANDED MULTIPLEXED MODE



OUTLINE DIMENSIONS





	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	50.29	51.31	1.980	2.020
B	14.86	15.62	0.585	0.615
C	2.54	4.19	0.100	0.165
D	0.38	0.53	0.015	0.021
F	0.76	1.40	0.030	0.055
G	2.54 BSC		0.100 BSC	
H	0.76	1.78	0.030	0.070
J	0.20	0.33	0.008	0.013
K	2.54	4.19	0.100	0.165
M	0°	10°	0°	10°
N	0.51	1.52	0.020	0.060

L SUFFIX
CERAMIC PACKAGE
CASE 715-02

NOTE:
1. LEADS, TRUE POSITIONED WITHIN 0.25 mm (0.010) DIA (AT SEATING PLANE), AT MAX. MAT'L CONDITION.

	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	51.82	52.32	2.040	2.060
B	13.72	14.22	0.540	0.560
C	4.57	5.08	0.180	0.200
D	0.36	0.51	0.014	0.020
F	1.02	1.52	0.040	0.060
G	2.54 BSC		0.100 BSC	
H	1.65	2.16	0.065	0.085
J	0.20	0.30	0.008	0.012
K	3.05	3.56	0.120	0.140
L	15.24 BSC		0.600 BSC	
M	0°	10°	0°	10°
N	0.51	1.02	0.020	0.040

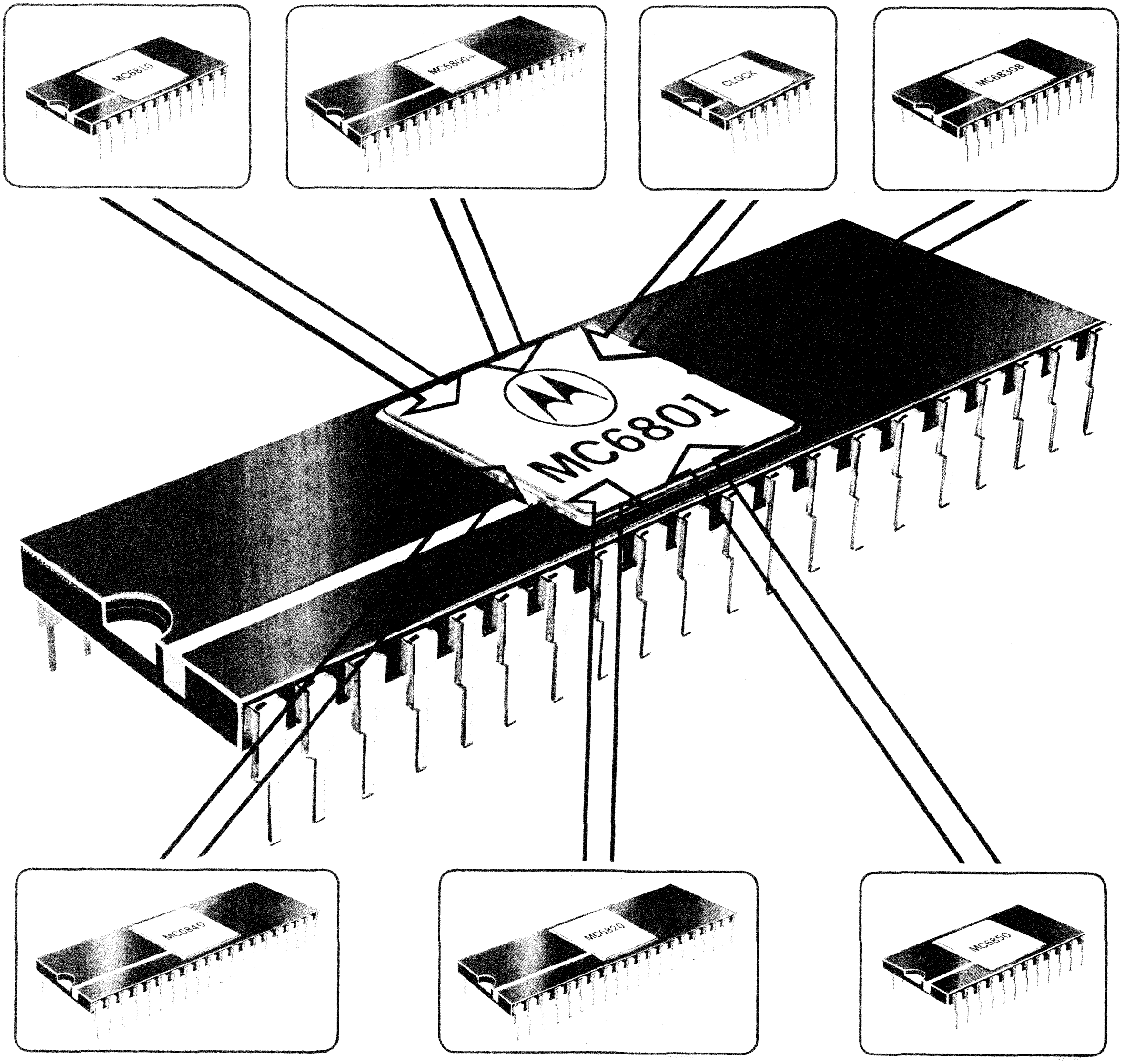
P SUFFIX
PLASTIC PACKAGE
CASE 711-02

NOTES:
1. LEADS TRUE POSITIONED WITHIN 0.25 mm (0.010) DIA AT SEATING PLANE AT MAXIMUM MATERIAL CONDITION (DIM "D").
2. DIM "L" TO CENTER OF LEADS WHEN FORMED PARALLEL.

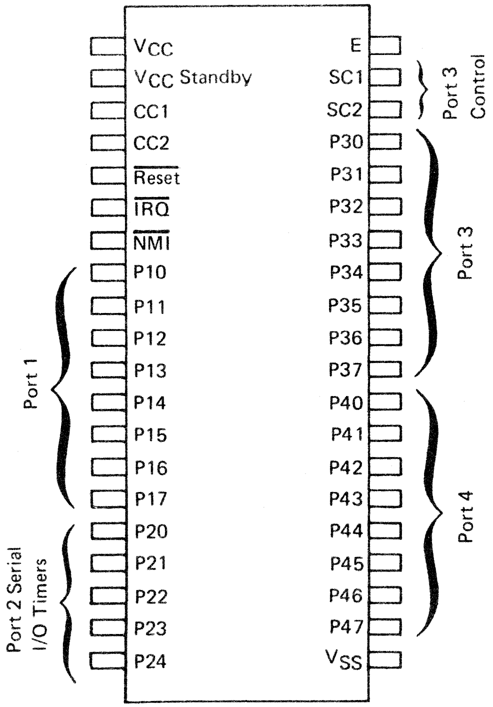
Motorola reserves the right to make changes to any products herein to improve reliability, function or design. Although the information in this document has been carefully reviewed for broad application, Motorola does not assume any liability arising out of the application or use of any product or circuit described herein neither does it convey any license under its patent rights nor the rights of others.

MOTOROLA

microcomputer fact book



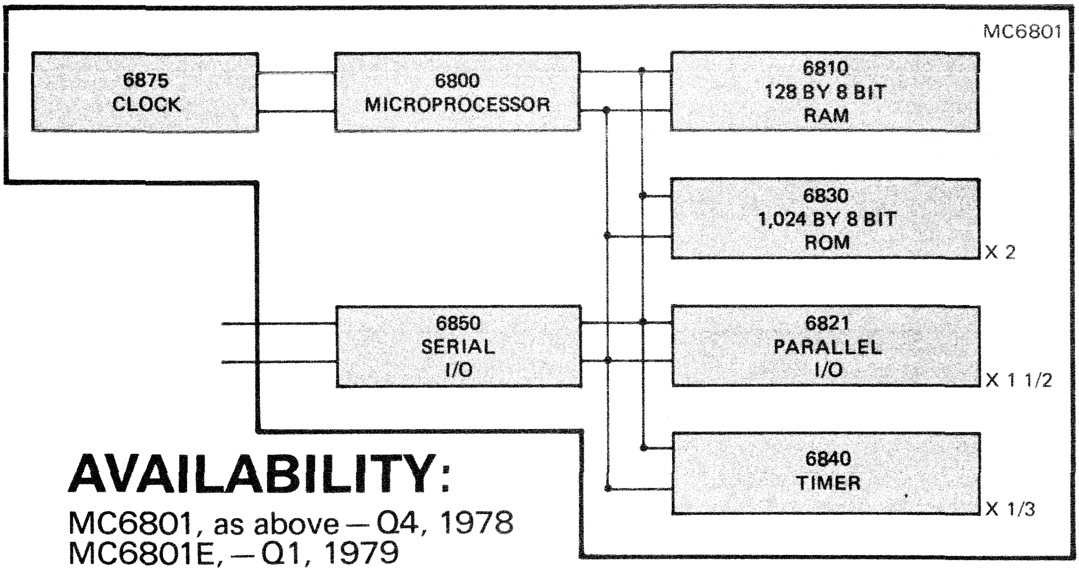
MC6801 pinout diagram



Introducing...

the motorola MC6801

a complete microcomputer on a chip



AVAILABILITY:

MC6801, as above — Q4, 1978
MC6801E, — Q1, 1979
as above, but wired for
external clock operation
MC6803, No-ROM version — Q4, 1978
MC68701, EPROM version — Q1, 1979

FEATURES:

Architectural

- 128 Bytes of On-Chip RAM
  64 Bytes Retainable
  128 Bytes Retainable Option
- 2K Bytes of On-Chip ROM
  No-ROM Version Option
  EPROM Version Option
- 33 Parallel I/O Data Lines
- On-Chip Three-Function Timer
- On-Chip Serial I/O Port
- On-Chip Clock Oscillator/Driver
  External or Internal Clock Option

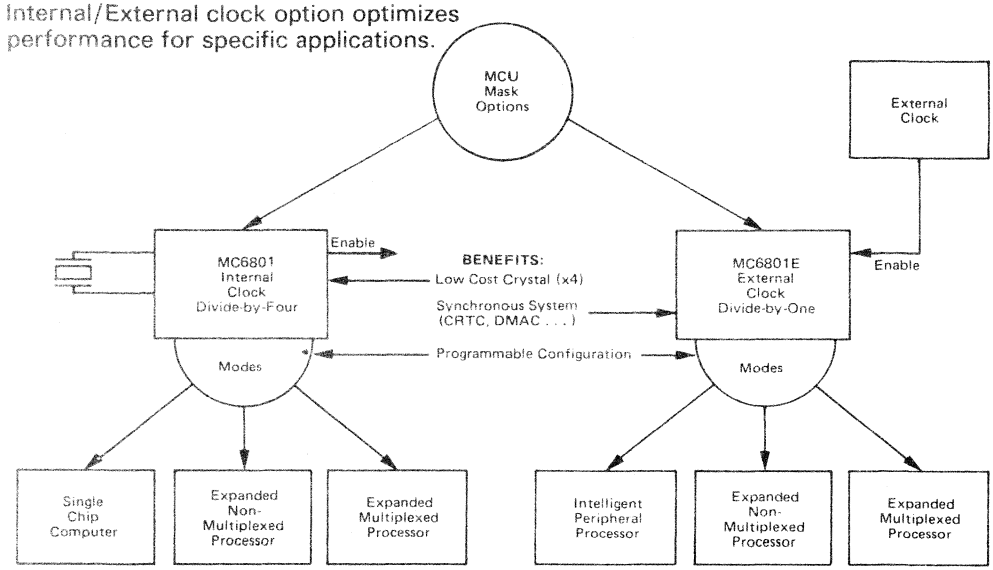
Functional

- 25% Higher Performance than MC6800
  5 V Operation
  M6800-Compatible OP CODE
  Lower System Cost
  80% Component Reduction
- Expandable with all M6800-Compatible peripheral chips
- 64K External Addressability
- 16-Bit Instructions
  (Load, Store, Add, Sub, Psh, Pul, Shift)

functional highlights

CLOCK OPTIONS

Internal/External clock option optimizes performance for specific applications.

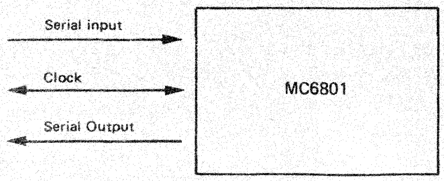


SERIAL I/O

for low-cost communications including multiprocessor operation

FEATURES:

- 8-Bit asynchronous (1 Start, 8-Bits, 1 Stop)
- Full Duplex, double buffered
- Internal or external clock
- Standard or bi-phase (F/2F) Data

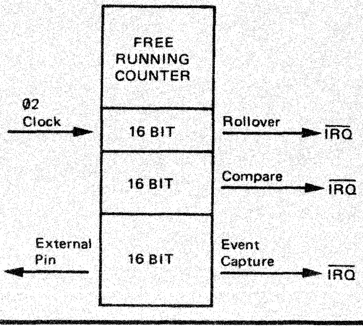


ON-CHIP TIMER

Measures signal time, generates output signals, permits periodic interrupts

FEATURES:

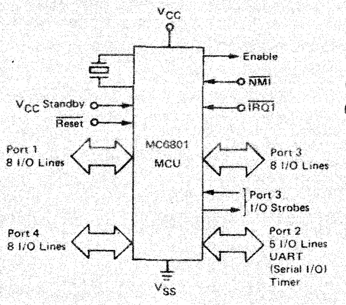
- 3 timer functions
- Separate internal interrupts
- 16-bit timer
- Timing derived from Ø2



processor operating modes

SINGLE-CHIP COMPUTER

The most powerful single-chip computer made.



FEATURES

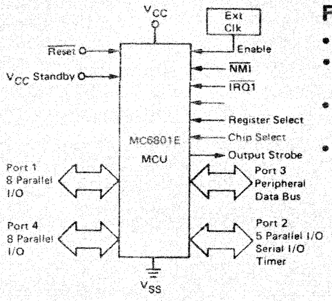
- 29 Parallel I/O
- 2 Interrupts
- 2 Handshake

OR

- 24 Parallel I/O
- 2 Interrupts
- 2 Handshake
- Serial I/O & Timer

PERIPHERAL PROCESSOR

Intelligent peripherals now possible

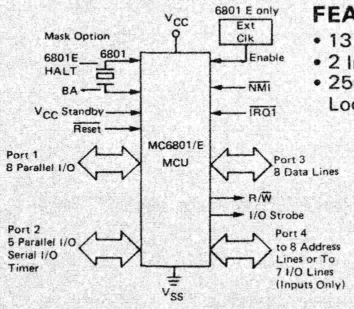


FEATURES

- 23 Parallel I/O
- Serial I/O & Timer
- Interrupt or Polled Handshake
- Operates as a peripheral on the M6800 family bus

EXPANDED NON-MULTIPLEXED MODE

Works with all M6800 peripherals to provide special functions

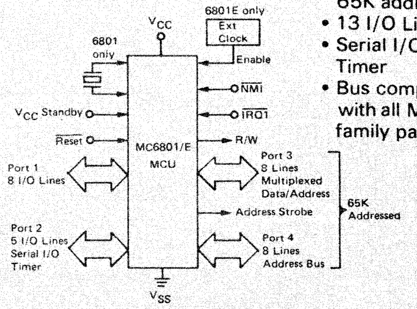


FEATURES

- 13 Parallel I/O
- 2 Interrupts
- 256 Addressable Locations

EXPANDED MULTIPLEXED MODE

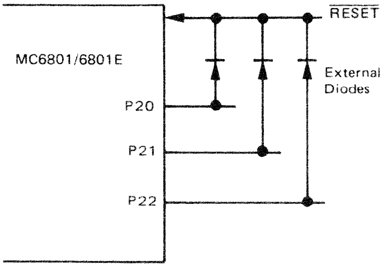
Large system capabilities



FEATURES

- Full expansion 65K addresses
- 13 I/O Lines
- Serial I/O & Timer
- Bus compatible with all MC6800 family parts

HARDWARE PROGRAMMABLE OPTIONS



P22	P21	P20	OPERATIONAL MODE
0	0	0	TEST-DATA OUT PUTTED FROM ROM & RAM TO I/O PORT 3
0	0	1	64K ADDRESS TEST AT I/O PORTS 3 & 4
0	1	0	
0	1	1	
1	0	0	SINGLE CHIP TEST
1	0	1	EXPANDED NON-MULTIPLEXED
1	1	0	EXPANDED MULTIPLEXED
1	1	1	SINGLE CHIP / PERIPHERAL PROCESSOR

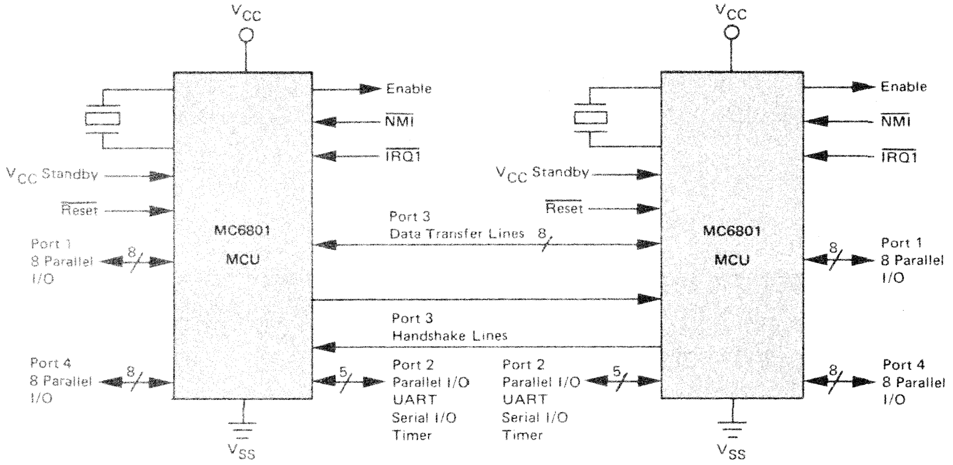
1 = No diode in circuit   0 = diode in circuit

multiprocessing operation

The most powerful two-chip set to date

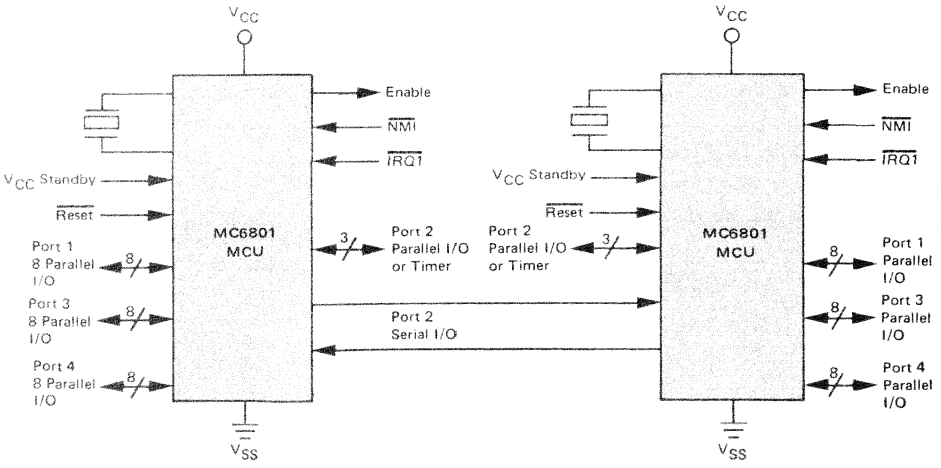
PARALLEL INTERCONNECT FEATURES

- 46 Parallel I/O Lines
- 2 Serial I/O Lines & 2 Timers
- 2 Processors
- 4K Bytes ROM, 256 Bytes RAM



SERIAL INTERCONNECT FEATURES

- 62 Parallel I/O Lines
- 2 Timers
- 2 Processors
- 4K Bytes ROM, 256 Bytes RAM



MC6800 plus...

COST SAVINGS — DUE TO:

- High-Level Integration
- Single Power Supply Operation
- Lower Inventory
- Lower Total Parts Cost
- Lower PC Board Cost (size and number of insertion holes)
- Lower Component Insertion Cost
- Lower System Manufacturing Cost

IMPROVEMENTS FOR EQUIVALENT FUNCTIONAL CONTENT

Benefits	6801 System	6802 System	6800 System
Lower Power	900MW	1625MW	1805MW
Lower Chip Count	1	4	6
Parts Required	MC6801	MC6802 6846 6850 6821	MC6800 6875 6846 6850 6821 6810

SOFTWARE IMPROVEMENT

Not only the most powerful hardware but an upward compatible enhanced 6800.

Instructions Reduced 3 Cycles
JMP To Subroutine Extended

Instructions Reduced 2 Cycles
JMP To Subroutine Index
Branch To Subroutine

Instructions Reduced 1 Cycle
All Indexed Addressing
All Branches
STAA, STAB, STX, STS
INX, INS, DEX, DES
PSHA, PSHB
TSX

Instructions Increased 1 Cycle
CPX (now sets all flags)

16 Bit Operations
Load And Store D-Register (ACCB:ACCA)
Arithmatic Shifts Left And Right
Add And Subtract D-Register
Add B-Register To X-Register

8 x 8 Multiply With 16-Bit Result

MC6801 SUPPORT

HARDWARE SUPPORT

The 6801 Support System is EXORciser/EXORterm compatible and is capable of operating with any of the existing M6800 MPU modules allowing MC6801 software and hardware development in either the EXORciser I/II or EXORterm. The Support System consists of a control module, an extender cable terminated by a 40 pin male DIP plug and control software. This allows the user to evaluate and debug his own MC6801 system in either the single-chip, expanded-multiplexed or non-multiplexed mode.

FEATURES:

- EXORciser/EXORterm Compatible
- User System Evaluator (U.S.E.) capability
- Current Product Compatibility
- Multi-Processor Development
- Real time emulation of MC6801 in all modes

SOFTWARE SUPPORT

Software supported with a disk based Cross Macro Assembler, Editor, and Debug package. The MC6801 software is developed in the EXORciser and down loaded into the MC6801 support module. Then the Debug package controls the execution and provides trace, breakpoint and modification commands using familiar EXORciser command syntax.

TRAINING

The MC6801 is supported by two training courses:

- One day training course for previous M6800 family users.
- Three day training course for new M6800 family users.

competitive analysis

MC6801 vs Intel

	MOTOROLA	INTEL	
	MC6801	I8049	I8085
Single Chip	Yes	Yes	No (3 chips)
Full Capability Serial Comm.	Yes	No	No
Timer	16 bit	8 bit	14 bit
Instruction Execution timing	Simple	Simple	Complex
Distributed Processing	Yes	No	Complex
Intelligent Peripheral	Yes	No	No
Compatible with all peripherals	Yes	Yes	Yes
Expandability	Yes	Limited	Yes
I/O Instructions	All memory instructions	Special instruction (Limited)	Special instructions (Limited)
8 x 8 Multiplier	Yes	No	No
RAM - Battery Backup mode	Yes - with standby power indicator	Yes	No
16 bit operations — Loads, Stores, Adds, Subtracts, Push/Pulls	Yes	No	Yes (Limited)

MC6801 Positioning Within the Motorola Family

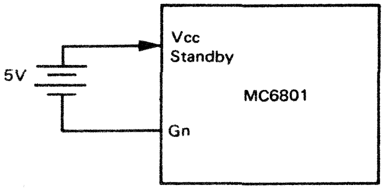
	6801	6800	68B00	68B09
PURPOSE	High Performance Single Chip Computer for Controller Functions and for Intelligent Peripheral Functions	Medium Performance General Purpose CPU	General Purpose High Performance 6800	High Performance CPU Featuring High Level Software
Instruction Cycle Time	2μs Fewer Operation Cycles Per Instruction	2μs	1μs	1μs Fewer Operation Cycles Per Instruction
Program Memory	2K Internal 63K External	65K External	65K External	65K External
Data Memory	128 Bytes Internal 63K External	65K External	65K External	65K External
Minimum Equivalent Chip Count System	1	3 (6846, 6810)	3	3
Specialized I/O	All	All	All	All
On-Chip I/O	29	0	0	0
Interrupts	2 1 SWI	2 1 SWI	2 1 SWI	3 3 SWI

MC6801 applications

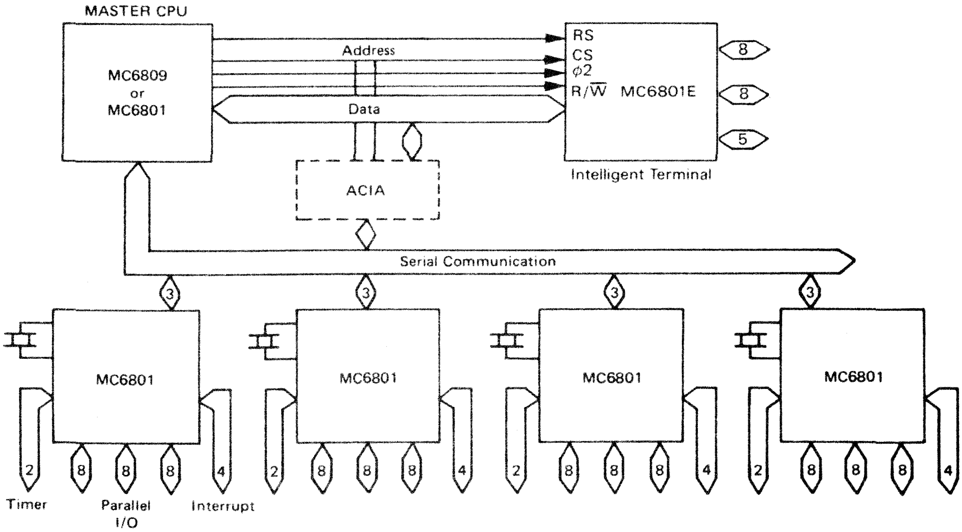
STANDBY POWER

64 Bytes of internal RAM are connected to the V_{CC} standby power pin. There is also a standby bit in the status register which is set when power is first applied to the standby RAM. This bit is reset to "0" if the standby power is lost or removed.

BENEFIT:
Allows the user to test (by his software) the integrity of the data retained in the on-chip RAM.



SERIAL COMMUNICATION LINKED DISTRIBUTED PROCESSING



Serial Communication utilizes the serial port on the MC6801. This type of communication is enhanced with the 6801 because of the WAKEUP (WU) BIT associated with the serial port.

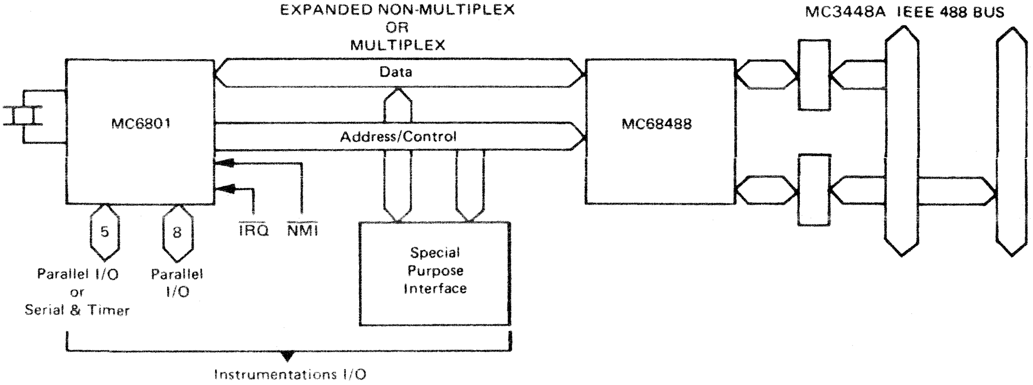
TYPICAL USE:
Each processor would have a unique address which is transmitted as a prefix to the message. Each processor checks this prefix and determines if the message is addressed for its use. If not the processor sets the WU Bit and the serial port ignores the following stream of information on the serial communication line; i.e., the Receiver Buffer Full bit is not set nor is an interrupt activated. This WU Bit is automatically reset when the serial communication line is inactive (high) for 11 bit time slots.

BENEFIT:
The MC6801 is utilizing its processing power servicing the I/O and must only switch to the communication port when checking addresses or actually communicating.

INSTRUMENTATION COMPUTER WITH IEEE 488 BUS

BENEFITS

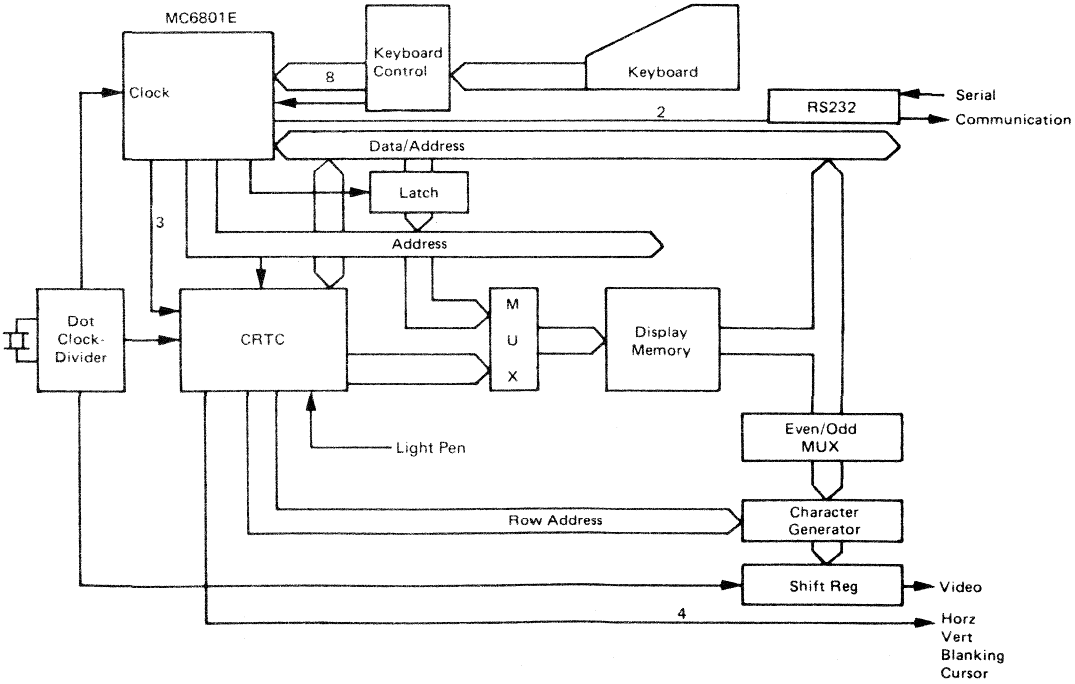
Minimum Part Count
Will Work With All M6800 Family Peripherals
Enhanced Instruction Set



LOW COST TERMINAL

FEATURES

2K ROM	80 x 24 Display Format
1920 Byte Display RAM	29 Total Package Count
256 Type RAM	



MOTOROLA
microsystems

PRELIMINARY INFORMATION

August 21, 1978

NAME: MC6801 Microcomputer Evaluation Module

PART NUMBER: MEX6801EVM

AVAILABILITY: October 1, 1978

FEATURES

- Single Module - EXORciser-bus Compatible
- Incorporates an MC6801 Debug Monitor
- Provides RS-232C Interface
- Has Sockets for Optional ACIA, PTM, and 2K EPROM
- Includes 4K Bytes of Static RAM
- Address Map Established by ROM to Permit Reconfiguration
- Additional Decoding Provided for 8K of Off-Board Memory
- Large Wire Wrap Area
- Two Modes of Operation: Single Chip or Expanded
- Low Cost

DESCRIPTION

The MEX6801EVM Microcomputer Evaluation Module is a completely self-contained microcomputer on a single printed circuit card that provides the user with the means of evaluating the MC6801 microcomputer before the EXORciser-based MEX6801 Module becomes available.

This Microcomputer Evaluation Module is a hardware and software system contained on a single, low cost printed circuit card. This system allows the user to easily evaluate the MC6801 microcomputer. As configured, the MC6801 may be evaluated in either the Single Chip or Expanded mode by attaching an RS-232C-compatible terminal to the serial port of the Module. Thus, the minimum functioning system consists of only the MC6801 and an MC1488 and MC1489.

The Evaluation Module provides the user with the ability to evaluate the MC6801 microcomputer in the Single Chip mode using the debug monitor via the serial I/O port and RS-232C interface. Refer to Table A for a description of the debug commands. A 2.4576 MHz crystal is used to generate the standard baud rates of 150, 600, and 4800 baud. Sufficient space remains within the on-chip RAM for the user to write a small I/O program to work in conjunction with the debug monitor. The debug monitor program (LILBUG) uses a patch table established within RAM for all I/O. Thus, the debug program's I/O routines can be readily modified by the user for the purpose of evaluation. Since the debug monitor uses the timer output and the serial I/O port, these resources are not available to the user in the Single Chip mode. However, by using the Expanded mode, the user has the choice of adding an ACIA or PTM, thereby freeing more of the on-chip resources. The Evaluation Module also provides 4K bytes of static RAM and a 2K byte EPROM socket for the user to develop his programs if desired. In addition, the Expanded mode allocates 8K bytes of off-board program space for further programming flexibility.

A wirewrap area is provided to permit the user to interface other peripheral devices or special interface circuits to the MC6801.

TABLE A. MEX6801EVM Debug Commands

COMMAND	EXPLANATION
L	Load a program from tape.
L <OFFSET>	Load a program from tape with an offset.
V	Verify that a program was properly loaded.
V <OFFSET>	Verify that a program was properly loaded with an offset.
D <ADDR1>,<ADDR2>	Display contents of memory from <ADDR1> to <ADDR2>.
P <ADDR1>,<ADDR2>	Punch/record on tape the contents of memory from <ADDR1> to <ADDR2>.
M <ADDR>	Examine/modify the contents of the specified address location.
<DATA>	Enter one byte of data to replace the value at the current address location.
LF	Display the contents of the next sequential memory location on the next line and enable the contents of this location to be changed (LF = Line Feed).
SP	Display the contents of the next sequential memory location on the same line and enable the contents of this location to be changed (SP = SPace).
,	Increment the memory location pointer, but do not display the address or data. The contents of this memory location may still be changed. This entry permits data to be entered at sequential memory locations without displaying the current address or data.
UA	Display the contents of the previous memory location on the next line and enable the contents of this location to be changed (UA = Up Arrow).
/	Display the current address of the memory location pointer and the contents of that location.
CR	Terminate the memory examine/modify command and accept next command (CR = Carriage Return).
<ADDR>/	Display the contents of the specified address location and enable the contents of this location to be changed.
/	Display the address and contents of the memory location last referenced by the memory examine/modify command.

TABLE A. MEX6801EVM Debug Commands (Contd)

COMMAND	EXPLANATION
O <ADDR1> <ADDR2>	Calculate the relative offset of a branch instruction from ADDR1 to ADDR2.
B	Display all breakpoints.
B -	Delete all breakpoints.
B <ADDR>	Enter a breakpoint at the specified address.
B -<ADDR>	Remove a breakpoint at the specified address.
G <ADDR>	Start program execution at the specified address.
G	Start program execution at the current program counter setting.

R	Display/change the contents of the program registers and counter. Register contents are displayed using the following format:
	P-XXXX X-XXXX A-XX B-XX C-XX S-XX
	P-
	where: P = Program Counter
	X = Index Register
	A = Accumulator A
	B = Accumulator B
	C = Condition Code Register
	S = Stack Pointer
	XXXX = Current 16-bit value
	XX = Current 8-bit value

<DATA>	Replace current register value with new DATA.
SP	Display current register value (unless changed by the <DATA> command). Close the current register and display the next register mneumonic and dash. The R command terminates after examining/changing the stack pointer or whenever any character other than <DATA> or SP is entered (SP = SPace).
.	Trace one instruction.
T <HEX NUMBER>	Trace the number of instructions specified in hexadecimal
C <ADDR>	Call and execute a user routine as a subroutine starting at the specified address. A return address to the debug program is stored in the user's stack. When the user's RTS instruction is executed (at the end of the user's routine), the debug program regains program control and the current contents of the registers are displayed.
C	Same as the C <ADDR> command except that the execution begins from the current address in the program counter.
LO	Set low speed - 30/15 characters per second for on-chip clock.
HI	Set high speed - 120/60 characters per second for on-chip clock.

MOTOROLA microsystems

PRELIMINARY INFORMATION

September 11, 1978

PRODUCT: MC6801 EXORciser Support Module

PART NUMBER: MEX6801

AVAILABILITY: 1st Quarter, 1979

FEATURES

. EXORciser/EXORterm compatible

. User System Evaluator (U.S.E.) capability

. Current Product Compatibility

. Multi-Processor Development

. Extensive Debug Commands

. Real Time Emulation of MC6801

- Single Chip Mode
- Multiplexed Mode
- Non-Multiplexed Mode

MINIMUM SYSTEM REQUIREMENTS

. EXORciser/EXORterm with MPU and debug modules

. EXORdisk II

. 24K Memory

. EXORterm 100 (EXORciser users only)

DESCRIPTION

The MEX6801 is an EXORciser/EXORterm compatible support package for the NMOS MC6801 Microcomputer. It consists of a control module, an extender cable terminated by a 40-pin male DIP plug, and control software that allows evaluation and debug of MC6801 systems, including single chip, expanded multiplexed and non-multiplexed modes.

The MEX6801 support package is capable of operating with any of the existing 6800 MPU modules and therefore allows development of MC6801 software and hardware in either EXORciser I/II or EXORterm configurations. Therefore, the addition of MEX6801 to present Motorola development systems provides the added development of MC6801 systems while maintaining all of its present features. Additionally, the package provides the User System Evaluator (U.S.E.) capability, as well as multiprocessor development capability.

With the 6801 support package installed in an EXORciser/EXORterm, the user can power-up the system, and use the EXORdisk II disk operating system and software to edit and assemble 6801 source code, and debug the resulting object code using familiar EXORciser command syntax. See Table I for a listing and description of MEX6801 commands.

The MEX6801 is completely compatible with the current EXORciser options such as System Analyzer, PROM Programmer, EXORdisk, etc., since the operating programs for them are being executed by the 6800 MPU module in the EXORciser/EXORterm. It is also compatible with static memory and I/O modules.

TABLE I. MEX6801 DEBUG COMMANDS

COMMAND	EXPLANATION
PRNT return	Print memory in both hexadecimal and ASCII format
LOAD return	Load an object tape from the terminal to memory
VERF return	Verify an object tape from the terminal against memory
SRCH return	Search an object tape on the terminal
PNCH return	Punch an object tape on the terminal from memory
;A nn [byte] return	Display and change the A accumulator
;B nn [byte] return	Display and change the B accumulator
;C nn [byte] return	Display and change the condition code register
;E nn [byte] return	Display and change the second level SWI vector
;G	Go (jump) to the target program at its restart address
addr;G	Go (jump) to the target program at the specified address
$H nnnn [addr] return	Enable and change the halt on address or scope sync
;H	Disable the halt on address and scope sync
byte;I	Initialize memory using the specified byte
;K nnnn [value] return	Display and change the terminal null pad value
;L nnnn [addr] return	Display and change the program (location) counter
$M or ;M	Display and change the memory search beginning and ending addresses and search mask
;N	Trace the next instruction
value;N	Trace the next value instructions
;P	Proceed with program execution
value;P	Proceed with program execution from breakpoint; value specifies the number of times the breakpoint location is to be passed before returning control to EXbug and providing a register printout.
;Q nnnn [value] return	Display and change the default debug offset
$R	Display the target program registers
;R	Display the target program registers
;S nnnn [addr] return	Display and change the stack pointer
$T nnnn [addr] return	Enable and change the trace to ending address
;T	Disable the trace to ending address

COMMAND	EXPLANATION
;U	Remove all breakpoints
addr;U	Remove a specified breakpoint
$V	Display the breakpoint addresses
;V	Display the breakpoint addresses
addr;V	Set a breakpoint at the specified address
byte;W	Search memory for the specified byte (word). See the M command.
;X nnnn [addr] return	Display and change the index register
EXIT return	Switch to 6800 MPU, Reset, and enter EXBUG
Control-X	Abort the current command or entry
Control-W	Wait until some other character is entered
addr/ nn cmnd	The memory change function is invoked by entering addr/. Cmnd is one of the following memory change function commands. These commands are accepted as long as EXbug remains in the memory change function.
[byte]LF	Change memory if byte entered, and display the next sequential location
[byte]space	Change memory if byte entered, and display the previous sequential location
[byte]/	Change memory if byte entered, and redisplay the current location
[byte]return	Change memory if byte entered, and exit the memory change function
addr;O	Calculate relative offset from currently open location to the specified location

NOTE: Hexadecimal numbers may be preceded by a minus sign to obtain the two's complement of the value entered.

Values shown in brackets ([ ]) in above explanations indicate optional user inputs.

MOTOROLA
Semiconductors

MC6805

Product Preview

MOS

(N-CHANNEL, SILICON-GATE DEPLETION LOAD)

MICROCOMPUTER

MICROCOMPUTER

The MC6805 is an 8-bit microcomputer containing a CPU, on-chip clock, ROM, RAM, I/O and timer. It is designed for the user who needs an economical low-end microcomputer with the enhanced capabilities of an M6800-based instruction set. The following are some of the hardware and software highlights of the MC6805:

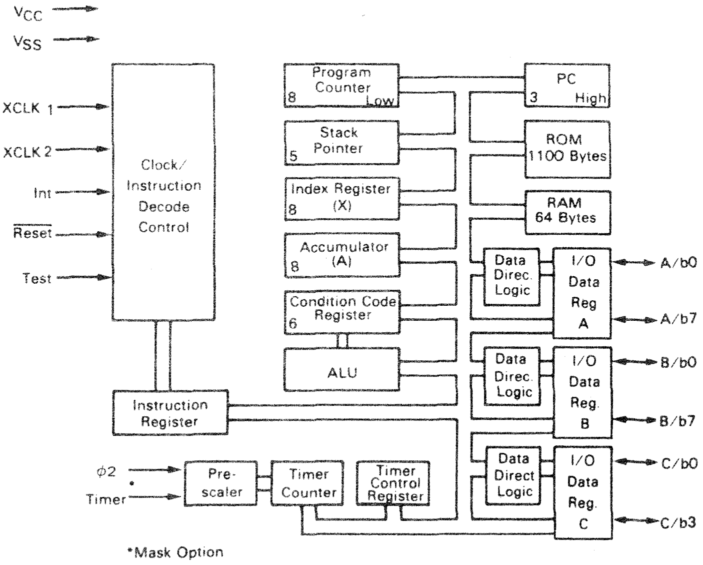
Hardware Features:

- 8-Bit Architecture
- 64 Bytes of RAM
- Memory Mapped I/O
- 1100 Bytes of ROM
- Internal 8-Bit Timer
- Vectored Interrupts
- 20 TTL/CMOS Compatible I/O Lines, 8 Lines LED Compatible
- On Chip Clock Circuit Utilizing External Crystal, Resistor or No
- External Components
- External Interrupt/Sinusoidal Zero Crossing Detect

Software Features:

- Similar to M6800
- Byte Efficient
- Easy to Program
- True Bit Manipulation
- Versatile Interrupt Handling
- Bit Test And Set Instrutions
- Powerful Addressing for Tables
- Full Set of Conditional Branches

FIGURE 1 — CHIP ARCHITECTURE



*Mask Option

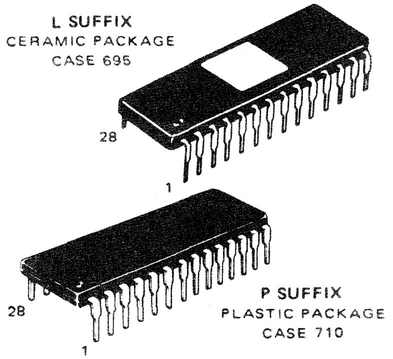
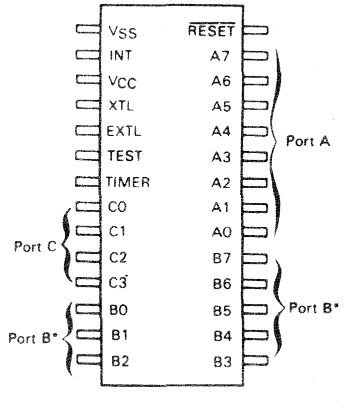


FIGURE 2 — PRELIMINARY PINOUT



*Hi Current Port

This is advance information and specifications are subject to change without notice.

PIN DESCRIPTION

V_{CC} and V_{SS}

These pins are used to supply power, +5 volts, and ground to the chip.

XTAL and EXTAL

Two pins are provided for clocking the microprocessor. They can be connected to a crystal or to an external resistor. Divide by 4 circuitry is internal to the chip, allowing use of the inexpensive 3.58 MHz color TV crystal for 895 KHz operation. Maximum allowed external crystal frequency is 4 MHz.

INT

This is a maskable interrupt pin. Unlike the MC6800's IRQ, the 6805's INT is edge sensitive rather than level sensitive. Also it can be used for zero-crossing detection of an AC input.

$\overline{\text{RESET}}$

Reset is similar to the reset on the MC6802 except that the Stack Pointer is initialized on reset. All I/O lines are set as inputs and both interrupts are masked at power on/Reset. Reset contains power fail detect circuitry.

TIMER INPUT

This is the input to the on-chip timer. There are two choices for the mask option that determines the function of this pin. If the internal $\phi 2$ signal is used for clocking the timer, this input is used for enabling or disabling the timer. If the external clock option is chosen, this input will be used for clocking the timer.

TEST

When asserted, this pin will cause the I/O lines to be converted to multiplexed address and data, allowing the ROM program to be interrogated.

I/O LINES

There are twenty I/O lines on the MC6805 available as two eight-bit ports (Ports A and B), and one four-bit port (Port C). Each of the bits in the ports is programmable as either an input or output depending on the values written into the appropriate data direction register. At power on/Reset, all lines come up as inputs. The sink/source capability of the I/O pins range from TTL to CMOS to LED and Darlington type devices. The state of the output latched data will be correctly read (as input data) regardless of the output loading.

SOFTWARE

The software of the MC6805 has a similarity with the MC6800 in mnemonics and instruction functionality. One of the most powerful additions included on the MC6805 is its bit handling capability. Another benefit of the MC6805 is its ease of programming, especially to an MC6800 user. The MC6805 programming model is shown in Figure 3.

ADDRESSING MODES

The MC6805 has all of the MC6800 addressing modes required for accessing the on-chip memory locations. These include inherent addressing, immediate addressing, direct addressing, extended addressing, indexed addressing (with no offset, one byte offset, and two byte offset), and relative addressing.

BIT MANIPULATION

The bit manipulation of the MC6805 allows any individual bit in RAM or I/O to be set or cleared with single instruction (BSET and BCLR). Also a test and branch on the state (set or clear) of any bit in page 0 (0-255) of the address space is accomplished with a single instruction (BRSET, BRCLR). This eases packing control lines. Bit manipulation also makes possible serial communications as explained later.

TIMER

A software initialized 8-bit timer is included on the MC6805. After initialization, the timer counts down and an interrupt occurs when the count equals zero. It will continue counting after zero so that the software can determine the elapsed time since the interrupt. Two mask options are available to the user: The first selects internal or external source for the timer clock; the second option selects a pre-scaler factor to divide the clock source by 2^n where n = 0, 1, 2-7 (÷128 maximum).

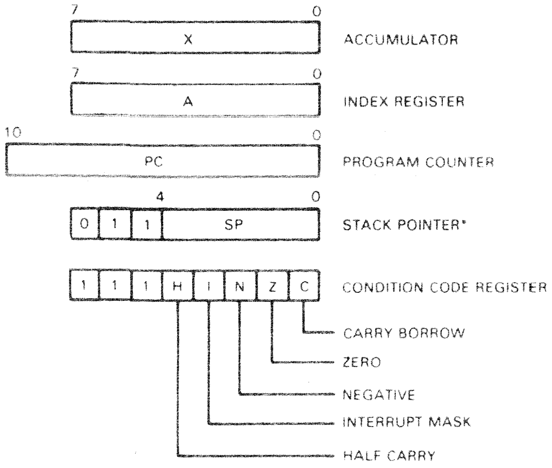
SERIAL COMMUNICATIONS

Due to the bit manipulation capabilities of the MC6805, serial communication is easily accomplished. Ease in serial input is due to the fact that the bit test instructions put the state of the bit tested in the C-bit position. Succeeding rotate instructions place the bits received in series into the accumulator, memory or index register. For serial output, the bit set and clear instructions are utilized.

CPU REGISTERS

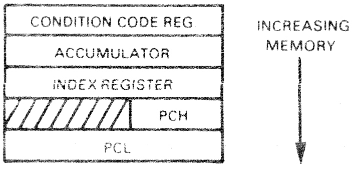
The CPU has one 11-bit register, two 8-bit, and two 5-bit registers (Figure 3).

FIGURE 3 — PROGRAMMING MODEL



*STACK POINTER IS INITIALIZED TO $7F ON RESET

FIGURE 4 — STACK POINTER STACKING ORDER



ACCUMULATOR

The 8-bit Accumulator is a general purpose register and is used for arithmetic calculations and data manipulations.

INDEX REGISTER

The index register is an 8-bit register used to store data or a memory address for the indexed mode of memory addressing. Read-modify-write instructions are available on the index register, allowing it to be used as a back-up accumulator.

PROGRAM COUNTER

The program counter is an 11-bit register that points to the current program address.

STACK POINTER

The stack pointer is a 5-bit register that contains the address of the next available location. On reset, the stack pointer is initialized to $7F, the highest location of RAM

CONDITION CODE REGISTER

The condition code register indicates the results of an Arithmetic Logic Unit operation: Half carry from bit 3 (H), interrupt mask bit (I), Negative (N), Zero (Z), Carry/Borrow (C).

MC6805 INSTRUCTION SET

Included here is a complete list of instructions available on the MC6805. They are grouped according to register/memory instructions, read/modify/write instructions, branch instructions, bit manipulation instructions, and control instructions.

REGISTER/MEMORY INSTRUCTIONS

Instruction Lists

ADC	Add Memory and Carry to Accumulator
ADD	Add Memory to Accumulator
AND	And Memory to Accumulator
BIT	Bit Test Memory with Accumulator
CMP	Compare Accumulator with Memory
CPX	Compare Index Register with Memory
EOR	Exclusive or Memory with Accumulator
JMP	Jump Absolute
JSR	Jump to Subroutine
LDA	Load Accumulator from Memory
LDX	Load Index Register from Memory
ORA	Or Memory with Accumulator
SBC	Subtract Memory and Borrow from Accumulator
STA	Store Accumulator in Memory
STX	Store Index Register in Memory
SUB	Subtract Memory from Accumulator

READ/MODIFY/WRITE INSTRUCTIONS

ASL	Arithmetic Shift Left (Same as LSL)
ASR	Arithmetic Shift Right
CLR	Clear
COM	Complement
DEC	Decrement
INC	Increment
LSL	Logical Shift Left (Same as ASL)
LSR	Logical Shift Right
NEG	Negate
ROL	Rotate Left thru Carry
ROR	Rotate Right thru Carry
TST	Test for Negative or Zero

BIT MANIPULATION INSTRUCTIONS

BCLR	Bit Clear Bit n in Memory
BRCLR	Branch if Bit n is Clear in Memory
BRSET	Branch if Bit n is Set in Memory
BSET	Bit Set Bit n in Memory

BRANCH INSTRUCTIONS

BRA	Branch Always
BCC	Branch if Carry Clear (Same as BHS)
BCS	Branch if Carry Set (Same as BLO)
BEQ	Branch if Equal
BHCC	Branch if Half Carry Clear
BHCS	Branch if Half Carry Set
BHI	Branch if Higher
BHS	Branch if Higher or Same (Same as BCC)
BIH	Branch if Interrupt Line is HIGH
BIL	Branch if Interrupt Line is Low
BLO	Branch if Lower (Same as BCS)
BLS	Branch if Lower or Same
BMC	Branch if Interrupt Mask is Clear
BMI	Branch if Minus
BMS	Branch if Interrupt Mask is Set
BNE	Branch if Not Equal
BPL	Branch if Plus
BSR	Branch to Subroutine

CONTROL INSTRUCTIONS

CLC	Clear Carry Bit
CLI	Clear Interrupt Mask Bit
NOP	No-Operation
RSP	Reset Stack Pointer
RTI	Return from Interrupt
RTS	Return from Subroutine
SEC	Set Carry Bit
SEI	Set Interrupt Mask Bit
SWI	Software Interrupt
TAX	Transfer Accumulator to Index Register
TXA	Transfer Index Register to Accumulator

Motorola reserves the right to make changes to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein, neither does it convey any license under its patent rights nor the rights of others.

MOTOROLA Semiconductor Products Inc.

71-C79/5.0 Printed in Switzerland

MOTOROLA Semiconductors

Product Preview

MC6809(E)
(1.0 MHz)

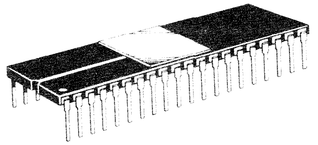
MC68A09(E)
(1.5 MHz)

MC68B09(E)
(2.0 MHz)

MOS

(N-CHANNEL, SILICON-GATE, DEPLETION LOAD)

HIGH-PERFORMANCE MICROPROCESSOR



L SUFFIX
CERAMIC PACKAGE
CASE 715

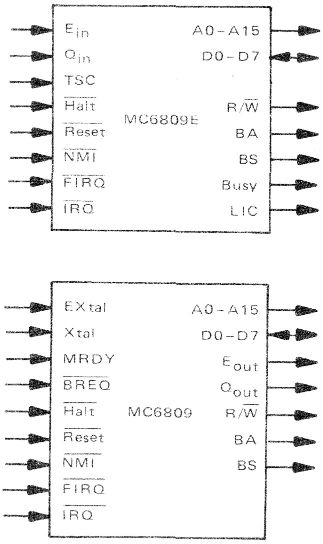
NOT SHOWN:
P SUFFIX
PLASTIC PACKAGE
CASE 711

HIGH-PERFORMANCE MICROPROCESSOR

- MC6800 COMPATIBLE
  - Hardware – Interfaces With All M6800 Peripherals
  - Software – Upward Compatible Instruction Set and Addressing Modes
- HARDWARE FEATURES
  - On-Chip Oscillator (MC6809) 4 × fo Clock
  - Optional ÷1 External Clock Inputs (MC6809E)
  - MRDY Input Extends Data Access Times for Use With Slow Memory
  - $\overline{\text{BREQ}}$/TSC Allows Quick Access to Bus for DMA and Memory Refresh
  - Last Instruction Cycle Output for Identification of Opcode Fetch (MC6809E)
  - Fast Interrupt Request Input Stacks Only Program Counter and Condition Code
  - Interrupt Acknowledge Output Allows Vectoring by Device
  - Busy Output Eases Multiprocessor Design (MC6809E)
- ARCHITECTURAL FEATURES
  - Two 8-Bit Accumulators Can Be Concatenated to Form One 16-Bit Accumulator
  - Two 16-Bit Index Registers
  - Two 16-Bit Indexable Stack Pointers
  - Direct Page Register Allows Direct Addressing Throughout Memory Space
- INSTRUCTION SET
  - Extended Range Branches
  - 16-Bit Arithmetic
  - Push/Pull Any Register or Set of Registers To/From Either Stack
  - 8 × 8 Unsigned Multiply
  - Transfer/Exchange Any Two Registers of Equal Size
  - Enhanced Pointer Register Manipulation
- ADDRESSING MODES
  - All MC6800 Modes, Plus PC Relative, Extended Indirect, Indexed Indirect, and PC Relative Indirect
  - Direct Addressing Available for All Memory Access Instructions
  - Index Mode Options Include Accumulator or Up to 16-Bit Constant Offset, and Auto-Increment/Decrement (by 1 or 2) With Any of the Four Pointer Registers

This is advance information and specifications are subject to change without notice.

FIGURE 1 – BLOCK DIAGRAMS



M6809 FEATURES

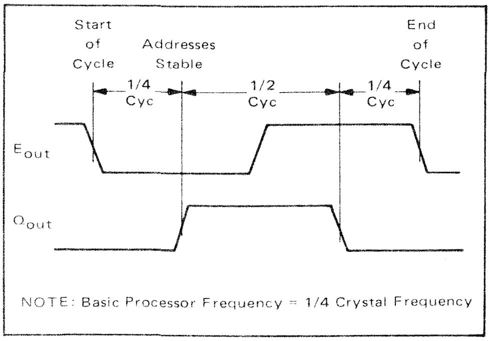
The MC6809 is an advanced processor within the M6800 family offering greater through-put, improved byte-efficiency, and increased adaptability to various software disciplines. These include position independence, reentrancy, recursion, block structuring, and high level language generation. It is compatible with all M6800 peripherals, and is software compatible with the MC6800 at source code level.

This product preview stresses the improvements inherent in the MC6809. The basic operation of the MPU is similar to the MC6800.

Hardware Features — Internal Clock MC6809

For internal clock use, an external crystal is connected between Extal and Xtal. A synchronization signal is available at the E_{out} terminal to be used as a system clock. This signal operates at the basic processor frequency, and is normally connected to the Enable (ϕ2) inputs of M6800 family peripherals. A Quadrature output (Q_{out}) provides additional system timing by signifying that addresses and data are stable as shown in Figure 2.

FIGURE 2 — MC6809 TIMING SIGNALS



NOTE: Basic Processor Frequency = 1/4 Crystal Frequency

A Memory Ready (MRDY) input allows extension of data access times for use with slow memories. A logic zero at this input (when E_{out} goes high) causes E_{out} to remain high until MRDY returns high. Stretching is always an integral number of quarter bus cycles, and is limited to a maximum of ten microseconds. The negative transition of Q_{out} is unaffected by MRDY, but further positive transitions of Q_{out} are inhibited for the duration of the clock stretch.

A Bus Request ($\overline{\text{BREQ}}$) input allows fast access to the bus for DMA or Memory Refresh. This is a request to temporarily suspend MPU operation and take the MPU off the MOS bus. The BA line will immediately go high (as a result of the trailing edge of E), signifying a Bus Available condition. One-half cycle later, the user may place the DMA device on the MPU buses. This will eliminate bus contentions into DMA.

Hardware Features — External Clock MC6809E

The External Clock mode of the MC6809 is particularly useful when it is desired to synchronize the processor to an externally generated signal. The external clock generator provides an output only at the basic MPU frequency, since the internal frequency dividers of the MC6809 are not used in the external clock mode. E_{in} and Q_{in} signals are required with phasing as shown for E_{out} and Q_{out}, respectively.

A Three-State Control (TSC) input replaces the $\overline{\text{BREQ}}$ input of the MC6809, and serves to place the Address *and R/$\overline{\text{W}}$ line in the high-impedance state for DMA or* Memory Refresh. (The Data Bus is in the high-impedance state when E_{in} and Q_{in} are both low.) The E_{out} and Q_{out} terminals are replaced by two status outputs (LIC and Busy).

A last Instruction Cycle output (LIC) is activated during the last cycle of any instruction. The first low cyle after LIC is high signifies that this processor cycle will be an opcode fetch.

A Processor Busy signal (BUSY) facilitates multiprocessor applications. This signal is asserted during MPU Read-Modify-Write instructions, allowing the designer to insure that flags being modified by one processor are not accessed by another simultaneously. The signal is also asserted during execution of double byte instructions and when using the Indirect Addressing modes.

Hardware Features — Either Clock Option

A Fast Interrupt Request ($\overline{\text{FIRQ}}$) is added to the $\overline{\text{IRQ}}$ and $\overline{\text{NMI}}$ inputs available on the MC6800. When a logic zero is recognized at this input, the MC6809 places only the Program Counter and Condition Code Register on the stack prior to accessing the $\overline{\text{FIRQ}}$ vector location to obtain the starting location of the $\overline{\text{FIRQ}}$ service routine. Either an $\overline{\text{IRQ}}$ or Non-Maskable Interrupt stacks the contents of all registers (same as MC6800) prior to vectoring. The three interrupt inputs have separate vector locations, and are prioritized ($\overline{\text{IRQ}}$ lowest, $\overline{\text{FIRQ}}$ next, $\overline{\text{NMI}}$ highest).

An Interrupt Acknowledge function (IACK) indicates that a vector is being fetched as a result of a $\overline{\text{Reset}}$, a Software Interrupt (SWI, SWI2, or SWI3), or recognition of $\overline{\text{NMI}}$, $\overline{\text{FIRQ}}$, or $\overline{\text{IRQ}}$. This function allows the program counter to be loaded according to the interrupting device, thus providing full vectored interrupt handling. IACK is denoted by a logic zero at the Bus Available (BA) output in conjunction with a logic one at the Bus Status (BS) output.

A low level on the Halt input causes the MPU to halt at the end of the present instruction. It will remain halted indefinitely without loss of data, until the Halt line is driven high. When the MPU is halted, the BA and BS outputs are driven high to acknowledge the halt, and the Address Bus and Data Bus drivers are made high-impedance. While halted, the MPU cannot respond to interrupt requests, nor can it be released from reset.

The MC6809 has the capability of entering an idle state under program control via SYNC and CWAI instructions. SYNC Acknowledge can be detected via BA and BS output states. These two status signals are defined in Table 1.

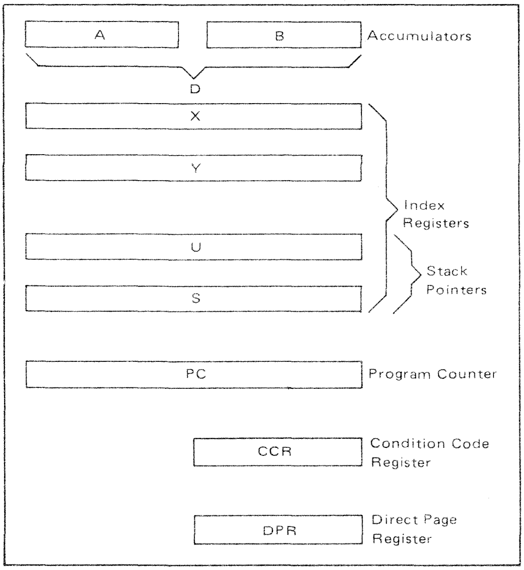
TABLE 1 – STATUS SIGNALS

BA	BS	Function
Ø	Ø	Normal Operation
Ø	1	Interrupt Acknowledge
1	Ø	Sync Acknowledge
1	1	Bus Grant or Halt Acknowledge

PROGRAMMING MODEL

The MC6809 adds three registers to the complement available in the MC6800. These are the Direct Page Register, the User Stack Pointer, and a second (Y) Index Register (shown in Figure 3). In addition to increasing the number of registers, the MC6809 features much greater flexibility of register usage. For example, indexed addressing is available using a stack pointer (U or S) as the base register in addition to X or Y. Conversely, the Automatic Increment/Decrement option of indexed instructions allows X and Y to be used as stack pointers if desired.

FIGURE 3 – PROGRAMMING MODEL



The enhanced instruction set of the MC6809 also increases register uses. Exchanges, as well as transfers, between any two similar-width registers are allowed. Two's complement addition of the A, B, or D register (or an immediate value) to X, Y, U, or S registers can be implemented via the Load Effective Address instruction. Any register—or set of registers—may be pushed to (pulled from) either stack with a single instruction.

Accumulators

The A and B registers are general-purpose accumulators used for arithmetic calculators and data manipulation. In general, the two registers are identical, although some special purpose instructions apply to one register alone.

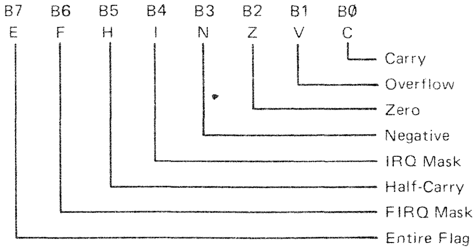
Certain instructions concatenate the A and B registers to form a single 16-bit accumultor. This is referred to as the D register, and is formed with the A register as the Most Significant Byte.

Direct Page Register

The Direct Page Register of the MC6809 serves to enhance the Direct Addressing Mode. The contents of this register appear at the higher order address outputs (A8–A15) during direct addressing instruction execution. This allows the Direct Page to be located at any place in memory under program control. All bits of this register are cleared during Processor Reset. This insures 6800 compatibility.

Condition Code Register

The Condition Code Register is the storage area for processor flags. Bits Ø through 5 are identical to the MC6800. CCR–Bits 6 and 7 are used for FIRQ handling.



Specifically, the E-bit controls the Return from Interrupt (RTI) instruction to insure that the proper number of registers are pulled from the Hardware Stack.

Index Registers

The Index Registers are used in indexed mode addressing. They provide a 16-bit address to be added to an optional offset (of up to 16 bits) to form the effective address of the instruction. The X and Y registers are essentially equivalent in usage and support the same instructions. Either index register is capable of performing the same function as the MC6800 index register, but can also support many additional modes of operation.

Stack Pointers

The Hardware Stack Pı inter (S) is used by the processor to automatically store machine states during subroutines and interrupts in a manner similar to that of the MC6800. A User Stack Pointer (U) is also provided to be controlled exclusively by the program—thus allowing arguments to be passed to and from subroutines with ease.

Both U and S have the same indexed-mode addressing capabilities as the X and Y registers. This allows the MC6809 to be used efficiently as a stack processor—greatly enhancing its ability to support higher level languages. To facilitate use of the stack pointers in indexing mode addressing, the registers always point to the last byte placed on the stack. (The MC6800 stack pointer points to the location in which the next byte will be stored.)

Program Counter

The PC is used by the processor to point to the next instruction to be executed by the processor. Program Counter Relative addressing is allowed on the MC6809, effectively allowing the PC register to be used as an index register. This allows Position Independent Code to be written with greater ease than with other 8-bit processors.

ADDRESSING MODES

The MC6809 retains the Addressing modes available with the MC6800, and adds long relative branches, 16 variations of indexed addressing, program counter relative and extended indirect addressing.

The enhancements to indexed addressing include auto (post) increment, auto (pre) decrement, indexing with 0-, 5-, 8-, or 16-bit two's complement offsets, and indexing with an accumulator as an offset. Also, any of these modes may have one additional level of indirection applied. Any of the four index registers (X, Y, U, or S) may be used as the base register for the indexed addressing modes.

Inherent, Accumulator, Immediate, Direct, Extended

These modes are effectively the same as those available with the MC6800. (The Direct Mode utilizes the DPR to form the most significant address byte as explained previously.)

Indexed

The Indexed Mode of addressing has several options which are selected by the byte following the opcode (postbyte). Bits 5 and 6 of this postbyte are always used to select the pointer register (X, Y, U, or S). Other options are:

Zero Offset — This option allows selection of Auto Increment/Decrement by one or two bits. It is a minimum two-byte instruction (opcode + postbyte).

Constant Offset, ±4 Bits — This option uses Bit 4 of the postbyte as a sign bit and Bits Ø through 3 as a constant offset. It is a minimum two-byte instruction.

Constant Offset, ±7 Bits — This option designates the byte after the postbyte as a two's complement offset. It is a minimum three-byte instruction (opcode + postbyte + offset).

Constant Offset, ±15 Bits — This option designates the two bytes following the postbyte as a two's complement offset. it is a minimum four-byte instruction (opcode + postbyte + two-byte offset).

Accumulator Offset — This option designates the A, B, or D register as a two's complement offset. It is a minimum two-byte instruction.

In all cases, the offset is temporarily added to the contents of the selected pointer register to form an effective address.

Indexed Indirect

Except for the ±4-Bit Constant Offset and the Auto Increment/Decrement by one, all indexed addressing modes may be used with an additional level of indirection. Thus, the address formed by adding the offset to the selected pointer register designates a location containing the effective address of the operand data. Bit 4 of the postbyte is used to select the Indexed Indirect mode. (Note that this bit is used as a sign bit in the ±4-bit constant offset mode.) The number of bytes for a given instruction is the same for either Indexed or Indexed Indirect addressing modes.

Relative

Relative addressing involves adding a signed constant to the contents of the program counter. When used in conjunction with a Branch Instruction, this sum becomes the new Program Counter content if the branch is taken. (If the branch is not taken, the PC advances to the next instruction.)

Relative addressing differs from that contained in the MC6800 due to two important additions. The first of these is that the offset (signed constant) can be either ±7 bits or ±15 bits in length. This allows the program to branch to any location in the memory field.

The second important addition to the Relative mode is that it no longer is limited to branch instructions. An Effective Address—which retains the position-independent nature of relative addressing—may be formed by adding a ±7-bit or ±15-bit offset to the program counter. This is, in effect, an Indexed Addressing mode with one of two specific postbytes. (The optional postbytes allow selection of ±7 or ±15-bit offset.) Examples of its use would be:

```
2015  LDA  -$1FDA,PCR        2018  LDA   $413B,PCR

2015  [A6]  OPCODE           2018  [A6]  OPCODE
2016  [8C]  POSTBYTE         2019  [8D]  POSTBYTE
2017  [C2]  OFFSET           201A  [21]  OFFSET (MSB)
2018  [  ]  NEXT INST        201B  [1F]  OFFSET (LSB)
                             201C  [  ]  NEXT INST

1FDA  [  ]  DATA             413B  [  ]  DATA
```

Note that the offset is added to the new value of the program counter, i.e., the location of the next instruction.

Relative Indirect

This addressing mode is, in effect, Indexed Indirect with the Program Counter used as an index register. One or two bytes (optional) following the postbyte are used to provide a ±7 byte or ±15 bit offset. This signed number is added to the contents of the program counter, forming a pointer to consecutive locations in memory which contain the new effective address. Examples are:

```
2015  LDA  [$1FDA,PCR]          2018  LDA   [$413B,PCR]

2015   A6   OPCODE              2018   A6   OPCODE
2016   9C   POSTBYTE            2019   9D   POSTBYTE
2017   C2   OFFSET              201A   21   OFFSET (MSB)
2018   □    NEXT INST           201B   1F   OFFSET
                                201C   □    NEXT INST
1FDA   01 } NEW                 413B   03 } NEW
1FDB   00 } EA                  413C   00 } EA

0100   □    DATA                0300   □    DATA
```

Extended Indirect

This addressing mode is actually another option of Indexed Indirect Addressing. In this case, the two bytes following the postbyte are used as a pointer to consecutive locations in memory which contain the new effective address. An example is:

```
201C  LDA   [$C200]

201C   A6    OPCODE
201D   9F    POSTBYTE
201E   C2 }
201F   00 }  POINTER
2020   □     NEXT INST

C200   00    NEW
C201   80    EA

0080         DATA
```

Absolute Indirect

This mode is exclusively used for Restart and Interrupt vectoring. Servicing of these conditions involves fetching the contents of an exact location in memory to be loaded into the Program Counter.

Instruction Set

A complete listing of the Executable instructions is contained in Tables 2 through 6. Some of the more unique instructions include Load Effective Address, Synchronization with Interrupt, and Exchange Registers. These, along with others not available on the MC6800, are detailed in the following paragraphs.

Load Effective Address — Besides its obvious use, this instruction represents a convenient means of modifying any of the pointer (X, Y, S, or U) registers. The processor forms an Effective Address as dictated by the addressing mode, then loads this value into the designated register rather than outputting the data on the address lines. As an example, LEAX A,X will add the signed number contained in the A register to the contents of the X register, then place this result into the X register.

Synchronize with Interrupt — This instruction causes processing to discontinue until an Interrupt Input ($\overline{NMI}$, $\overline{IRQ}$, or $\overline{FIRQ}$) is activated. When an interrupt occurs, the processor services the interrupt normally if the associated mask is clear. If the interrupt mask is set when the interrupt occurs, the processor exits the Sync mode by continuing to the next instruction. The Bus Available output is activated during the Sync mode, but BS remains low. The instruction is useful for synchronizing the program with a peripheral and for performing DMA under program control.

Exchange and Transfer Registers — Both of these instructions utilize an immediate byte to define the source and destination registers. The only restriction is that both source and destination must be similarly sized registers.

Push/Pull Register(s) — These instructions also use an immediate byte to designate whether the register assigned to a particular bit is to be affected. Thus, a Push instruction followed by a byte containing a "one" in bit 7 causes the Program Counter to be pushed onto a stack. One to eight registers can be pushed or pulled with a single instruction.

Sign Extend — This instruction causes all bits in the A register to take on the value of the Most Significant Bit of the B register.

CWAI — This instruction is similar to the Wait for Interrupt used with the MC6800, but includes an immediate byte to clear condition codes if desired. The CWAI instruction can be used with any of the three interrupt lines, even though CWAI stacks all registers (except S). After stacking is complete, the processor idles until an interrupt occurs.

Those familiar with the MC6800 will note that some instructions are missing from the complement available with the MC6800. Provisions have been made to perform such operations in alternate ways when required. An example might be Decrement X. This will not often be needed with the MC6809 due to the Auto-Decrement option with Indexed Addressing. If needed, however, the operation can be accomplished with an LEAX −1,X instruction. Likewise, the MC6800 instructions to Clear/Set various condition codes are replaced with ANDCC/ORCC. In this manner, the MC6809 uses fewer instruction mnemonics (59 versus 72) than the MC6800, yet is fully software compatible, as well as being considerably more powerful.

TABLE 2 – 8-BIT ACCUMULATOR AND MEMORY INSTRUCTIONS

		Addressing Modes								
Mnemonic(s)	Operation	Implied	Immediate	Direct	Extended	Extended Indirect	Indexed	Indexed Indirect	Relative	Relative Indirect
ADCA, ADCB	Add memory to accumulator with carry	–	X	X	X	X	X	X	X	X
ADDA, ADDB	Add memory to accumulator	–	X	X	X	X	X	X	X	X
ANDA, ANDB	And memory with accumulator	–	X	X	X	X	X	X	X	X
ASL	Arithmetic shift left memory location	–	–	X	X	X	X	X	X	X
ASLA, ASLB	Arithmetic shift left accumulator	X	–	–	–	–	–	–	–	–
ASR	Arithmetic shift right memory location	–	–	X	X	X	X	X	X	X
ASRA, ASRB	Arithmetic shift right accumulator	X	–	–	–	–	–	–	–	–
BITA, BITB	Bit test memory with accumulator	–	X	X	X	X	X	X	X	X
CLR	Clear memory location	–	–	X	X	X	X	X	X	X
CLRA, CLRB	Clear accumulator	X	–	–	–	–	–	–	–	–
CMPA, CMPB	Compare memory with accumulator	–	X	X	X	X	X	X	X	X
COM	Complement memory location	–	–	X	X	X	X	X	X	X
COMA, COMB	Complement accumulator	X	–	–	–	–	–	–	–	–
DAA	Decimal adjust A-accumulator	X	–	–	–	–	–	–	–	–
DEC	Decrement memory location	–	–	X	X	X	X	X	X	X
DECA, DECB	Decrement accumulator	X	–	–	–	–	–	–	–	–
EORA, EORB	Exclusive or memory with accumulator	–	X	X	X	X	X	X	X	X
EXG R1, R2	Exchange R1 with R2 (R1, R2 = A, B, CC, DP)	X	–	–	–	–	–	–	–	–
INC	Increment memory location	–	–	X	X	X	X	X	X	X
INCA, INCB	Increment accumulator	X	–	–	–	–	–	–	–	–
LDA, LDB	Load accumulator from memory	–	X	X	X	X	X	X	X	X
LSL	Logical shift left memory location	–	–	X	X	X	X	X	X	X
LSLA, LSLB	Logical shift left accumulator	X	–	–	–	–	–	–	–	–
LSR	Logical shift right memory location	–	–	X	X	X	X	X	X	X
LSRA, LSRB	Logical shift right accumulator	X	–	–	–	–	–	–	–	–
MUL	Unsigned multiply (AXB → D)	X	–	–	–	–	–	–	–	–
NEG	Negate memory location	–	–	X	X	X	X	X	X	X
NEGA, NEGB	Negate accumulator	X	–	–	–	–	–	–	–	–
ORA, ORB	Or memory with accumulator	–	X	X	X	X	X	X	X	X
ROL	Rotate memory location left	–	–	X	X	X	X	X	X	X
ROLA, ROLB	Rotate accumulator left	X	–	–	–	–	–	–	–	–
ROR	Rotate memory location right	–	–	X	X	X	X	X	X	X
RORA, RORB	Rotate accumulator right	X	–	–	–	–	–	–	–	–
SBCA, SBCB	Subtract memory from accumulator with borrow	–	X	X	X	X	X	X	X	X
STA, STB	Store accumulator to memory	–	–	X	X	X	X	X	X	X
SUBA, SUBB	Subtract memory from accumulator	–	X	X	X	X	X	X	X	X
TST	Test memory location	–	–	X	X	X	X	X	X	X
TSTA, TSTB	Test accumulator	X	–	–	–	–	–	–	–	–
TFR, R1, R2	Transfer R1 to R2 (R1, R2 = A, B, CC, DP)	X	–	–	–	–	–	–	–	–

NOTE: A and B may be pushed to (pulled from) either stack with PSHS, PSHU (PULS, PULU) instructions. See Table 3.

TABLE 3 — 16-BIT ACCUMULATOR AND MEMORY INSTRUCTIONS

		Addressing Modes								
Mnemonic(s)	Operation	Implied	Immediate	Direct	Extended	Extended Indirect	Indexed	Indexed Indirect	Relative	Relative Indirect
ADDD	Add memory to D accumulator	—	X	X	X	X	X	X	X	X
CMPD	Compare memory with D accumulator	—	X	X	X	X	X	X	X	X
EXG D, R	Exchange D with X, Y, S, U, or PC	X	—	—	—	—	—	—	—	—
LDD	Load D accumulator from memory	—	X	X	X	X	X	X	X	X
SEX	Sign Extend	X	—	—	—	—	—	—	—	—
STD	Store D accumulator to memory	—	—	X	X	X	X	X	X	X
SUBD	Subract memory from D accumulator	—	X	X	X	X	X	X	X	X
TFR D, R	Transfer D to X, Y, S, U, or PC	X	—	—	—	—	—	—	—	—
TFR R, D	Transfer X, Y, S, U, or PC to D	X	—	—	—	—	—	—	—	—

TABLE 4 — INDEX REGISTER/STACK POINTER INSTRUCTIONS

		Addressing Modes								
Mnemonic(s)	Operation	Implied	Immediate	Direct	Extended	Extended Indirect	Indexed	Indexed Indirect	Relative	Relative Indirect
CMPS, CMPU	Compare memory with stack pointer	—	X	X	X	X	X	X	X	X
CMPX, CMPY	Compare memory with index register	—	X	X	X	X	X	X	X	X
EXG R1, R2	Exchange D, X, Y, S, U, or PC with D, X, Y, S, U, or PC	X	—	—	—	—	—	—	—	—
LEAS, LEAU	Load effective address into stack pointer	—	—	—	—	X	X	X	X	X
LEAX, LEAY	Load effective address into index register	—	—	—	—	X	X	X	X	X
LDS, LDU	Load stack pointer from memory	—	X	X	X	X	X	X	X	X
LDX, LDY	Load index register from memory	—	X	X	X	X	X	X	X	X
PSHS	Push any register(s) onto hardware stack (except S)	X	—	—	—	—	—	—	—	—
PSHU	Push any register(s) onto user stack (except U)	X	—	—	—	—	—	—	—	—
PULS	Pull any register(s) from hardware stack (except S)	X	—	—	—	—	—	—	—	—
PULU	Pull any register(s) from hardware stack (except U)	X	—	—	—	—	—	—	—	—
STS, STU	Store stack pointer to memory	—	—	X	X	X	X	X	X	X
STX, STY	Store index register to memory	—	—	X	X	X	X	X	X	X
TFR R1, R2	Transfer D, X, Y, S, U, or PC to D, X, Y, S, U, or PC	X	—	—	—	—	—	—	—	—
ABX	Add B-accumulator to X (unsigned)	X	—	—	—	—	—	—	—	—

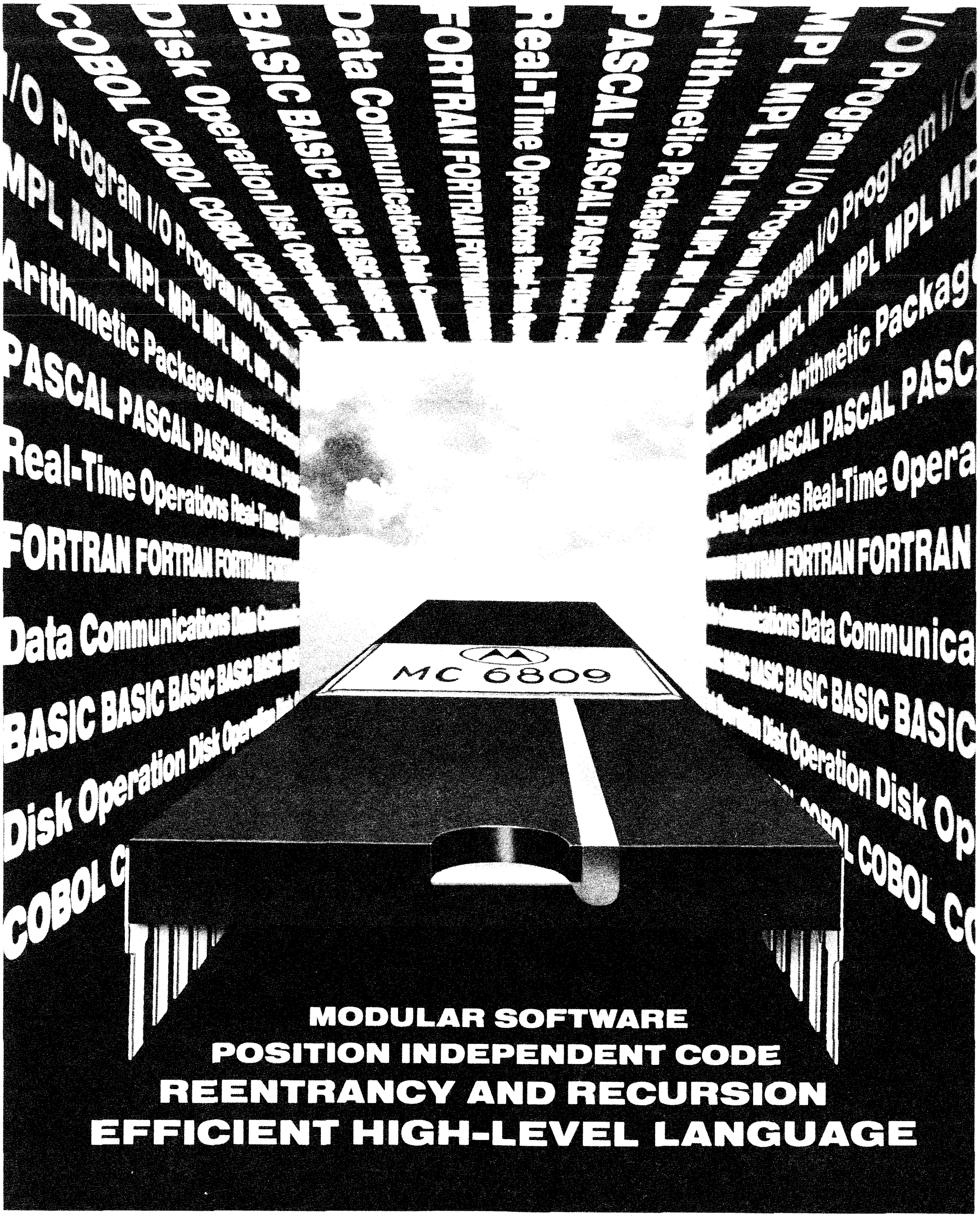
TABLE 5 – BRANCH INSTRUCTIONS

		Addressing Modes								
Mnemonic(s)	Operation	Implied	Immediate	Direct	Extended	Extended Indirect	Indexed	Indexed Indirect	Relative	Relative Indirect
BCC, LBCC	Branch if carry clear	—	—	—	—	—	—	—	X	—
BCS, LBCS	Branch if carry set	—	—	—	—	—	—	—	X	—
BEQ, LBEQ	Branch if equal	—	—	—	—	—	—	—	X	—
BGE, LBGE	*Branch if greater than or equal (signed)*	—	—	—	—	—	—	—	X	—
BGT, LBGT	Branch if greater (signed)	—	—	—	—	—	—	—	X	—
BHI, LBHI	Branch if higher (unsigned)	—	—	—	—	—	—	—	X	—
BHS, LBHS	Branch if higher or same (unsigned)	—	—	—	—	—	—	—	X	—
BLE, LBLE	Branch if less than or equal (signed)	—	—	—	—	—	—	—	X	—
BLO, LBLO	Branch if lower (unsigned)	—	—	—	—	—	—	—	X	—
BLS, LBLS	Branch if lower or same (unsigned)	—	—	—	—	—	—	—	X	—
BLT, LBLT	Branch if less than (signed)	—	—	—	—	—	—	—	X	—
BMI, LBMI	Branch if minus	—	—	—	—	—	—	—	X	—
BNE, LBNE	Branch if not equal	—	—	—	—	—	—	—	X	—
BPL, LBPL	Branch if plus	—	—	—	—	—	—	—	X	—
BRA, LBRA	Branch always	—	—	—	—	—	—	—	X	—
BRN, LBRN	Branch never (3, 5 Cycle NOP)	—	—	—	—	—	—	—	X	—
BSR, LBSR	Branch to subroutine	—	—	—	—	—	—	—	X	—
BVC, LBVC	Branch if overflow clear	—	—	—	—	—	—	—	X	—
BVS, LBVS	Branch if overflow set	—	—	—	—	—	—	—	X	—

TABLE 6 – MISCELLANEOUS INSTRUCTIONS

		Addressing Modes								
Mnemonic(s)	Operation	Implied	Immediate	Direct	Extended	Extended Indirect	Indexed	Indexed Indirect	Relative	Relative Indirect
ANDCC	AND condition code register	—	X	—	—	—	—	—	—	—
CWAI	AND condition code register, then wait for interrupt	—	X	—	—	—	—	—	—	—
NOP	No operation	X	—	—	—	—	—	—	—	—
ORCC	OR condition code register	—	X	—	—	—	—	—	—	—
JMP	Jump	—	—	X	X	X	X	X	X	X
JSR	Jump to subroutine	—	—	X	X	X	X	X	X	X
RTI	Return from interrupt	X	—	—	—	—	—	—	—	—
RTS	Return from subroutine	X	—	—	—	—	—	—	—	—
SWI, SWI2, SWI3	Software interrupt (absolute indirect)	X	—	—	—	—	—	—	—	—
SYNC	Synchronize with interrupt line	X	—	—	—	—	—	—	—	—

Motorola reserves the right to make changes to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others.



MODULAR SOFTWARE
POSITION INDEPENDENT CODE
REENTRANCY AND RECURSION
EFFICIENT HIGH-LEVEL LANGUAGE

MC6809 MICROPROCESSOR FACT BOOK

MOTOROLA
Semiconductor Products Inc.

introducing

the Motorola MC6809

An 8-Bit Microprocessor (MPU) designed with particular attention to providing an enhanced software and higher performance central processing unit.

2.5 to 5 Times the Performance of the MC6800

MC6800 Software Compatible (Source Code)

Architectural Improvements

- Additional 16-Bit Registers
- Expanded Addressing Mode Common to All Index (3) Registers
- 16-Bit Operations
- 8 X 8 Multiplier

Software Improvements Support

- Tailored to Higher-Level-Language
- Position Independence
- Structured, Highly Subroutined Code
- Multi-task and Multi-processor Organization
- Stack-Oriented Compiler Instructions
- Reentrancy and Recursion

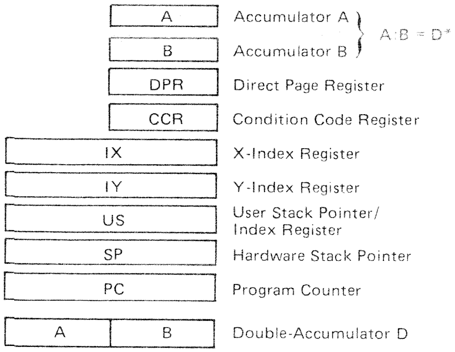
Hardware Improvements

- On-Chip Clock (MC6809) or Off-Chip Clock (MC6809E)
- Additional Control (Fast IRQ, Ready, Busy, Last Instruction Cycle Output)
- 2-MHz Base Operation
- Interrupt Acknowledge

Compatible with All 6800 Family Peripherals

the architectural improvements

Programming Model



*The concatenation of A:B is the Double-Accumulator.

10 Addressing Modes

- Inherent
- Immediate
- Direct
- Extended
- Extended Indirect
- Register
- Indexed
- Indexed Indirect
- Relative
- Long Relative

Powerful Indexing Capabilities

- There are 5 Indexable Registers: X, Y, S, U, and PC
- With 4 Options for Both Indexed and Indexed Indirect:

  Constant-Offset
  Accumulator Offset Using A, B, or D
  Auto-Increment or Auto-Decrement
  Indirection

6 Interrupts

- NMI Non-maskable
- IRQ Normal-Maskable
- FIRQ Fast-Maskable
- SWI Software
- SWI2 Software
- SWI3 Software

16-Bit Operations

- Add to D Accumulator
- Subtract from D Accumulator
- Load D Accumulator
- Store to D Accumulator
- Compare D Accumulator
- Load X, Y, S, U Registers
- Store X, Y, S, U Registers
- Compare X, Y, S, U Registers
- Load Effective Address
- Sign Extend
- Transfer Registers
- Exchange Registers
- Push/Pull X, Y, S, U
- Add B Accumulator to X Register

Additional Operations

- 8 × 8 Unsigned Multiply = 16-Bit
- Push/Pull Multiple Registers
- Long Branches
- Synchronize with Interrupt Line (to synchronize instructions with an external event)
- 3 Software Interrupts

AN 8-BIT MICROPROCESSOR WITH 16-BIT OPERATIONS

software features

The MC6809 attacks the problem of high-cost software

HIGH-LEVEL LANGUAGE EFFICIENCY

Specifically designed to efficiently handle high-level languages such as BASIC, MPL, COBOL, FORTRAN, and PASCAL.

POSITION-INDEPENDENT CODE

Any program written for the MC6809 can be structured to execute in any position of the address map.

REENTRANT PROGRAM

Any program written for the MC6809 can be structured to be interrupted in any position of the address map and still execute properly on return.

What do these features mean?

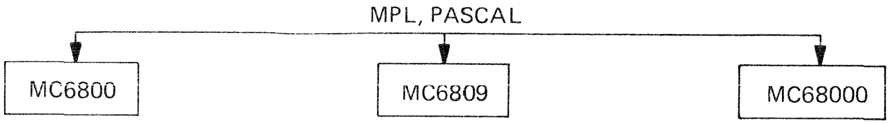
- Canned Software, "Software on Silicon"
- Modular Programming
- Ease of Use of Program Library
- Interchangeability of Program
- Efficient Programs Written in a Higher-Level Language

The software of the MC6809 is upward-software-compatible with the MC6800.

MC6800

MC6809

Common higher-level-language bridge.



MPL, PASCAL

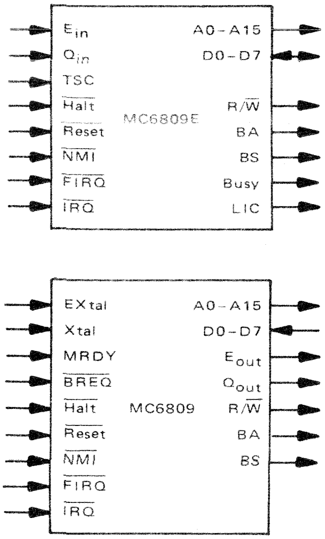
MC6800

MC6809

MC68000

hardware improvements

BLOCK DIAGRAMS



- On-Chip Clock (MC6809)
  Internal ÷ 4 Clock
- Off-Chip Clock (MC6809E)
  ÷ 1 TTL Input

Additional Interrupts

- NMI Non-maskable
- IRQ Maskable
- FIRQ Fast Response IRQ, Maskable

- **Busy**

  The busy output indicates that the processor is accessing memory.

- **LIC, Last Instruction Cycle**

  Indicates the processor is executing the last cycle of an instruction. The next cycle will be an opcode fetch.

- **MRDY, Memory Ready**

  Stretches the E signal for slow memory.

- **BREQ, Bus Request**

  Bus request to allow fast access to the bus for DMA or memory refresh.

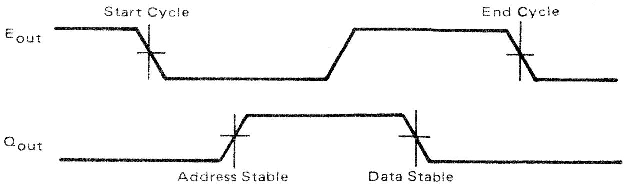
- **BA and BS, Bus Available and Bus Status**

BA	BS	Function
0	0	Normal
0	1	Interrupt Acknowledge
1	0	Sync Acknowledge
1	1	Bus Grant or Halt

- **Interrupt Acknowledge (BA = 0, BS = 1)**

  Indicates that a vector is being fetched by the CPU in response to an interrupt. Can be used to vector the interrupt.

BUS TIMING – ENABLE OUTPUT AND QUADRATURE OUTPUT



MC6809 development support

Advance MC6809 Support

To facilitate the design before the availability of an MC6809, Motorola has developed a Cross Macro Assembler and a Simulator Module that will operate in the EXORciser/EXORterm. The Simulator Module plugs directly into the EXORciser, allowing the user to simulate MC6809 instructions which were developed with the MC6809 Macro Assembler.

MC6809 Support Products

The MC6809 support package – MPU/USE Module, firmware, and software – upgrades the EXORciser I, II or EXORterm to an MC6809 EXORciser.

MC6809 EXORciser Features:

- Real-Time Execution
- 1.0, 1.5, and 2.0 MHz Operation
- Disk-Based Operation
- Emulate the user's MC6809 via a 40-pin DIP socket to be plugged into the user's MC6809 socket in the target system.

The software support of the MC6809 will include the following items:

- Resident Macro Assembler
- Resident Editor
- Disk Operating System (MDOS)
- Resident PASCAL Compiler
- Resident MPL Compiler

further support

- 3rd Quarter introduction of System Analyzer firmware and PROM Programmer software.
- 3rd Quarter introduction of an MC6809 micromodule (single board computer).
- Additional higher-level languages are planned for introduction in the 3rd and 4th quarters of 1979.

    1. Resident FORTRAN Compiler
    2. Resident COBOL Compiler
    3. Resident BASIC Interpreter
    4. Symbolic Debug Software
    5. Real-Time Operating System in a Micromodule Environment

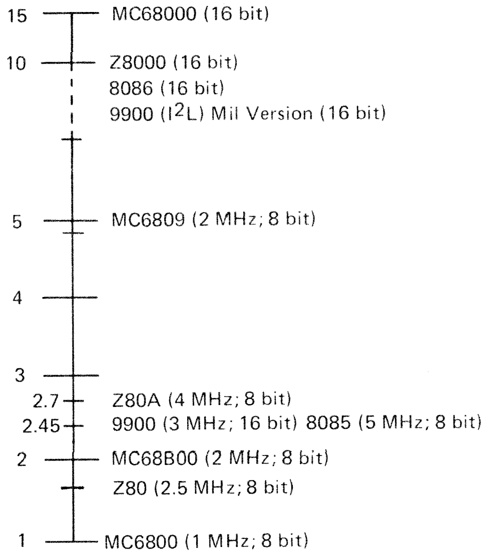
Training Support

The MC6809 is supported by two training courses:

1. One-day training course for previous MC6800-family users.
2. Three-day training course for new MC6800-family users.

with all these improvements, how does the MC6809 benchmark against competition?

EXECUTION TIME RELATIVE



SUMMARIZED BENCHMARKS

Performance Criteria	MC6809	Z80A	MC6800	8085
Number of Instructions	*1.0	1.56	1.72	2.30
Number of Bytes	1.0	1.31	1.58	1.80
Number of Microseconds	1.0 (2 MHz)	1.80 (4 MHz)	2.40 (2 MHz)	2.20 (5 MHz)

*Normalized to 1.00 for the MC6809 – poorer performance has higher numbers.

relative execution times for eight benchmarks

		I/O Handler	Character Search	Computed Go To	Double Shift Right 5 Bit	Vector Addition 16-Bit Elements	Vector Addition 8-Bit Elements	16 × 16-Bit Multiplication	Move Block (64 Bytes)	Average Execution Time
6809	2.0 MHz	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	1.5 MHz	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3	1.3
	1.0 MHz	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
Z80	4.0 MHz	1.4	0.8	2.1	2.7	1.6	1.8	3.3	1.0	1.8
	2.5 MHz	2.2	1.2	3.4	4.4	2.6	2.9	5.2	1.6	2.9
9900	3.0 MHz	2.6	2.3	2.8	1.5	1.7	3.0	0.5	1.6	2.0
6800	2.0 MHz	0.9	1.4	1.9	1.3	3.1	2.8	5.0	3.3	2.4
	1.5 MHz	1.2	1.9	2.5	1.7	4.1	3.7	6.7	4.3	3.3
	1.0 MHz	1.8	2.8	3.7	2.5	6.1	5.5	10	6.5	4.9
8080+	3.0 MHz	1.9	1.8	2.8	6.1	2.3	2.7	9.6	2.4	3.7
8085	2.0 MHz	2.8	2.6	4.2	9.1	3.4	4.1	14.3	3.7	5.5

actual execution times for eight benchmarks (μs)

		I/O Handler	Character Search	Computed Go To	Double Shift Right – 5 Bits	Vector Addition 16-Bit Elements	Vector Addition 8-Bit Elements	16 × 16-Bit Multiplication	Move Block (64 Bytes)
6809	2.0 MHz	28	287.5	34.5	15	325	180	82	344.5
	1.5 MHz	37.3	383	46	20	433	240	109.3	459.3
	1.0 MHz	56	575	69	30	650	360	164	689
Z80	4.0 MHz	38.3	220.5	73.3	41	518	323	267	342
	2.5 MHz	61.3	352.8	117.2	65.6	828.8	516.8	427.2	547.6
9900	3.0 MHz	72	661	98	22	537	537	42	537
6800	2.0 MHz	24.5	404	64.5	19	993.5	498.5	409.5	1123.5
	1.5 MHz	32.7	539	86	25.3	1325	665	546	1498
	1.0 MHz	49	808	129	38	1987	997	819	2247
8080+	3.0 MHz	52.7	506.7	96.7	91.3	732	492	784	841
8085	2.0 MHz	79	760	145	137	1098	738	1176	1262

sample programs

These programs were used in the benchmark evaluation and represent the software flexibility and architectural power of the MC6809. These benchmarks were chosen because they are unusually well-defined and have been applied to other computers for comparison.

```
M6800-M6809 CROSS-ASSEMBLER  2.2

00007                      ********** I/O HANDLER **********
00008                      *
00009                      *     A SINGLE INPUT INTERRUPT IS ARMED.  RECEI
00010                      *     AN INTERRUPT, SAVE REGISTERS, INPUT A CHA
00011                      *     CLEAR THE INTERRUPT, PUT THE CHAR IN A
00012                      *     SOFTWARE BUFFER, INCREMENT THE BUFFER PTR
00013                      *     TEST FOR END OF LINE, RECOVER REGISTERS,
00014                      *     AND RETURN.
00015                      *
00016                      *     SETUP:  NONE
00017                      *     TOTAL:  7 LN, 16 BY, 62 CY
00018                      *
00019                      ********************************

00021           000D       EOL    EQU   $0D      ASCII CR
00022 1004      00         MODEM  FCB   0
00023 1005      0100       BUFPTR FDB   $100

00025                      * ASSUME IRQ FROM PIA (19 CY)

00027 1007 B6   1004     5 BEGIN  LDA   MODEM    CLEARS PIA IRQ
00028 100A BE   1005     6        LDX   BUFPTR   GET PTR
00029 100D A7   80       6        STA   ,X+      STORE CHAR
00030 100F BF   1005     6        STX   BUFPTR   UPDATE PTR
00031 1012 81   0D       2        CMPA  #EOL     END OF LINE?
00032 1014 27   01       3        BEQ   EOLGP    IF YES, MORE TO DO
00033 1016 3B           15        RTI            ELSE, RETURN

00035 1017 20   FE       3 EOLGP  BRA   *
```

```
00038                         *********** CHARACTER SEARCH ***********
00039                         *
00040                         *     SEARCH A TABLE OF N CHARACTERS FOR A SPECIFIC
00041                         *     CHARACTER.  IF FOUND, RETURN THE ADDRESS OF
00042                         *     THE MATCH, ELSE RETURN ZERO.  LET N BE 40.
00043                         *     LET THE SEARCH FAIL.
00044                         *
00045                         *     SETUP:        3 LN,  7 BY,   7 CY
00046                         *     OPERATION:    6 LN, 12 BY, (14*40)+8=568 CY
00047                         *     TOTAL:        9 LN, 19 BY, 575 CY
00048                         *
00049                         **************************************

00051 1019 86   4A       2 CSRCH  LDA   #CHAR     CHAR TO FIND
00052 101B 8E   102E     3        LDX   #BUF      PTR INTO TABLE
00053 101E C6   28       2        LDB   #40       LENGTH OF TABLE

00055 1020 A1   80       6 CS1    CMPA  ,X+       SAME CHAR?
00056 1022 27   06       3        BEQ   CS2       IF YES, POINT AT IT
00057 1024 5A            2        DECB            ANOTHER ONE DOWN
00058 1025 26   F9       3        BNE   CS1       ALL DONE?
00059 1027 8E   0001     3        LDX   #1        TRICKY CLRX
00060 102A 30   1F       5 CS2    LEAX  -1,X      WENT PAST!

00062 102C 20   FE       3        BRA   *


00064           004A       CHAR   EQU   'J
00065 102E      00         BUF    FCB   0,,,,,,,,,,,,,,,,,,,0
00066 1042      00                FCB   0,,,,,,,,,,,,,,,,,,,0
```

```
M6800-M6809 CROSS-ASSEMBLER  2.2


00069                             ********** COMPUTED GO TO **********
00070                             *
00071                             *     LSB FIRST, TEST A CONTROL BYTE WHICH HAS
00072                             *     HAS EXACTLY ONE BIT TRUE.  THE POSITION
00073                             *     OF THE TRUE BIT DETERMINES WHICH OF EIGHT
00074                             *     TABLE VECTORS IS USED FOR CONTROL-TRANSFER
00075                             *     LET B7 BE TRUE.
00076                             *
00077                             *     SETUP:        2 LN,  5 BY,   5 CY
00078                             *     OPERATION:    5 LN,  8 BY, 2+(7*8)+7=65 CY
00079                             *     TOTAL:        7 LN, 13 BY,  70 CY
00080                             *
00081                             ************************************


00083 1056 86    80         2 COMPGO LDA    #CONTBY
00084 1058 8E    1061       3        LDX    #TABLE-2 START OF TABLE


00086 105B 5F               2        CLRB
00087 105C CB    02         2 C01    ADDB   #2         TWO BYTES / VECTOR
00088 105E 44               2        LSRA
00089 105F 24    FB         3        BCC    C01
00090 1061 6E    95         7        JMP    [B,X]      REGISTER-OFFSET INDIRECT



00092            0080         CONTBY EQU    $80
00093 1063       1073         TABLE  FDB    ERR,ERR,ERR,ERR,ERR,ERR,ERR
00094 1071       1075                FDB    NOERR
00095 1073 20    FE         3 ERR    BRA    *
00096 1075 20    FE         3 NOERR  BRA    *

M6800-M6809 CROSS-ASSEMBLER  2.2


00201                             ********** DOUBLE SHIFT RIGHT FIVE PLACES *********
00202                             *
00203                             *     LOGICALLY SHIFT RIGHT A 16-BIT QUANTITY
00204                             *     FROM MEMORY EXACTLY 5 PLACES.
00205                             *     REPLACE THE RESULT IN MEMORY.
00206                             *
00207                             *     SETUP:  NONE
00208                             *     TOTAL:  12 LN, 16 BY, 30 CY
00209                             *
00210                             ***************************************************


00212 1171 FC    1183       6        LDD    WORD       GET DOUBLE BYTE
00213 1174 44               2        LSRA              : 16-BIT SHIFT
00214 1175 56               2        RORB              :
00215 1176 44               2        LSRA              AGAIN
00216 1177 56               2        RORB
00217 1178 44               2        LSRA              AGAIN
00218 1179 56               2        RORB
00219 117A 44               2        LSRA              AGAIN
00220 117B 56               2        RORB
00221 117C 44               2        LSRA              AGAIN
00222 117D 56               2        RORB
00223 117E FD    1183       6        STD    WORD       STORE DOUBLE BYTE


00225 1181 20    FE         3        BRA    *



00227 1183       F1CD         WORD   FDB    $F1CD
```

```
00099                               ********** VECTOR ADDITION / 16-BIT **********
00100                               *
00101                               *     PERFORM AN ELEMENT-BY-ELEMENT ADDITION ON
00102                               *     TWO VECTORS OF N 16-BIT ELEMENTS EACH.
00103                               *     PLACE THE RESULT IN A DIFFERENT VECTOR.
00104                               *     LET N BE 20.
00105                               *
00106                               *     SETUP:         3 LN, 10 BY,  10 CY
00107                               *     OPERATION:     5 LN, 11 BY, 32*20=640 CY
00108                               *     TOTAL:         8 LN, 21 BY, 650 CY
00109                               *
00110                               **********************************************


00112 1077 8E   108E      3 ANBNCN LDX    #TABLEA
00113 107A 108E 10B6      4        LDY    #TABLEB
00114 107E CE   10DE      3        LDU    #TABLEC


00116 1081 EC   81        8 AN1    LDD    ,X++
00117 1083 E3   A1        9        ADDD   ,Y++
00118 1085 ED   C1        8        STD    ,U++
00119 1087 8C   10B6      4        CMPX   #2*20+TABLEA
00120 108A 26   F5        3        BNE    AN1


00122 108C 20   FE        3        BRA    *



00124 108E      0000        TABLEA FDB    $00,$01,$02,$03,$04
00125 1098      0005               FDB    $05,$06,$07,$08,$09
00126 10A2      0010               FDB    $10,$11,$12,$13,$14
00127 10AC      0015               FDB    $15,$16,$17,$18,$19
00128 10B6      0099        TABLEB FDB    $99,$98,$97,$96,$95
00129 10C0      0094               FDB    $94,$93,$92,$91,$90
00130 10CA      0089               FDB    $89,$88,$87,$86,$85
00131 10D4      0084               FDB    $84,$83,$82,$81,$80
00132 10DE      0000        TABLEC FDB    0,,,,,,,,,,,,,,,,,,,0
```

```
00135                          ********** VECTOR ADDITION / 8-BIT **********
00136                          *
00137                          *     PERFORM AN ELEMENT-BY-ELEMENT ADDDITION
00138                          *     ON TWO VECTORS OF N 8-BIT ELEMENTS EACH.
00139                          *     PLACE THE RESULT IN A DIFFERENT VECTOR.
00140                          *     LET N BE 20.
00141                          *
00142                          *     SETUP:         3 LN, 10 BY,  10 CY
00143                          *     OPERATION:     6 LN, 13 BY, 10*35=350 CY
00144                          *     TOTAL:         9 LN, 23 BY, 360 CY
00145                          *
00146                          *******************************************


00148 1106 8E   111F      3 ABCNNN LDX    #TABLA
00149 1109 108E 1133      4        LDY    #TABLB
00150 110D CE   1147      3        LDU    #TABLC


00152 1110 EC   81        8 ABC1   LDD    ,X++
00153 1112 AB   A0        6        ADDA   ,Y+
00154 1114 EB   A0        6        ADDB   ,Y+
00155 1116 ED   C1        8        STD    ,U++
00156 1118 8C   1133      4        CMPX   #TABLA+20
00157 111B 26   F3        3        BNE    ABC1


00159 111D 20   FE        3        BRA    *




00161 111F      00           TABLA  FCB    $00,$01,$02,$03,$04
00162 1124      05                  FCB    $05,$06,$07,$08,$09
00163 1129      10                  FCB    $10,$11,$12,$13,$14
00164 112E      15                  FCB    $15,$16,$17,$18,$19
00165 1133      99           TABLB  FCB    $99,$98,$97,$96,$95
00166 1138      94                  FCB    $94,$93,$92,$91,$90
00167 113D      89                  FCB    $89,$88,$87,$86,$85
00168 1142      84                  FCB    $84,$83,$82,$81,$80
00169 1147      00           TABLC  FCB    0,,,,,,,,,,,,,,,,,,,0
```

```
00230                             ********** 16 X 16 MULTIPLY **********
00231                             *
00232                             *     MULTIPLY TWO 16-BIT POSITIVE VALUES
00233                             *     TO GENERATE A 32-BIT PRODUCT.
00234                             *     AT TERMINATION, BOTH INPUT VALUES
00235                             *     AND THE RESULT WILL BE IN MEMORY.
00236                             *
00237                             *     (A:B) X (C:D) =          BDH:BDL
00238                             *                   +      BCH:BCL
00239                             *                   +      ADH:ADL
00240                             *                   + ACH:ACL
00241                             *                    ----------------
00242                             *
00243                             *     SETUP:       3 LN, 10 BY,  10 CY
00244                             *     OPERATION: 25 LN, 46 BY, 154 CY
00245                             *     TOTAL:      28 LN, 56 BY, 164 CY
00246                             *
00247                             **************************************

00249 1185 8E   11BF       3 ABC     LDX   #AA       POINTER TO A (MS BYTE)
00250 1188 108E 11C1       4         LDY   #BB
00251 118C CE   11C3       3         LDU   #C

00253 118F 6F   C4         6         CLR   0,U
00254 1191 6F   41         7         CLR   1,U
00255 1193 A6   01         5         LDA   1,X       : #A LS BYTE
00256 1195 E6   21         5         LDB   1,Y       : #B LS BYTE
00257 1197 3D             11         MUL
00258 1198 ED   42         6         STD   2,U
00259 119A A6   84         4         LDA   0,X       : #A MS BYTE
00260 119C E6   21         5         LDB   1,Y       : #B LS BYTE
00261 119E 3D             11         MUL
00262 119F E3   41         7         ADDD  1,U
00263 11A1 ED   41         6         STD   1,U
00264 11A3 24   02         3         BCC   AB1
00265 11A5 6C   C4         6         INC   0,U
00266 11A7 A6   01         5 AB1     LDA   1,X       : #A LS BYTE
00267 11A9 E6   A4         4         LDB   0,Y       : #B MS BYTE
00268 11AB 3D             11         MUL
00269 11AC E3   41         7         ADDD  1,U
00270 11AE ED   41         6         STD   1,U
00271 11B0 24   02         3         BCC   AB2
00272 11B2 6C   C4         6         INC   0,U
00273 11B4 A6   84         4 AB2     LDA   0,X       : #A MS BYTE
00274 11B6 E6   A4         4         LDB   0,Y       : #B MS BYTE
00275 11B8 3D             11         MUL
00276 11B9 E3   C4         6         ADDD  0,U
00277 11BB ED   C4         5         STD   0,U

00279 11BD 20   FE         3         BRA   *

00281 11BF      03E8         AA      FDB   1000
00282 11C1      01F4         BB      FDB   500
00283 11C3      0000         C       FDB   0,0
```

```
00286                           ********** MOVE BLOCK **********
00287                           *
00288                           *     COPY N BYTES TO ANOTHER LOCATION
00289                           *     LET N BE 64.
00290                           *
00291                           *     SETUP:        3 LN, 10 BY,  10 CY
00292                           *     OPERATION: 7LN,  11 BY, 2+(21*32)+5=679 CY
00293                           *     TOTAL:      10 LN, 21 BY, 689 CY
00294                           *
00295                           ********************************

00297 11C7 CC   0020     3          LDD   #LENGTH/2
00298 11CA 108E 0100     4          LDY   #FROM
00299 11CE CE   0200     3          LDU   #TO

00301 11D1 4C            2          INCA               MS COUNT CORRECTION
00302 11D2 AE   A1       8 B1       LDX   ,Y++         GET TWO BYTES
00303 11D4 AF   C1       8          STX   ,U++         PUT TWO BYTES
00304 11D6 5A            2          DECB               LS COUNT
00305 11D7 26   F9       3          BNE   B1
00306 11D9 4A            2          DECA               MS COUNT
00307 11DA 26   F6       3          BNE   B1

00309 11DC 20   FE       3          BRA   *


00311           0100        FROM    EQU   $100
00312           0200        TO      EQU   $200
00313           0040        LENGTH  EQU   64
```

Reprinted with permission
of McGraw-Hill Book
Company from

MICROCOMPUTER-BASED
DESIGN
JOHN B. PEATMAN

MOTOROLA'S ADVANCED COMPUTER SYSTEM ON SILICON

MC6809

MOTOROLA Semiconductor Products Inc.

THE MC6809 HIGH-PERFORMANCE MICROPROCESSOR

With their 6809 CPU chip, Motorola has significantly upgraded their 6800 microcomputer family of parts. Because of its greatly enhanced instruction set, we will treat it here in a separate appendix*. And because the 6809's instruction set is an expansion of the 6800's instruction set, this appendix will build upon the material in Appendix A4**. In particular, we will explore some of the **implications** of the expanded capabilities since they are both far-reaching and subtle.

The 6809's design encompasses major features heretofore not found in a microcomputer. For those who wish to use it for general purpose computing in a floppy disk environment, the 6809 supports the generation of **position-independent** code. Programs prepared in this way can be brought from disk into RAM and located anywhere without requiring the services of a "relocating loader." The 6809 also supports the compilation of high-level, block-structured languages like PASCAL with extensive stack-handling capability (and up to four separate stacks).

For the instrument designer, the 6809 will permit programs to be written in PASCAL (or other high-level, block-structured language) and to be compiled into significantly more efficient, faster running, machine code than has been previously possible in a microcomputer. This is a consequence of the variety of features in the 6809 which can handle the block structure of the high-level language directly.

The instrument designer will also find a new and unique industry springing up to aid software development for the 6809 which will not exist for other microcomputers (presently available). The 6809 permits code to be written for general distribution, in ROM form, which can be assigned an arbitrary address on the address bus by the instrument designer and which will not interfere with any of the designer's other software. Manufacturers of peripheral devices can offer a working, debugged, flawless driver routine for their device in the form of a low-cost ROM for the 6809. Likewise, developers of generally useful software packages (e.g., math routines, real-time executive routines, etc.) can market these as ROMs for the 6809 with the knowledge that they will not give rise to bugs in other software. Finally, the designer's own code can be written in modular form with the knowledge that as the software grows, the interactions between parts can be kept in good working order as a natural consequence of the structure of the code. Later in this appendix we will explore the features of the 6809 which yeild these capabilities.

The philosophy taken by the designers of the 6809 has been to achieve the upgrading of 6800 system capability while maintaining 6800 system compatibility. These designers made a brilliant decision in the manner in which they achieved software compatibility with the 6800. Rather than constrain themselves to maintain the **machine code** instruction definitions, they instead maintained the assembly language **source code** instruction definitions. Thus, any assembly language code prepared for the 6800 can be passed through a 6809 assembler to produce code which will run on the 6809. As a consequence, the 6809 instruction set has been powerfully augmented and yet the coding of these instructions has been optimized.

The register structure of the 6809 has been enhanced, from that of the 6800, to include:

- A second, sixteen-bit index register.
- A second, sixteen-bit "user" stack pointer.
- An eight-bit direct-page register which permits a user to determine (and to change) **which** page of memory is accessed by the instructions employing "page zero" addressing.

Some of the most significant features of the 6809 augmented instruction set are:

- The replacement of the 6800's indexed addressing capablity by a **family** of addressing capabilities which can employ any of the 6809's four index registers and stack pointers as a pointer to an operand or as a pointer to the **address** of an operand (i.e., indirect addressing). The pointer can have either a fixed or a variable signed offset. It can be (optionally, but automatically) incremented or decremented. Even the program counter can be used as a pointer to access operands, or addresses of operands, using relative addressing. Finally, any of the indexed addressing modes can be used to form an address which can then be loaded into any of the index registers or stack pointers.

*This material is Appendix 7 of Dr. Peatman's book; the Appendix nomenclature has been retained.

**The MC6800 Instruction Set, referenced as Appendix A4, is included on the Motorola MC6800 data sheet. It is felt that this discussion of the 6809 is not dependent on the presence of the 6800 information.

- Sixteen-bit add, subtract, compare, load, and store instructions for the sixteen-bit "D" accumulator (made up of the A and B accumulators) plus the ability to Add A, B, or D to any of the CPU's index registers and stack pointers, plus the ability to perform exchanges between any two like-sized CPU registers. This greatly enhances the ability to manipulate pointers.
- A multiply instruction which forms the sixteen-bit product of the two eight-bit unsigned binary numbers in A and B.
- Two-byte instructions which will push, or pull, any number of CPU registers onto, or from, either stack.
- The addition of "long branch" instructions so that **relative** addressing can be used throughout a program.

Some of the hardware enhancements of the 6809 are:

- The inclusion of an on-chip clock, eliminating the need for a separate clock chip.
- The inclusion of a Schmitt trigger on the "Reset" input so that nothing more than an RC circuit is needed to initialize operations when power comes up.
- The elimination of VMA (Valid Memory Address) and the corresponding gating needed to use it in device selection. The 6809 appears to other devices to be reading from address FFFF during any clock cycle when it is not actually using the address bus/data bus structure.
- Support of three vectored priority-interrupt levels, each of which automatically disables lower priority interrupt levels while leaving higher priority interrupt levels enabled.

The 6809 register structure is shown in Fig. A7-1. The two most significant bits of the condition code register have been put to use in the 6809. E is an "entire" flag bit which is used automatically by an RTI (return from Interrupt) instruction. It draws a distinction between a return from the new FIRQ (fast interrupt request) interrupt on the one hand (for which only the condition code register and the program counter must be automatically restored from the stack) and a return from IRQ and NMI interrupts on the other hand (for which the entire CPU state must be automatically restored from the stack).

The F bit of the condition code register is a mask bit for FIRQ interrupts. If it is clear and if the FIRQ input is pulled low, then a fast interrupt is initiated. The E bit is cleared, the program counter and condition code register are stacked, and the F and I mask bits are set (masking further FIRQ and IRQ interrupts). The CPU then vectors to the FIRQ service routine whose address is stored in addresses FFF6 and FFF7.

In contrast, if the I bit is clear and if the IRQ input is pulled low, then an IRQ interrupt is initiated. The E bit is set, all CPU registers (except S) are stacked, and the I bit is set. The CPU then vectors to the IRQ service routine whose address is stored in addresses FFF8 and FFF9. Because the F bit is not automatically set by the IRQ interrupt, the FIRQ is a higher priority interrupt input which can interrupt the service routine of an IRQ interrupt service routine.

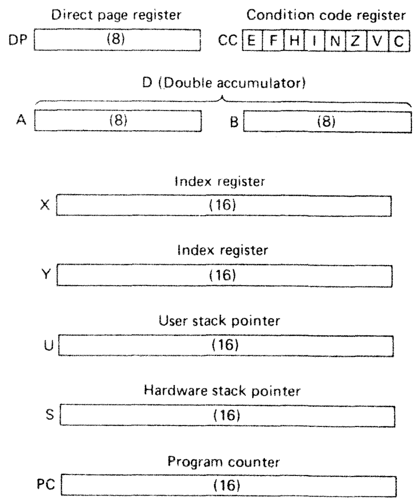
The NMI (non-maskable interrupt) input has the highest priority of all. A negative edge on the NMI input unconditionally causes an NMI interrupt which sets E, stacks all CPU registers (except S), sets the I and F mask bits, and vectors to the NMI service routine whose address is stored in addresses FFFC and FFFD. Thus, with no extra hardware and no extra instructions the 6809 can support three vectored priority interrupts. In addition, the IRQ and FIRQ inputs can each handle several sources of interrupts using a polling routine, just as is done with the 6800's IRQ input, as described in Appendix A4.

The DP (direct page) register of Fig. A7-1 defines which page of memory is accessed by "direct" addressing. If DP is cleared (as is automatically done by Reset), then the "direct" addressing of the 6809 becomes the "page zero" addressing of the 6800.

Some 6809 instructions deal with the sixteen-bit double accumulator D, made up of accumulators A and B. For such instructions, A holds the upper byte while B holds the lower byte.

The four sixteen-bit pointers X, Y, U, and S each provide identical capability in accessing operands for any instruction which permits the "indexed" family of addressing modes, to be discussed

Figure A7-1: Motorola 6809 CPU Registers



shortly. In addition, the two stack pointers, U and S, make use of push and pull instructions to stack CPU registers under program control. Finally, it is the hardware stack pointer, S, which corresponds to the 6800's stack pointer for handling return addresses and CPU registers automatically during subroutine calls and interrupts.

The 6809 instruction set is given in Fig. A7-2. While the CPU register involved in an instruction is broken away from the mnemonic and shown in the REG column, the 6809 assembly language actually expects the CPU register name to be combined with the mnemonic, as in the following examples:

```
CLRA                (instead of CLR A)
STD     TEMP        (instead of ST D TEMP)
ORCC    #01         (instead of OR CC #01)
ADDD    ,X++        (instead of ADD D ,X++)
CMPX    #0123H      (instead of CMP X #0123H)
```

The four exceptions to this rule are the TFR, EXG, PSHS (PSHU), and PULS (PULU) instructions, as exemplified below:

```
TFR     D,X
EXG     A,DP
PULS    A,X,CC
```

The 6809 permits "full indirect" addressing of an operand. For example, in order to set up so that when a programmable timer causes an interrupt, a jump to address TIMR4 will be executed, we can first store the full address labeled TIMR4 in the two consecutive RAM locations TIMER and TIMER+1 using the sequence

```
LDX    #TIMR4    Load X with the address
                 labeled TIMR4
STX    TIMER     Store it in the RAM address-
                 pair labeled TIMER
```

Then, when subsequently an interrupt occurs, the following instruction in the interrupt service routine

```
JMP    [TIMER]
```

will load the program counter with the **contents** of the address-pair labeled TIMER.

Figure A7-2: Motorola 6809 Instruction Set (See Note 1)

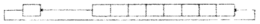
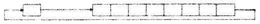
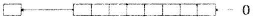
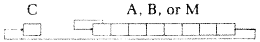
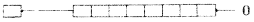
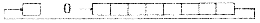
				← Bytes/Cycles →						
Operation	**MNE**	**REG**	**Description**	Inherent	Immediate	Direct (Page zero)	Full	Full indirect	Indexed family	Flags C Z N V
Move, 8-bit	TFR	R, R′	R′ ⟵ R } for R, R′ = A, B, CC, DP	2/6						— — — —
	EXG	R, R′	R′ ⟷ R }	2/7						— — — —
	CLR	A(B)	A ⟵ 0	1/2						0 1 0 0
	CLR		M ⟵ 0			2/6	3/7	4/12	2+/6+	0 1 0 0
	LD	A(B)	A ⟵ M		2/2	2/4	3/5	4/10	2+/4+	— ↕ ↕ 0
	ST	A(B)	M ⟵ A			2/4	3/5	4/10	2+/4+	— ↕ ↕ 0
Move, 16-bit	TFR	R, R′	R′ ⟵ R } for R, R′ = D, X, Y, U, S, PC							— — — —
	EXG	R, R′	R′ ⟷ R }							— — — —
	LD	R	R ⟵ M for R = D, X, U		3/3	2/5	3/6	4/11	2+/5+	— ↕ ↕ 0
			for R = Y, S		4/4	3/6	4/7	5/12	3+/6+	— ↕ ↕ 0
	ST	R	M ⟵ R for R = D, X, U			2/5	3/6	4/11	2+/5+	— ↕ ↕ 0
			for R = Y			3/6	4/7	5/12	3+/6+	— ↕ ↕ 0
Load effective address	LEA	R	R ⟵ Addres[illegible]					4/10	2+/4+	— ↕ — —
								4/10	2+/4+	— — — —
Move, bytes	PSHS	list	S [illegible] egisters	2/(4 + b)						— — — —
	PSHU	list	[illegible] CPU registers	(b = number						— — — —
	PULS	[illegible]	[illegible] sired C. [illegible] gisters ⟵ S stack	of bytes						(CC)
	PULU	li[illegible]	[illegible] of desired CPU registers ⟵ U stack	moved)						(CC)
Increment	INC	A(B)	A ⟵ A + 1	1/2						— ↕ ↕ ↕
	INC	R	R ⟵ R + 1 for R = X, Y	2/5						— ↕ — —
			for R = U, S	2/5						— — — —
	INC		M ⟵ M + 1			2/6	3/7	4/12	2+/6+	— ↕ ↕ ↕
Decrement	DEC	A(B)	A ⟵ A − 1	1/2						— ↕ ↕ ↕
	DEC	R	R ⟵ R − 1 for R = X, Y	2/5						— ↕ — —
			for R = U, S	2/5						— — — —
	DEC		M ⟵ M − 1			2/6	3/7	4/12	2+/6+	— ↕ ↕ ↕
Complement	COM	A(B)	A ⟵ $\overline{A}$	1/2						1 ↕ ↕ 0
	COM		M ⟵ $\overline{M}$			2/6	3/7	4/12	2+/6+	1 ↕ ↕ 0
Set selected CC bits	OR	CC	CC ⟵ CC OR M		2/3					↕ ↕ ↕ ↕
Clear selected CC bits	AND	CC	CC ⟵ CC AND M		2/3					↕ ↕ ↕ ↕

USE REVISED TABLE
SEE PAGE 11

Note 1. Since the writing of this material, some changes have been made in design of the MC6809. See new Instruction Set.

(continued)

Figure A7-2: Motorola 6809 Instruction Set (See Note 1) (continued)

Operation	MNE	REG	Description	Inherent	Immediate	Direct (Page zero)	Full	Full indirect	Indexed family	Flags
				← Bytes/Cycles →						C Z N V
Add, 8-bit	ADD	A(B)	A ← A + M		2/2	2/4	3/5	4/10	2+/4+	↕ ↕ ↕ ↕
Add with carry, 8-bit	ADC	A(B)	A ← A + M + C		2/2	2/4	3/5	4/10	2+/4+	↕ ↕ ↕ ↕
Decimal adjust acc. A	DAA		Correct BCD addition	1/2						↕ ↕ ↕ ↕
Add, 16-bit	ADD	D	D ← D + M		3/4	2/6	3/7	4/12	2+/6+	↕ ↕ ↕ ↕
	ABX		X ← X + B (unsigned addition of B to X)	1/3						— — — —
Sign extend	SEX		A ← FF if bit 7 of B is set 00 if bit 7 of B is clear	1/2						— ↕ ↕ 0
Subtract, 8-bit	SUB	A(B)	A ← A − M		2/2		3/5	4/10	2+/4+	↕ ↕ ↕ ↕
Subtract with carry, 8-bit	SBC	A(B)	A ← A − M − C			2/4	3/5	4/10	2+/4+	↕ ↕ ↕ ↕
Negate, 8-bit	NEG	A(B)	A ← 00 − A							↕ ↕ ↕ ↕
	NEG		M ← 00 − M			2/6	3/7	4/12	2+/6+	↕ ↕ ↕ ↕
Subtract, 16-bit	SUB	D	D ← D − M		3/4	2/6	3/7	4/12	2+/6+	↕ ↕ ↕ ↕
Multiply	MUL		D ← A*B (unsigned	1/11						↕ ↕ — —
AND	AND	A(B)	A ← A AN		2/2	2/4	3/5	4/10	2+/4+	— ↕ ↕ 0
Exclusive-OR	EOR	A(B)	A ←		2/2	2/4	3/5	4/10	2+/4+	— ↕ ↕ 0
OR (inclusive)	OR	A(B)	A		2/2	2/4	3/5	4/10	2+/4+	— ↕ ↕ 0
Compare, 8-bit	CMP	A			2/2	2/4	3/5	4/10	2+/4+	↕ ↕ ↕ ↕
	TST	A,	− 00	1/2						— ↕ ↕ 0
	TST		M − 00 Only			2/6	3/7	4/12	2+/6+	— ↕ ↕ 0
	BIT	A(B)	A AND M flags are		2/2	2/4	3/5	4/10	2+/4+	— ↕ ↕ 0
Compare, 16-bit	CMP	R	R-M for R = X affected		3/4	2/6	3/7	4/12	2+/6+	↕ ↕ ↕ ↕
			for R = D, Y, U, S		4/5	3/7	4/8	5/13	3+/7+	↕ ↕ ↕ ↕
Rotate left	ROL	A(B)	C A, B, or M	1/2						↕ ↕ ↕ ↕
	ROL					2/6	3/7	4/12	2+/6+	↕ ↕ ↕ ↕
Rotate right	ROR	A(B)	C A, B, or M	1/2						↕ ↕ ↕ —
	ROR					2/6	3/7	4/12	2+/6+	↕ ↕ ↕ —
Arithmetic shift left	ASL	A(B)	C A, B, or M	1/2						↕ ↕ ↕ ↕
	ASL					2/6	3/7	4/12	2+/6+	↕ ↕ ↕ ↕
Arithmetic shift right	ASR	A(B)	C A, B, or M	1/2						↕ ↕ ↕ —
	ASR					2/6	3/7	4/12	2+/6+	↕ ↕ ↕ —
Logic shift left	LSL	A(B)	C A, B, or M	1/2						↕ ↕ ↕ ↕
	LSL					2/6	3/7	4/12	2+/6+	↕ ↕ ↕ ↕
Logic shift right	LSR	A(B)	C A, B, or M	1/2						↕ ↕ 0 —
	LSR					2/6	3/7	4/12	2+/6+	↕ ↕ 0 —
No operation	NOP		PC ← PC + 1	1/2						— — — —

USE REVISED TABLE
SEE PAGE 12

(Continued)

Note 1. Since the writing of this material, some changes have been made in design of the MC6809. See new Instruction Set.

Figure A7-2: Motorola 6809 Instruction Set (continued) (See Note 1)

USE REVISED TABLE SEE PAGE 13

Operation	MNE	Description	Bytes/Cycles Inherent	Short relative	Long relative	Direct (Page zero)	Full	Full indirect	Indexed family	Flags C Z N V
Jump unconditionally	JMP	PC ⟵ M				2/3	3/4	4/9	2+/3+	— — — —
Branch unconditionally	BRA	PC ⟵ PC + M		2/3	3/5					— — — —
if carry set	BCS	if C = 1		2/3	4/6(5)					— — — —
if carry clear	BCC	if C = 0		[illegible]	4/6(5)					— — — —
if equal zero	BEQ	if Z = 1		[illegible]	4/6(5)					— — — —
if not equal zero	BNE	if Z = 0		[illegible]	4/6(5)					— — — —
if minus	BMI	if N = 1		2/3	4/6(5)					— — — —
if plus	BPL	if N = 0		2/3	4/6(5)					— — — —
if *overflow set*	BVS	*if V = 1* LBCS,		2/3	4/6(5)					— — — —
if overflow clear	BVC	if V = 0 s the		2/3	4/6(5)					— — — —
if > zero	BGT	if Z [illegible] it the		2/3	4/6(5)					— — — —
if ≥ zero Signed	BGE	[illegible] relative		2/3	4/6(5)					— — — —
if ≤ zero numbers	BLE	[illegible] branch is desired		2/3	4/6(5)					— — — —
if < zero	BLT	[illegible] N ⊕ V		2/3	4/6(5)					— — — —
if higher	B[illegible]	if C + Z = 0		2/3	4/6(5)					— — — —
if higher or same Unsigned	BH[illegible]	if C = 0		2/3	4/6(5)					— — — —
if lower or same numbers	BLS	if C + Z = 1		2/3	4/6(5)					— — — —
if lower	BLO	if C = 1		2/3	4/6(5)					— — — —
Jump to subroutine	JSR	S stack ⟵ PC; PC ⟵ M				2/7	3/8	4/13	2+/7+	— — — —
Branch to subroutine	BSR	S stack ⟵ PC; PC ⟵ PC + M (or use LBSR)		2/7	3/9					— — — —
Return from subroutine	RTS	PC ⟵ S stack	1/5							— — — —

Operation	MNE	Description	Bytes/Cycles Inherent	Immediate	Flags C Z N V
Software interrupt	SWI	S stack ⟵ PC, U, Y, X, DP, B, A, CC; Set I, F; PC ⟵ M(FFFA, FFFB)	1/19		— — — —
	SWI2	S stack ⟵ PC, U, Y, X, DP, B, A, CC; Set I, F; PC ⟵ M(FFF4, FFF5)	2/20		— — — —
	SWI3	S stack ⟵ PC, U, Y, X, DP, B, A, CC; Set I, F; PC ⟵ M(FFF2, FFF3)	2/20		— — — —
Clear and wait	CWAI	CC ⟵ CC AND M; S stack ⟵ PC, U, Y, X, DP, B, A, CC; Halt		2/20	↕ ↕ ↕ ↕
Synchronize I/O	SYNC	Halt; Continue when interrupt occurs (see text)	1/2		— — — —
Return from interrupt	RTI	CC ⟵ S stack; If E is set, then A, B, DP, X, Y, U, PC ⟵ S stack If E is cleared, then PC ⟵ S stack	1/15(6)		↕ ↕ ↕ ↕

Note 1. Since the writing of this material, some changes have been made in design of the MC6809. See new Instruction Set.

The extensive new addressing possibilities made possible by Motorola's redefinition of its 6800 indexed addressing mode into its 6809 indexed **family** of addressing modes is hardly evident in Fig. A7-2. A "bytes/cycles" listing of 2+/4+ indicates the **minimum** bytes and cycles taken by any of these modes. The actual bytes and cycles required are found by adding the value listed in Fig. A7-3 for the indexed addressing possibility selected to the value listed in Fig. A7-2 for the instruction operation selected. For example, the instruction

```
LDA     3,S
```

which loads accumulator A with the fourth byte into the stack is actually a 2-byte, 5-cycle instruction. Before discussing the implications of these indexed addressing modes, we will clarify other instructions listed in Fig. A7-2.

The load effective address instruction, LEA, permits all of the flexibility of the 6809's indexed addressing modes to be used to form a pointer and to load it into any one of the index registers or stack pointers. For example, the two-byte instruction

```
LEAX    3,S
```

loads the X index register with a pointer to the fourth byte into the hardware stack.

The instructions for incrementing, decrementing, and adding into pointers are actually the following instructions:

```
LEAX    1,X      Increment X
LEAY    — 1,Y    Decrement Y
LEAU    D,U      U ← U + D
```

Also, LEAX B,X is similar, but not identical to the ABX instruction. The former instruction treats B as a 2s-complement, signed number whereas ABX uses B as a positive offset between 0 and 255.

A lot of the power of the 6809 resides in the flexibility of the indexed addressing modes. An assembly language user will achieve more or less powerful use of the 6809 depending upon the amount of thought given the use of these indexed addressing modes, and of the LEA instruction. A high-level language user will find these used in powerful ways automatically.

The push and pull instructions permit us to set aside, and later restore, any combination of CPU registers on either stack using two-byte instructions, regardless of the number of registers involved. While we have the freedom to list whatever registers we want stacked or restored, the **order** of stacking or restoring is a hardware function, with the order shown in Fig. A7-4 (if **all** registers are included). Thus the instructions

```
        PSHS    A,CC,X
```

and

```
        PSHS    A,X,CC
```

will actually push X, then A, then CC onto the hardware stack (pointed to by S). Since four bytes are moved, this instruction takes 4 + 4 = 8 cycles to execute, or 8 microseconds, using a 1.0 MHz clock-rate 6809.

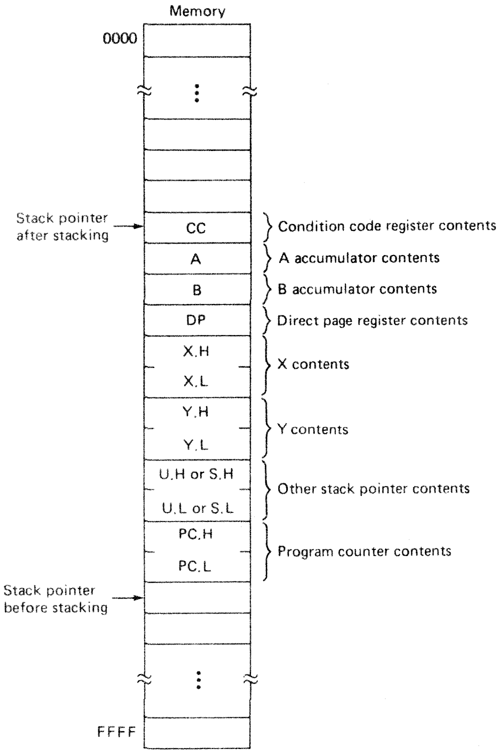
Figure A7-3: Indexed Addressing Options (See Note 1)

Indexed addressing form		Source code form	Description (where R = X, Y, U, or S)	Additional bytes	Additional cycles
Non-indirect addressing	No offset, no change in R	,R	Get operand pointed to by R	0	0
	No offset, change R	, R+	Get operand and then increment R once	0	+2
		, R+ +	Get operand and then increment R twice	0	+3
		, −R	Decrement R once and then get operand	0	+2
		, − −R	Decrement R twice and then get operand	0	+3
	Fixed offset	offset, R	Temporarily add signed offset (expressed as sign plus magnitude) to R and then get operand, for −16 ≤ offset ≤ +15	0	+1
			−128 ≤ offset ≤ +127	+1	+2
			−65536 ≤ off[illegible] +65536	+2	+4
	Variable offset	A, R B, R D, R	Temporarily add [illegible] (held in accumulato [illegible] plement form) to R and th [illegible] nd	0 0 0	+1 +1 +4
	Relative addressing	label, PC[illegible]	[illegible] essing to access a [illegible] aving a label [illegible] nd located within ±128 bytes of the next instruction	+1	+3
			– and located anywhere	+2	+5
		offset, PC	Temporarily add signed offset (expressed as sign plus magnitude) to the address of the next instruction and then get operand, for −128 ≤ offset ≤ +127	+1	+3
			−65536 ≤ offset ≤ +65536	+2	+5
Indirect addressing Use operand found as above as a pointer to the actual operand	No offset, no change in R	[, R]		0	+3
	No offset, change R	[, R++] [, − −R]		0 0	+6 +6
	Fixed offset	[offset, R]	−128 ≤ offset ≤ +127	+1	+5
			−65536 ≤ offset ≤ +65536	+2	+7
	Variable offset	[A, R] [B, R] [D, R]		0 0 0	+4 +4 +7
	Relative addressing	[label, PCR]	For label located within ±128 bytes of the next instruction	+1	+6
			For label located anywhere	+2	+8
		[offset, PC]	−128 ≤ offset ≤ +127	+1	+6
			−65536 ≤ offset ≤ +65536	+2	+8

USE REVISED TABLE
SEE PAGE 14

Note 1. Since the writing of this material, some changes have been made in design of the MC6809. See new Instruction Set.

Figure A7-4: 6809 Stacking Order



Note that operands pushed onto the stack go into successively decreasing addresses, just as in the 6800. However, the 6809 stack pointers actually point to the top of the stack (i.e., to the last operand pushed onto the stack) rather than to the address just below this, as is done by the 6800's stack pointer. Even though this represents a hardware difference between the 6800 and the 6809, the related instructions are designed to achieve identical results. For example, the 6800's TSX instruction is equivalent to the 6809's TFR S,X instruction in the sense that the resulting contents of X will, in both cases point to the top of the stack. Recall that the 6800's TSX instruction loads X with an address one higher than that held in the stack pointer.

To set or clear selected bits of the condition code register, the 6809 ANDs or ORs an immediate operand into the condition code register. Thus, the 6800's six one-byte instructions SEC, CLC, SEV, CLV, SEI, and CLI instructions have been replaced by the two two-byte instructions

ORCC and ANDCC

For example,

ORCC #00000001B

will set the least significant bit of the condition code register. Looking at Fig. A7-1 we see that this sets the carry bit. Hence it is equivalent in execution to the 6800's SEC set carry instruction. The designers of the 6809 thus took these infrequently used instructions and redefined their coding so as to free up some operation codes for more significant use.

The unconditional jump and jump to subroutine instructions, JMP and JSR, can employ any of the indexed family of addressing modes. For example, it is now possible, with one instruction, to execute an indirect jump or an indirect jump to a subroutine through a vector taken from a table. Thus, if X contains the base address of a table of subroutine addresses, then

JSR [B,X]

will jump to the subroutine whose address is stored in the table at X + B and X + B + 1.

Each of the conditional and unconditional branch instructions now has two forms of addressing. A short branch is the 6800's two-byte instruction for branching up to (approximately) ± 128 locations from the present location. A long unconditional (conditional) branch will branch anywhere, but requires three (four) bytes. The long conditional branches take six clock cycles to execute when the branch is taken but only five cycles otherwise.

The new SYNC instruction is used to provide software synchronization with an external hardware process. The CPU will remain in the *stopped state until an interrupt is received.* If that interrupt is enabled, the CPU will handle the interrupt just as it would if it were not in the stopped state. Upon return from the interrupt service routine, the stopped state has been cleared and the sequencing of instruction continues. On the other hand, if the interrupt is disabled, the stopped state is simply cleared and the sequencing of instructions continues **without** vectoring to the interrupt service routine. For example, consider the following loop for reading a byte of data into an array from a device each time the device causes an "interrupt" input — with that interrupt source disabled:

```
FAST  SYNC             Wait for interrupt
      LDA    ,X        Get data from input port
      STA    ,Y+       Store data and increment
                       address
      DECB             Are we done?
      BNE    FAST      If not, do it again
```

Augmenting the 6800's SWI software interrupt instruction, the 6809 supports two new software interrupts, SWI2, and SWI3. Like SWI, they put all of the CPU registers on the S stack, disable IRQ and FIRQ interrupts, and then vector through an address on page FF to a service routine. With this and the FIRQ (fast interrupt request) vectors, the memory map for interrupt vectors corresponding to Fig. A4-5 has three new entries, beginning at address FFF2. The following 6809 assembly language sequence assigns these vectors:

```
ORG    $FFF2
FDB    SWI3
FDB    SWI2
FDB    FIRQ
FDB    IRQ
FDB    SWI
FDB    NMI
FDB    RESET
```

where FDB (form double byte) is Motorola's equivalent of the DW (define word) assembler directive of Fig. 2-32. For this sequence to be assembled correctly, each of the service routines must be given the label listed above. For example, the fast interrupt request service routine must be labeled FIRQ.

The 6809 supports the use of block-structured, high-level languages. The availability of a user stack and the addressing of operands within either stack are some of the capabilities which help a compiler of a high-level language to run efficiently on the 6809. In fact, the two index registers, X and Y, can also be employed as stack pointers. Thus,

```
STA    ,—X
```

uses the X index register as a stack pointer and pushes the contents of accumulator A onto this stack. Likewise,

```
STB    ,—X
STD    ,——X
STY    ,——X
STU    ,——X
STS    ,——X
```

can be used to push other registers onto an "X" stack. The correspondong "pulls" from the "X" stack are

```
LDA    ,X+
LDB    ,X+
LDD    ,X++
LDY    ,X++
LDU    ,X++
LDS    ,X++
```

Of course, in actual use we would pull registers in the opposite order from that used to push them. Also, we would not tend to stack A and B and then stack D also.

It may be interesting to note that only the following instructions require a two-byte operation code:

```
          conditional long branches

          SWI2   SWI3
CMPD      CMPY   CMPS      CMPU
          LDY    LDS
          STY    STS
```

The first byte, 10 or 11 (hex), tells the CPU to look at the second byte to decode which of these instructions it is dealing with. From the extensive 6809 instruction set, the designers have carefully picked these to be the inefficiently encoded instructions since they are least likely to see much use in the design of dedicated instruments and devices (having relatively small programs stored in ROM). All of the other instructions, and all of the various addressing modes, have been given efficient encodings for this environment. At the same time, the inefficiently encoded instructions listed above give the 6809 useful capabilities for a large-system, floppy-disk-based environment. Here we hold little concern for the efficiency of instruction encoding because of the large amount of memory available. This contrasts sharply with the Zilog Z80 instructions which augment the Intel 8080 instruction set by using only operation codes unused by the 8080. Thus, a Z80 instruction employing indexed addressing might be expected to be a two-byte instruction (with one byte for the operation code and one byte for the index). Actually, all Z80 instructions employing indexed addressing are either three-byte or four-byte instructions.

The indexed addressing options of Fig. A7-3 are THE exciting new feature of the 6809, especially in light of their applicability to **thirty-one** of the instructions listed in Fig. A7-2. Once a pointer is set up, it can be used with a fixed or variable offset or it can be automatically incremented or decremented. Furthermore, the contents of the address so identified can either be used as an operand or as the **address** of an operand. Note also that for users who have no interest in setting up a separate user stack involving the U stack pointer, there are really **three** general pointers available (X, Y, and U) simultaneously for accessing operands.

An intriguing capability of the 6809 has been afforded simply by giving the hardware stack pointer S all of the same indexing capability as any of the other pointers. For example, the two byte instruction

```
LEAS   -8,S
```

when used as the first instruction of a subroutine subtracts eight from the stack pointer. This sets up a temporary workspace on the stack consisting of eight locations which are addressed within the subroutine using instructions with operands identified as 0,S through 7,S. The beauty of this method of creating storage locations for temporary variables within a subroutine is that we know that these locations can be freely used by the subroutine without disturbing any other variables in our software. As we terminate the subroutine with the two-instruction sequence

```
LEAS    8,S
RTS
```

we relinquish the eight storage locations making up the workspace.

The 6809 has extensive instructions for operating upon pointers, in addition to the indexed family of addressing modes. Just as one example, the multiple-entry-point, table-driver subroutine of Fig. 2-27 can be easily implemented in just five bytes of ROM as follows, if X is first loaded with the base address of a table and accumulator B with the desired offset into it.

```
TBL4    ABX     Add B to X
TBL3    ABX     Add B to X
TBL2    ABX     Add B to X
TBL1    ABX     Add B to X
        RTS     Return
```

Even if we want to handle sixteen-bit offsets held in D (the double accumulator), we need only replace the ABX instructions with LEAX D,X instructions to achieve the same results, but now with a nine-byte subroutine.

As stated earlier, one of the unusual, and unusually powerful, features of the 6809 is the full support it gives to the preparation of **position-independent** code. It does this in five ways:

1. Position-independent control transfer — with long and short relative branches.

2. Position-independent temporary storage — using workspace on the stack.

3. Position-independent access to constants located within the position-independent code — using the indexed addressing mode exemplified by LDA CONST,PCR.

4. Position-independent access to tables located within the position-independent code — using the load effective address instruction LEAX TABLE,PCR to load the base address of a table into X, followed by indexed addressing into the table.

5. Position-independent access to constants located in ROM outside of the position-independent code as well as access to global variables located in RAM, whose addresses are not known at the time the position-independent code is assembled. This is vital for dedicated applications like instrument design which involve hardware-support software developed in a ROM for "mass use" by many instrument designers for a variety of projects. For example, a printer manufacturer might support the printer with a ROM prepared for the 6809 which contains a subroutine to drive the printer. However, the subroutine may require that a variety of parameters be passed to it. If these parameters are constants (e.g., I/O port addresses, a number used to derive the printer timing from the microcomputer's clock rate, etc.) or global variables (e.g., a "BUSY" state variable), then the 6809 can support this by permitting one pointer (i.e., X, or Y, or U) to be initialized prior to any calls to subroutines in the "mass-use" ROM. By organizing these parameters with a table and initializing the pointer to the base address of the table, the parameters can be picked up within the subroutines using the indexed addressing modes. For example, the "mass-use" ROM might require that we create the following table:

```
RTBL  FDB  $1400  Full address of output port 1
      FDB  $1402  Full address of output port 2
      FCB  25     8-bit number to scale the clock
                  rate
      FDB  BUSY   Full address of the global
                  variable BUSY
```

Then a call to a PRINT subroutine in the "mass-use" ROM might take the form:

```
LDX   #RTBL     Initialize X to point to RTBL
JSR   PRINT
```

The PRINT subroutine can load accumulator A with the 8-bit number to scale the clock rate with

```
LDA   4,X
```

It can write to output port 2 with

```
STA   [2,X]
```

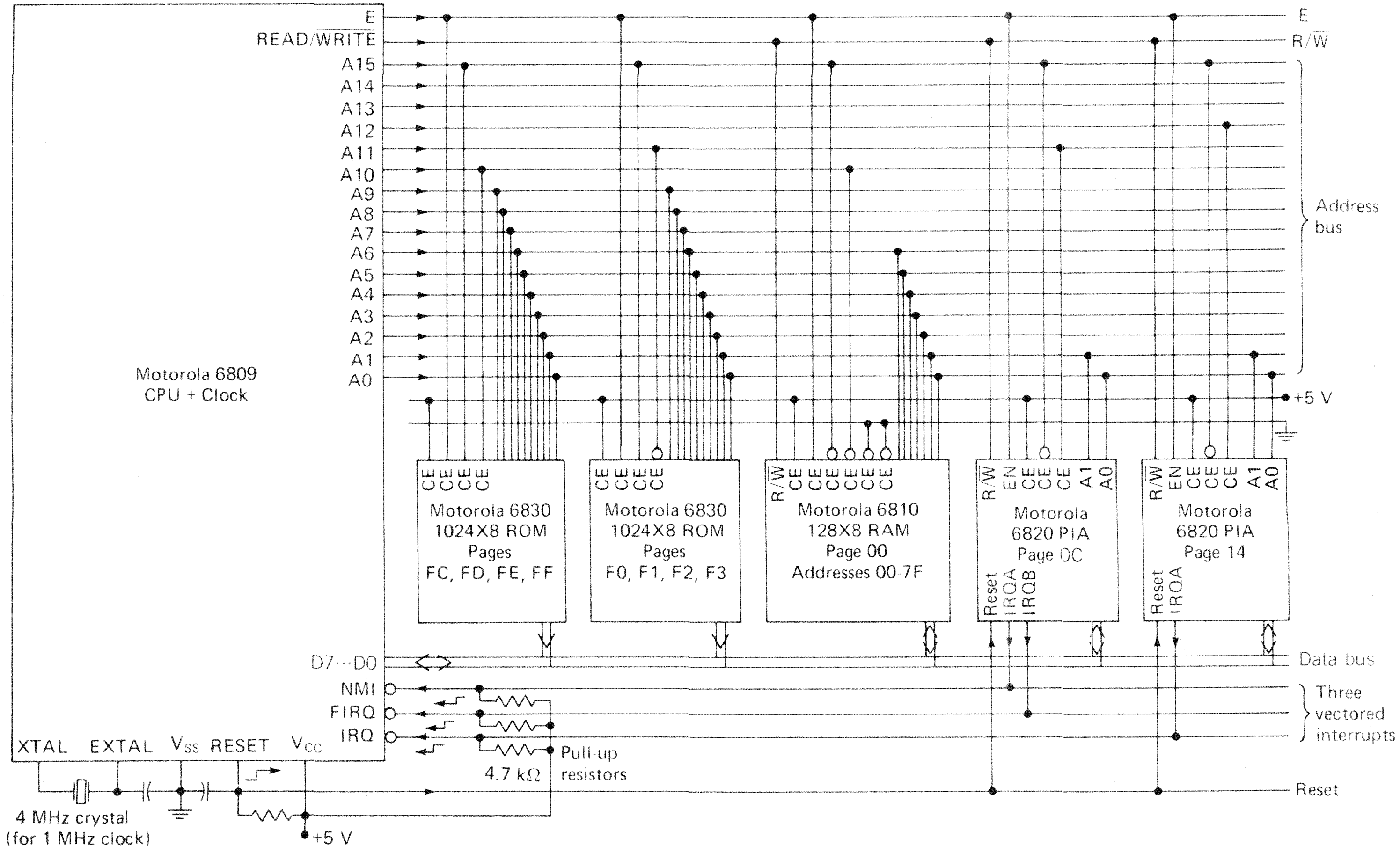
It can read the global RAM variable labeled BUSY with

```
LDA   [5,X]
```

In concluding this appendix, we repeat, in Fig. A7-5, the microcomputer configuration of Fig. A4-9 as it would be implemented with the 6809. Note that the elimination of VMA (Valid Memory Address) frees a chip enable on all chips. This permits significantly larger systems to be implemented without page decoders. In fact, satisfactory performance can be achieved at the maximum specified clock rate with up to eight 68XX chips plus one TTL load hung on the address bus/data bus structure. Even more 68XX chips can be employed, without buffering, if the clock rate is reduced so as to reduce the effect of capacitive loading.

The threshold of the reset input on the 6809 CPU chip has been designed to be higher than the threshold of the reset input on the 68XX peripheral chips (e.g., the 6820 PIA chip). Consequently, when the 6809 starts up and initializes the PIA, we can be assured that the PIA is no longer being reset. Within the 6809 itself, resetting clears the I and F bits, disabling maskable interrupts. Resetting also clears the direct page register and then causes the CPU to vector through addresses FFFE and FFFF to the address of the first instruction.

Figure A7-5: Motorola 6809 Microcomputer Configuration

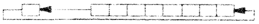
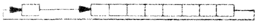
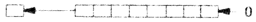
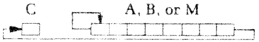
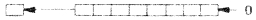
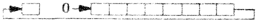


New Figure A7-2: Motorola 6809 Instruction Set

				Bytes/Cycles						Flags				
Operation	**MNE**	**REG**	**Description**	Inherent	Immediate	Direct (Page zero)	Full	Full indirect	Indexed family	H	C	Z	N	V
Move, 8-bit	TFR	R, R′	R′ ← R } for R, R′ = A, B, CC, DP	2/6						—	—	—	—	—
	EXG	R, R′	R′ ↔ R }	2/7						—	—	—	—	—
	CLR	A(B)	A(B) ← 0	1/2						—	0	1	0	0
	CLR		M ← 0			2/6	3/7	4/11	2+/6+	—	0	1	0	0
	LD	A(B)	A(B) ← M		2/2	2/4	3/5	4/9	2+/4+	—	—	↕	↕	0
	ST	A(B)	M ← A(B)			2/4	3/5	4/9	2+/4+	—	—	↕	↕	0
Move, 16-bit	TFR	R, R′	R′ ← R } for R, R′ = D, X, Y, U, S, PC	2/6						—	—	—	—	—
	EXG	R, R′	R′ ↔ R }	2/7						—	—	—	—	—
	LD	R	R ← M for R = D, X, U		3/3	2/5	3/6	4/10	2+/5+	—	—	↕	↕	0
			for R = Y, S		4/4	3/6	4/7	5/11	3+/6+	—	—	↕	↕	0
	ST	R	M ← R for R = D, X, U			2/5	3/6	4/10	2+/5+	—	—	↕	↕	0
			for R = Y, S			3/6	4/7	5/11	3+/6+	—	—	↕	↕	0
Load effective address	LEA	R	R ← Address of M for R = X, Y					4/9	2+/4+	—	—	↕	—	—
			for R = U, S					4/9	2+/4+	—	—	—	—	—
	PSHS	list	S stack ← List of desired CPU registers	2 (5 + b)						—	—	—	—	—
	PSHU	list	U stack ← List of desired CPU registers	(b = number						—	—	—	—	—
	PULS	list	List of desired CPU registers ← S stack	of bytes								(CC)		
	PULU	list	List of desired CPU registers ← U stack	moved)								(CC)		
Increment	INC	A(B)	A ← A + 1 (B ← B + 1)	1/2						—	—	↕	↕	↕
	INC		M ← M + 1			2/6	3/7	4/11	2+/6+	—	—	↕	↕	↕
Decrement	DEC	A(B)	A ← A − 1 (B ← B − 1)	1/2						—	—	↕	↕	↕
	DEC		M ← M − 1			2/6	3/7	4/11	2+/6+	—	—	↕	↕	↕
Complement	COM	A(B)	A ← $\overline{A}$ (B ← $\overline{B}$)	1/2						—	1	↕	↕	0
	COM		M ← $\overline{M}$			2/6	3/7	4/11	2+/6+	—	1	↕	↕	0
Set selected CC bits	OR	CC	CC ← CC OR M		2/3					↕	↕	↕	↕	↕
Clear selected CC bits	AND	CC	CC ← CC AND M		2/3					↕	↕	↕	↕	↕

(continued)

New Figure A7-2: Motorola 6809 Instruction Set (continued)

				← Bytes/Cycles →						Flags				
Operation	**MNE**	**REG**	**Description**	Inherent	Immediate	Direct (Page zero)	Full	Full indirect	Indexed family	H	C	Z	N	V
Add, 8-bit	ADD	A(B)	A ←— A + M (B ←— B + M)		2/2	2/4	3/5	4/9	2 + 4 +	↕	↕	↕	↕	↕
Add with carry, 8-bit	ADC	A(B)	A ←— A + M + C (B ←— B + M + C)		2/2	2/4	3/5	4/9	2+/4+	↕	↕	↕	↕	↕
Decimal adjust acc. A	DAA		Correct BCD addition	1/2						—	↕	↕	↕	↕
Add, 16-bit	ADD	D	D ←— D + M		3/4	2/6	3/7	4/11	2+/6+	↕	↕	↕	↕	↕
	ABX		X ←— X + B (unsigned addition of B to X)	1/3						—	—	—	—	—
Sign extend	SEX		A ←— FF if bit 7 of B is set 00 if bit 7 of B is clear	1/2						—	—	↕	↕	0
Subtract, 8-bit	SUB	A(B)	A ←— A − M (B ←— B − M)		2/2	2/4	3/5	4/9	2+/4+	*	↕	↕	↕	↕
Subtract with carry, 8-bit	SBC	A(B)	A ←— A − M − C (B ←— B − M − C)		2/2	2/4	3/5	4/9	2+/4+	*	↕	↕	↕	↕
Negate, 8-bit	NEG	A(B)	A ←— 00 − A (B ←— 00 − B)	1/2						—	↕	↕	↕	↕
	NEG		M ←— 00 − M			2/6	3/7	4/11	2+/6+	—	↕	↕	↕	↕
Subtract, 16-bit	SUB	D	D ←— D − M		3/4	2/6	3/7	4/11	2+/6+	—	↕	↕	↕	↕
Multiply	MUL		D ←— A*B (unsigned)	1/11						—	b_7	↕	↕	↕
AND	AND	A(B)	A ←— A AND M (B ←— B AND M)		2/2	2/4	3/5	4/9	2+/4+	—	—	↕	↕	0
Exclusive-OR	EOR	A(B)	A ←— A ⊕ M (B ←— B ⊕ M)		2/2	2/4	3/5	4/9	2+/4+	—	—	↕	↕	0
OR (inclusive)	OR	A(B)	A ←— A OR M (B ←— B OR M)		2/2	2/4	3/5	4/9	2+/4+	—	—	↕	↕	0
Compare, 8-bit	CMP	A(B)	A − M (B − M)		2/2	2/4	3/5	4/9	2+/4+	*	↕	↕	↕	↕
	TST	A(B)	A − 00 (B − 00)	1/2						—	—	↕	↕	0
	TST		M − 00 } Only			2/6	3/7	4/11	2+ 6+	—	—	↕	↕	0
	BIT	A(B)	A AND M (B AND M) } flags are		2/2	2/4	3/5	4/9	2+/4+	—	—	↕	↕	0
Compare, 16-bit	CMP	R	R-M for R = X } affected		3/4	2/6	3/7	4/11	2+/6+	—	↕	↕	↕	↕
			for R = D, Y, U, S		4/5	3/7	4/8	5/12	3+/7+	—	↕	↕	↕	↕
Rotate left	ROL	A(B)	C A, B, or M	1/2						—	↕	↕	↕	↕
	ROL					2/6	3/7	4/11	2+/6+	—	↕	↕	↕	↕
Rotate right	ROR	A(B)	C A, B, or M	1/2						—	↕	↕	↕	—
	ROR					2/6	3/7	4/11	2+/6+	—	↕	↕	↕	—
Arithmetic shift left	ASL	A(B)	C A, B, or M	1/2						—	↕	↕	↕	↕
	ASL					2/6	3/7	4/11	2+/6+	—	↕	↕	↕	↕
Arithmetic shift right	ASR	A(B)	C A, B, or M	1/2						—	↕	↕	↕	—
	ASR					2/6	3/7	4/11	2+/6+	—	↕	↕	↕	—
Logic shift left	LSL	A(B)	C A, B, or M	1/2						—	↕	↕	↕	↕
	LSL					2/6	3/7	4/11	2+/6+	—	↕	↕	↕	↕
Logic shift right	LSR	A(B)	C A, B, or M	1/2						—	↕	↕	0	—
	LSR		0 			2/6	3/7	4/11	2+/6+	—	↕	↕	0	—
No operation	NOP		PC ←— PC + 1	1/2						—	—	—	—	—

* Undefined

New Figure A7-2: Motorola 6809 Instruction Set (continued)

					← Bytes/Cycles →							
Operation		MNE	Description		Inherent	Short relative	Long relative	Direct (Page zero)	Full	Full indirect	Indexed family	Flags H C Z N V
ump unconditionally		JMP	PC ← M					2/3	3/4	4/8	2+/3+	— — — — —
Branch unconditionally		BRA	PC ← PC + M			2/3	3/5					— — — — —
if carry set		BCS	if C = 1			2/3	4/6(5)					— — — — —
if carry clear		BCC	if C = 0			2/3	4/6(5)					— — — — —
if equal zero		BEQ	if Z = 1			2/3	4/6(5)					— — — — —
if not equal zero		BNE	if Z = 0			2/3	4/6(5)					— — — — —
if minus		BMI	if N = 1			2/3	4/6(5)					— — — — —
if plus		BPL	if N = 0			2/3	4/6(5)					— — — — —
if overflow set		BVS	if V = 1	Use LBRA, LBCS,		2/3	4/6(5)					— — — — —
if overflow clear		BVC	if V = 0	LBCC, etc. as the		2/3	4/6(5)					— — — — —
if > zero		BGT	if Z + (N ⊕ V) = 0	mnemonic if the		2/3	4/6(5)					— — — — —
if ≥ zero	Signed	BGE	if N ⊕ V = 0	long relative		2/3	4/6(5)					— — — — —
if ≤ zero	numbers	BLE	if Z + (N ⊕ V) = 1	branch is desired		2/3	4/6(5)					— — — — —
if < zero		BLT	if N ⊕ V = 1			2/3	4/6(5)					— — — — —
if higher		BHI	if C or Z = 0			2/3	4/6(5)					— — — — —
if higher or same	Unsigned	BHS	if C = 0			2/3	4/6(5)					— — — — —
if lower or same	numbers	BLS	if C or Z = 1			2/3	4/6(5)					— — — — —
if lower		BLO	if C = 1			2/3	4/6(5)					— — — — —
NEVER		BRN	PC ← PC + 2 (3)			2/3	4/5					— — — — —
Jump to subroutine		JSR	S stack ← PC; PC ← M					2/7	3/8	4/12	2+/7+	— — — — —
Branch to subroutine		BSR	S stack ← PC; PC ← PC + M (or use LBSR)			2/7	3/9					— — — — —
Return from subroutine		RTS	PC ← S stack		1/5							— — — — —

			← Bytes/Cycles →		Flags
Operation	MNE	Description	Inherent	Immediate	H C Z N V
Software interrupt	SWI	S stack ← PC, U, Y, X, DP, B, A, CC; Set I, F; PC ← M(FFFA, FFFB)	1/19		— — — — —
	SWI2	S stack ← PC, U, Y, X, DP, B, A, CC; Set I, F; PC ← M(FFF4, FFF5)	2/20		— — — — —
	SWI3	S stack ← PC, U, Y, X, DP, B, A, CC; Set I, F; PC ← M(FFF2, FFF3)	2/20		— — — — —
Clear and wait	CWAI	CC ← CC AND M; S stack ← PC, U, Y, X, DP, B, A, CC; Halt		2/20	↕ ↕ ↕ ↕ ↕
Synchronize I/O	SYNC	Halt; Continue when interrupt occurs (see text)	1/2		— — — — —
Return from interrupt	RTI	CC ← S stack; If E is set, then A, B, DP, X, Y, U, PC ← S stack If E is cleared, then PC ← S stack	1/15(6)		↕ ↕ ↕ ↕ ↕

New Figure A7-3: Indexed Addressing Options

Indexed addressing form		Source code form	Description (where R = X, Y, U, or S)	Additional bytes	Additional cycles
Non-indirect addressing	No offset, no change in R	,R	Get operand pointed to by R	0	0
	No offset, change R	, R+	Get operand and then increment R once	0	+2
		, R++	Get operand and then increment R twice	0	+3
		, −R	Decrement R once and then get operand	0	+2
		, −−R	Decrement R twice and then get operand	0	+3
	Fixed offset	offset, R	Temporarily add signed offset (expressed as sign plus magnitude) to R and then get operand, for −16 ≤ offset ≤ +15	0	+1
			−128 ≤ offset ≤ +127	+1	+1
			−65536 ≤ offset ≤ +65536	+2	+4
	Variable offset	A, R	Temporarily add signed offset (held in accumulator in 2s-complement form) to R and then get operand	0	+1
		B, R		0	+1
		D, R		0	+4
	Relative addressing	label, PCR	Use relative addressing to access a constant having a label − and located within ±128 bytes of the next instruction	+1	+1
			− and located anywhere	+2	+5
		offset, PC	Temporarily add signed offset (expressed as sign plus magnitude) to the address of the next instruction and then get operand, for −128 ≤ offset ≤ +127	+1	+1
			−65536 ≤ offset ≤ +65536	+2	+5
Indirect addressing Use operand found as above as a pointer to the actual operand	No offset, no change in R	[, R]		0	+3
	No offset, change R	[, R++]		0	+6
		[, −−R]		0	+6
	Fixed offset	[offset, R]	−128 ≤ offset ≤ +127	+1	+4
			−65536 ≤ offset ≤ +65536	+2	+7
	Variable offset	[A, R]		0	+4
		[B, R]		0	+4
		[D, R]		0	+7
	Relative addressing	[label, PCR]	For label located within ±128 bytes of the next instruction	+1	+4
			For label located anywhere	+2	+8
		[offset, PC]	−128 ≤ offset ≤ +127	+1	+4
			−65536 ≤ offset ≤ +65536	+2	+8

MOTOROLA Semiconductor Products Inc.
P.O. BOX 20912 • PHOENIX, ARIZONA 85036 • A SUBSIDIARY OF MOTOROLA INC.

MOTOROLA

Product Preview
MICROSYSTEMS

MEX6809-1 or MEX6809-2 MC6809 EXORciser Support Package

- USE Function Included on MC6809 MPU Module
- EXORciser/EXORterm Compatible
- 1.0, 1.5 or 2.0 MHz Operation
- EXbug 2 DEbug Capability

A key feature of Motorola'a total microprocessor support is the use of the popular EXORciser/EXORterm Development Systems as a basis for new support tools. The main support for the 6809 follows this guideline.

The 6809 support package will enable 6809 users to evaluate their design as well as the real-time capability of the MC6809 microprocessor. The support package combines the 6809 MPU and USE functions:

a) The module will provide the timing and microprocessing unit for both the EXORciser or EXORterm and the user's prototype system.

b) The module will interface the MC6809 Microprocessing Unit with the EXORciser bus.

c) The module will be capable of emulating the user's 6809 via a 40-pin dip socket to be plugged into the user's 6809 socket in his target system.

The 6809 MPU/USE Module will be compatible with any EXORciser I and II modules (DEbug I module, excluded), currently operating with an MPU module.

The microprocessor function will be implemented with a MC6809 in order to support 1.0, 1.5, and 2.0 MHz operation. These three basic frequencies will be available on the board.

The software portion of this phase of the 6809 support will include the following items:

a) Resident Macro Assembler

b) Resident Editor

c) Resident Disk Operating System (MDOS), including firmware

d) Resident PASCAL Compiler

e) Resident MPL Compiler

With the 6809 MPU/USE module properly installed in an EXORciser the user can power-up the system, enter MDOS, edit and assemble source code, and debug the resulting 6809 object code. The User System Evaluator (USE) function will be an integral part of the support package. This function will allow a real-time evaluation of user's system. This feature will permit the user to debug his system hardware once the software appears to be functioning properly. Furthermore, the software debug features are extended to the user's system.

Minimum hardware system requirements are:

- EXORciser or EXORterm (DEbug II module with 6809 EXbug firmware and 6809 MPU/USE Module)
- 24K Memory
- EXORdisk II with 6809 MDOS and firmware
- Terminal (not necessary with EXORterm)

This is advance information and specifications are subject to change without notice.

ISSUE A

Further 6809 Support

Within 3 months after the 6809 MPU/USE module is available, the 6809 version of the present System Analyzer firmware and PROM Programmer software will be available for delivery.

The following additional high level languages are planned for introduction in the 3rd and 4th quarter 1979:

1. Resident FORTRAN Compiler for 6809
2. Resident COBOL Compiler for 6809
3. Resident BASIC Interpreter for 6809

Ordering Information

Part Number	Description
MEX6809-1	6809 Upgrade package for EXORciser I/EXORterm 200 consisting of MEX6809 MPU/USE Module, DEbug II Module with 6809 EXbug firmware, Resident Macro Assembler and Editor, MDOS software on diskette and MDOS controller module firmware.
MEX6809-2	6809 Upgrade package for EXORciser II/EXORterm 220 consisting of 6809 MPU/USE module, 6809 EXbug firmware, Resident Macro Assembler and Editor, MDOS software on diskette and MDOS controller module firmware.
M6809MPL	Resident MPL compiler on MDOS diskette. Requires 56K of RAM.
M6809PASCAL	Resident PASCAL compiler on MDOS diskette. Requires 56K of RAM.

Availability: 2nd Quarter, 1979

Motorola reserves the right to make changes to any products herein to improve reliability, function or design. Motorola does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others.

MOTOROLA *Semiconductor Products Inc.*

COMPILATION AND PASCAL ON THE NEW MICROPROCESSORS

Prepared By

Charles H. Forsyth and Randall J. Howard

Computer Communications Networks Group
University of Waterloo
Waterloo, Ontario
CANADA N2L 3G1

MOTOROLA
Semiconductor Products Inc.

COMPILATION AND PASCAL ON THE NEW MICROPROCESSORS

Prepared By

Charles H. Forsyth and Randall J. Howard

Computer Communications Networks Group
University of Waterloo
Waterloo, Ontario
CANADA N2L 3G1

We are concerned with the use of high level languages, and in particular Pascal, on microcomputer systems. We are most interested in the use of such languages for what is termed, on larger computer systems, *systems programming*. This includes writing code to drive floppy disks, interpreters for APL or BASIC, or all those bits of code that people have until now written in assembler, and which in some way make their microcomputer systems friendly.

Microcomputer users show a generally high level of sophistication, so it might be surprising at first that so much of their code is still written in assembler. The advantages of writing in a high level language have been often described in computing literature: programs can be made more portable; they exhibit better structure; and they are easier to write and debug. In addition, it is much easier to let a compiler worry about the efficiency of the object code; and deficiencies of the object machine are hidden. With the 8 bit microcomputers like the Intel 8080 and Motorola 6800, we feel that there is little choice but to write in assembler (or interpreter), since the facilities provided by their order codes are simply insufficient to support most high level languages.

Compilation may be inappropriate for 8 bit microcomputers, but it is the most attractive alternative for the hybrid 8 and 16 bit microcomputers (such as the Motorola 6809), especially with respect to eliminating most assembly code on these machines. We also feel that Pascal has facilities that enable a compiler to generate better code for such machines than might be expected from compilers for other languages.

Jensen and Wirth provide the definition of and tutorial introduction to Pascal in the *Pascal User Manual and Report.* Aho and Ullman's book, *Principles of Compiler Construction,* provides an excellent description of the elements of a compiler.

Options

Tiny BASIC, Tiny C, APL, and FOCAL are implemented on microcomputers with interpretive code. Interpretation has a number of advantages. Since the interpretive language is highly specialized, it can be made compact. New *macro operations* can be added easily as time and experience dictate. Array and structure addressing and the block copying associated with array and structure assignment may be made particularly cheap. When interpreting array indexing, run time checks of the index values against the array bounds are possible (although often left out) at little extra cost. This is true of other kinds of debugging facilities as well, such as value traces or stack tracebacks. Both compiler and interpreter are easy to write, especially if the interpreted code implements a stack machine. Interpretation's main disadvantage is that it is slow.

An alternative to interpretation that alleviates this latter problem of speed somewhat is *threaded code,* which has been described as "interpretive code which needs no interpreter" (see references 2 and 3). Rather than having a sequence of codes and an interpreter which reads them, calling out to the routines implementing each operation, threaded code simply contains the sequence of machine addresses of the routines to process each operation. These routines, much like the code segments called by the inter-

The listings in this article were prepared by arrangement with Walter Banks of the University of Waterloo.

```
type
        index = 0..10;
        twiceIndex = 0..20;
        unsigned = 0..32767;
        short = -128..127;
        shortUnsigned = 0..255;
        thing = record
                        field1: 0..7;
                        field2: 0..31
                end;
        packedThing = packed record
                        field1: 0..7;
                        field2: 0..31;
                end;
var
        a, b: array [index] of integer;
        i, j: index,
        k: twiceIndex;

        s: set of (READY, BLOCKED, RUNNING, SWAPIN, SWAPOUT);
begin
        a[i] := b[j];       {the dreaded array-indexing example}

        k := i+j;                   {subranges are useful}

        s := [READY, BLOCKED, RUNNING];{set operations}
        s := s - [READY, RUNNING];
        s := s + [SWAPIN];
        s := s * [SWAPIN, BLOCKED];
end
```

Listing 1: Pascal program fragment for array indexing.

```
tsx                     /Enable indexing off sp
lda     A, j(X)         /Fetch address of j relative..
lda     B, j+1(X)       /to sp into (A,B) register pair
asl     B               /Shift (AB) pair left by 1..
rol     A               /yielding integer offset
add     B, b+1(X)       /Add in 16-bit array
adc     A, b(X)         /pointer i to (A,B) pair
sta     A, temp         /Transfer (A,B) pair to X reg..
sta     B, temp+1       /..not re-entrant
ldx     temp
lda     A, 0(X)         /Finally, fetch b[j] into..
lda     B, 1(X)         /(A,B) pair..
psh     A               /and push onto stack
psh     B
tsx                     /Following code is repeat of..
lda     A, i(X)         /above for getting address of..
lda     B, i+1(X)       /array element a[i]
asl     B
rol     A
add     B, a+1(X)
adc     A, a(X)
sta     A, temp
sta     B, temp+1
ldx     temp            /X now points at a[i]
pul     B               /Pop b[j] from stack..
pul     A               /into (A,B) pair..
sta     A, 0(X)         /and store in a[i]
sta     B, 1(X)

Total code: 52 bytes
```

Listing 2: Motorola 6800 assembly code for the first line of the Pascal fragment shown in listing 1.

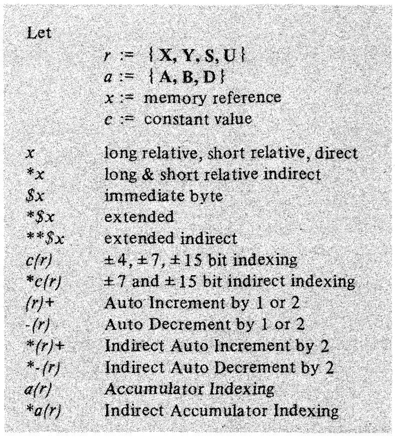
preter to implement the pseudo-machine, provide the run time support for the threaded code. Rather than return to an interpreter after it has done its work, though, a routine simply jumps (indirectly) to the next such routine in the code flow. Arguments are passed to these routines in various ways – for example, by placing values or addresses between the code pointers.

The third approach to language implementation is that traditionally adopted on larger machines: real code generation. This approach provides the fastest program execution at the possible expense of space used by the object code. On almost any machine, the high level constructs of flow of control and logical expressions as well as calls to the intrinsic built-in functions can be directly implemented as branch or jump instructions with relatively little expenditure of speed or time. However, for many of the existing microcomputers, code generation for even the simplest of the fundamental high level language constructs proves effectively impossible. Such constructs include most common arithmetic operations, array and structure accessing, and automatic storage manipulation. Particularly difficult on some machines are multiply, divide, modulus and string operations. Therefore it is important to determine what properties of a particular machine make it suitable for real code generation.

8 Bit Microcomputers

A detailed study of the common 8 bit computers available today (eg: Motorola 6800, Intel 8080) quickly reveals that such machines are not conducive to real code generation by compilers for high level languages such as Pascal.

On such machines, compilations of even the simplest arithmetic or pointer expressions lead to a very high object to source code ratio, if such constructs can be compiled at all. Listing 2 gives an example of code which might be compiled for a Motorola 6800 to implement the Pascal assignment statement: *a[i]:=b[j];* in listing 1. The assumption here is that automatic arrays are implemented as pointers on the stack to areas of storage residing elsewhere. In addition, we have assumed that the compiler keeps track of the stack offsets for its automatic variables relative to the moving stack pointer; we are using the notation *j* to represent the stack offset of variable j. In addition to this code segment, the procedure preamble must set up the pointers to the arrays a and b (stored at offsets *a* and *b* respectively), to point at the integer before the beginning of the array. Thus, for example, *a[1]* will then be identi-



Let

r := {**X, Y, S, U**}
a := {**A, B, D**}
x := memory reference
c := constant value

x	long relative, short relative, direct
*x	long & short relative indirect
$x	immediate byte
*$x	extended
**$x	extended indirect
c(r)	±4, ±7, ±15 bit indexing
*c(r)	±7 and ±15 bit indirect indexing
(r)+	Auto Increment by 1 or 2
-(r)	Auto Decrement by 1 or 2
*(r)+	Indirect Auto Increment by 2
*-(r)	Indirect Auto Decrement by 2
a(r)	Accumulator Indexing
*a(r)	Indirect Accumulator Indexing

Table 1: A summary of the Motorola MC6809 addressing modes.

fied with the beginning of the storage associated with the array a.

Beyond the actual code shown here, however, the most important insight to be gained from all of this is the sheer bulk of code that such a simple construct would generate (and it is not even reentrant at that). Imagine how large the object code size would be for even a reasonably short Pascal program.

Implementing threaded code is somewhat difficult on these machines because they require 16 bit memory pointers, an efficient mechanism for indirect addressing, and some method of incrementing such a pointer to the next 16 bit pointer. At least one of the above criteria is so troublesome on both the Motorola 6800 and the Intel 8080 that the threaded code becomes unwieldy. Thus, for these machines one has little choice but to interpret or write in assembler. This suggests that the interpreters themselves must be implemented in assembly language.

The above discussion is an attempt to analyze the reasons why programs written for 8 bit microcomputers have traditionally been interpreted or written in assembly or machine code, rather than being compiled into "true" code from a high level language.

16 Bit Microcomputers

Previously, the only alternative to the 8 bit architecture was that of the 16 bit microcomputer. Examples of such machines include the TI-990/4 and the DEC LSI-11. While the considerable costs of these processors tend to make them impractical for many computer experimenters, and for those applications in which many processors are required, it is instructive to consider what properties set these machines apart from their 8 bit counterparts with respect to code generation. In fact, it can be shown that, given a machine of sufficient sophistication, it should be possible for a compiler to do as good a job as an assembler programmer vis-à-vis machine resource utilization.

There are two main virtues of these 16 bit machines. In the first place, these machines have complete 16 bit instruction repertoires *including hardware multiplication, division,* and long shifts. As well, the 16 bit processors tend to have a good complement of addressing modes such as indexing, stack operations, automatic increment and decrement of pointers, and so on. (Here, as elsewhere in this article, the descriptive terms may seem fuzzy. *Good complement* does not admit of a precise meaning. With real machines, one usually loses clever addressing modes, for plenty of general purpose registers, and one must balance the benefits somehow. The final judgment will usually be that of the person writing the compiler.) With these attributes, it is a fairly straightforward task to construct a compiler for a high level language such as Pascal.

8 and 16 Bit Hybrids

The current trend in 8 bit microprocessor technology is towards a hybrid combination 8 and 16 bit machine. Essentially, these processors are capable of 16 bit operations while retaining 8 bit data paths throughout the processor architecture. A prime example of such a hybrid is the Motorola 6809, which is due for formal product release later this year. Table 1 gives a summary of the basic addressing capabilities of the Motorola 6809, expressed in a hypothetical assembler syntax which removes from the user the burden of understanding all of the details of the actual hardware addressing modes.

What advantages do these machines have over their pure 8 bit predecessors? In particular, these machines now have at least one accumulator for performing addition, subtraction, shifting and comparison operations on 16 bit data. A second feature of these machines is the 16 bit memory pointer, which, combined with the ability to automatically increment and decrement such pointers, provides a very general memory accessing capability. In addition, common high level language features such as stack frames and display pointers become quite easy with the general index and stack registers of the M6809. It is apparent that the Motorola 6809 is particularly well-endowed with addressing modes which tend to facilitate code generation for high level languages.

Consider again the array assignment which the 6800 handled so dismally. The Motorola 6809 code for the same construct is given in listing 3. (Note that the syntax of our assembler code is intended to be more or less consistent amongst the examples, and not necessarily that of the manufacturer's assembler. It is in fact the syntax used by our UNIX assemblers for these machines.) Code for the PDP-11/45, considered to be a good instruction set given in listing 4, is included for comparison.

It is rather precipitous to deduce much from this one example, although array indexing does exercise many of the addressing modes of a machine, and such assignment statements can provide a check on the register usage of a compiler. How a particular architecture fares with more general arithmetic expressions and function and procedure call, save, and return sequences would provide further basis of comparison. Indeed, other examples that we have tried suggest that the results of this comparison are typical.

Special Advantages of Pascal

We feel that the use of Pascal and a competent compiler can lead to better code in many cases on hybrid 8 and 16 bit machines than can be achieved with many other languages. Obviously, the best results will require that Pascal be properly used — that subranges be used where possible, for example — and that these be declared to be as small as possible. A Pascal program can contain a great deal of information that allows even a straightforward compiler to generate code which makes good use of the available registers. The Pascal declarations of listing 1 provide illustration for the following discussion, and the code given is for the Motorola 6809. Remember that the intent is not to describe an implementation of Pascal.

```
/ 'X' points to top of stack (display)
lda     D, i(X)       / i
asl     B
rol     A             / *2
add     D, $a-2       / +offset of 'a'
lea     Y, D(X)       / +stack top
lda     D, j(X)       /j
asl     B
rol     A             / *2
add     D, $b-2       / +offset of 'b'
lda     D, D(X)       / +stack top
sta     D, (Y)        / a[i] := b[j]

Total code: 20 bytes
```

Listing 3: Motorola MC6809 assembly code for array indexing program fragment.

```
/ r5 points to the "top" of the
/ stack frame
mov     j(r5),r0        /j
asl     r0              / *2
add     r5,r0           /+ display pointer
mov     i(r5),r1        /i
asl     r1              / *2
add     r5,r1           /+ display pointer
mov     b-2(r0),a-2(r1)                 / a[i] := b[j];

Total code: 22 bytes
```

Listing 4: DEC PDP-11 assembly code for array indexing example.

The declaration of scalar and subrange types essentially allows the declaration of *small* integers and makes known the detailed characteristics of variables of such types to the compiler. Variables may thus be completely bounded, and the compiler can compute upper and lower bounds on the value of an expression.

In our example, variables of type *short* or *shortUnsigned* may be loaded into the 8 bit accumulators of the 6809, and both registers may be used simultaneously. A variable may be recognized as *unsigned* if there are no negative values in the subrange to which it belongs. In the assignment statement *k := i+j;* the variables *i*, and *j*, are both in the range 0 thru 10. The result is thus in the range 0 thru 20, and an 8 bit accumulator may again be used to compute this result. (All of this is particularly useful if array indexing is also involved.)

The Pascal *set* type may be regarded as providing a readable way to do "bit twiddling." A set is typically implemented as a sequence of bits, one for each element of the base type of the set. The variable *s* might then be a byte in which the low order bit corresponds to the element READY, the next to BLOCKED, and so on. The sequence of assignments might then be compiled as in listing 5.

Pascal, of course, provides pointers, **record** structures and arrays.

The use of pointers is strictly controlled: arbitrary arithmetic operations on pointers are not allowed. About the only things that may be done with a pointer variable are: indirect addressing, assigning another pointer to it, or passing it to a procedure or function. This structured use of pointers and indexing results in a very stylized use of pointers in the compiler's internal representation. This in turn allows the compiler to detect the places where double indexing may be used to advantage rather easily, on machines like the 6809 which have this feature.

Indexing of an array of records does require multiplication of the index by the width, in bytes, of the record. Often, this may be accomplished by a shift. Of course, this cannot always be done, since records need not be a power of 2 in length, though a compiler could arrange to round the size of a record up to an appropriate boundary if

```
/ X is display pointer
/ equates are in octal
READY = 01
BLOCKED = 02
RUNNING = 04
SWAPIN = 010
SWAPOUT = 020
lda     A, $READY+BLOCKED+RUNNING / immediate load
sta     A, s(X)
/
lda     A, s(X)
anda    A, $![READY+RUNNING]  / complement
sta     A, s(X)
/
lda     A, s(X)
ora     A, $SWAPIN
sta     A, s(X)
/
lda     A, s(X)
anda    A, $[SWAPIN+BLOCKED]
sta     A, s(X)
```

Listing 5: Set assignment code for the Motorola MC6809 processor.

the difference were small. In any event, provided the size of the record is no more than eight bits (as an unsigned quantity), the code for the multiplication could reasonably be included in line.

We wondered how often division or multiplication is used in the UNIX system (an operating system developed at Bell Labs), and wrote a simple command file which would compile each of the source programs of the system and scan the resulting assembler for *mul* and *div* instructions. The number of multiplications was of interest in light of the above discussion; the number of divisions was collected as well, since these would have to be interpreted by subroutine on the 6809, and we wanted to know how many occurred in critical code. The results are shown in table 2.

Only one of the divide instructions occurs in a routine that might be regarded as significant, with respect to increasing system overhead, were a subroutine called to do the divide piecemeal; and that division was performed at a low priority level. 31 of the divide instructions in the device driver routines were in disk drivers, which had to compute track and cylinder offsets. The multiplications in all cases were of small amounts; it seems that (most likely by accident) record structures used in the kernel happened to be a power of 2 in length. It would have been more instructive, perhaps, to examine user programs, but in that case it would have been more difficult to separate multiplications written explicitly from those created implicitly by array indexing.

Section	Lines of C Code	Number of Multiplications	Number of Divisions
UNIX Kernel	6,013	4	9
Device Drivers	8,640	62	41

Table 2: A search through a particular operating system to determine the number of multiplications and divisions used. This was done to determine how important the speed of a multiplication and division routine would be to a typical program.

A Pascal programmer may declare particular record or array types as *packed*, which is a hint to the compiler that the programmer would prefer elements of the given type to occupy as little space as possible even if there is a cost in increased code to access them. This leaves the unit of packing to the compiler. For example, the types *thing* and *packedThing* (see listing 1) describe packed and unpacked records with similar fields (to Pascal, these record types are not compatible in any way). In a *thing*, both *field1* and *field2* will likely be bytes, but if a compiler implements the notion *packed* completely, then in a *packedThing*, *field1* will likely occupy three bits, and *field2* five bits, ie: they would share the same byte of storage. Packing of records on microcomputers is often much easier than on the larger processors, because microprocessors do not have the alignment problems that plague compiler writers on those machines.

Finally, as in many other languages, the order of evaluation of expressions is left to the implementor, but since side effects are not allowed, no legal Pascal program can possibly be harmed by this. This has two related effects: in arithmetic expressions, the compiler may evaluate the operands in the order that leads to the least amount of code, and in Boolean expressions the left-hand side of the logical operators *and* and *or* need not be evaluated if the expressions on the right determines the truth value of the entire expression. Faster or smaller code will usually result if a compiler takes advantage of these properties.

Pascal: Problems?

We feel that there are a number of areas where Pascal is likely to require expensive mechanisms, and which would be inappropriate for a systems programming environment. One solution might be to implement a subset of the language, leaving these hard features aside, but in most cases, since the expensive mechanisms are only invoked if the programmer asks for them, it should be sufficient to have the compiler avoid including the associated run time procedures when they are not requested. (This is worth mentioning, if only because this rule is often not followed.) We shall first mention those constructs which are expensive, but which appear only by programmer request.

The semantics of Pascal's *file* variables, and the input/output (IO) system in general tend to reflect characteristics of a batch environment, with a restricted character set. The basic IO procedures are badly designed for an interactive terminal. The *read* and *write* procedures are fairly expensive to implement, since they are extremely general and all encompassing.

On machines like the 6809 which lack a divide instruction of any sort (let alone a 16 bit one), division will be done by calling a run time support routine. Only if the programmer explicitly writes either a divide, or modulus operation, will the call be generated. Floating point numbers will be interpreted, as usual.

Pascal allows procedures and functions to be defined inside other procedures and functions. This requires either a display, which must be copied, or a system of pointers by which a routine may access the variables owned by routines in an outer scope. (The latter is the most likely choice.)

Strings, arrays, records and large sets (if implemented) may all be assigned or passed as parameters to routines. These operations require block copies, but only if the operations appear in the source program. Copying of actual parameters may be avoided, of course, by declaring the matching formal parameters as *var* parameters.

The remaining points concern some philosophical concerns about Pascal and its implementation. (Input and output might also be considered in this class.)

Philosophy

It has been observed that much of the checking done at run time in other languages may be done at compile time in Pascal. This is not always so, and run time checks are required on assignments of a variable from a larger subrange to a variable in a smaller subrange of a given type, or on similar use in array indexing, and pointers must always be checked to ensure that they are not *nil*. It might be argued that run time checks might not be done at all. It is better to arrange for them to be turned on and off, as required, in different sections of code.

The *Pascal Report* (see references) does not put boundaries on the number of elements in the base type of a *set* type, but it does say that an implementor will likely choose the word length of a given computer as that limit. Otherwise, routines are required to perform various Boolean operations on large bit strings. Unfortunately, a great many Pascal programs in existence, most notably those for the CDC 6600, assume that it is possible to delcare or use a *set of char*, as in:

```
if c in ['a'..'z'] then
   { c is a letter }
```

where *c* is declared as a *char*. The CDC Pascal compiler restricts the number of elements in the base type of a set to about the number of bits in a word (58), but the CDC character set is small enough that it (nearly) fits within a set. On a microcomputer with the ASCII character set even 16 bits is clearly insufficient, and larger sets may need to be implemented.

There is no method provided to initialize variables in their declaration. This is of consequence when one wishes to create a table with values that remain constant throughout the life of the program (eg: a translation table). The only way to do this in standard Pascal is to write a sequence of assignment statements. This will typically result in several bytes of code for each assignment, as well as forcing two copies of each data value in the table. On a large machine like the CDC 6600, this may be of little consequence, but on a microcomputer with little core, this is a distinct disadvantage. Of course, various implementations of Pascal have provided a means to do this sort of thing efficiently, but this results in a portability problem because each implementor tends to have slightly different rules about where and how these initializations may be accomplished.

Conclusions

For languages like Pascal, compilation is the preferred method of implementation on hybrid 8 and 16 bit microprocessors. The object code size on these machines for common constructs in these languages seems to compare quite favorably with that for larger processors like the PDP-11 or the Honeywell 66/60. We illustrated this with a very simple array operation; the reader can try other operations.

When choosing a programming language, one typically considers not only the ease or difficulty of implementation and the efficiency of the compiled code, but stylistic qualities as well. For example, we have found the C language a pleasant and effective language for developing programs, but it does not, of course, follow that everyone else would. The same holds true for Pascal. We merely note that the Pascal is interesting, in that Pascal programs may be so written as to allow a compiler to compile code which makes efficient use of 8 bit accumulators on machines that have them, and that amongst the other major high level languages this is an unusual property (PL/I is a likely exception). Whatever the language used, we hope to see the day when on microcomputer systems, as on UNIX, the use of assembly language for a program of any size is greeted with surprise, shock, depairs, dismay and outright hostility.■

REFERENCES

1. Jensen, K and Wirth, N, *Pascal User Manual and Report*, Springer-Verlag, New York 1975.
2. Bell, JR, "Threaded Code," *CACM*, volume 16, number 6, June 1973, pages 370 thru 372.
3. Dewer, RBK, "Indirect Threaded Code," *CACM*, volume 18, number 6, June 1975, pages 330 thru 331.
4. Aho, AV, and Ullman, JD, *Principles of Compiler Construction,* Addison-Wesley, Don Mills, Ontario 1977.
5. *990 Computer Family Systems Handbook*, manual number 945250-9701, Texas Instruments, Austin TX 1976.
6. *LSI11 PDP11/03 Processor Handbook*, Digital Equipment Corp, Maynard MA 1975.
7. *M6800 Microprocessor Programming Manual*, Motorola Semiconductor Products, Phoenix AZ 1975.
8. Kernighan, BW, and Ritchie, DM, *The C Proprogramming Language,* Prentice-Hall, Englewood Cliffs NJ 1978.
9. Thompson, KL, and Ritchie, DM, "The UNIX Time-Sharing System," *CACM*, volume 17, number 7, July 1974, pages 365 thru 375.
10. *Honeywell 66/60 Macro Assembly Program*, Honeywell Information Systems, Phoenix AZ 1972.
11. Wiles, et al, "Compatibility Cures Growing Pains of Microcomputer Family," *Electronics*, 2 February 1978.
12. *M6809 Advanced Microprocessor,* Motorola, Austin TX.

MOTOROLA Semiconductor Products Inc.

BOX 20912 • PHOENIX, ARIZONA 85036 • A SUBSIDIARY OF MOTOROLA INC.

ECL PROMs

Organization	Part Number	Access Time (ns max)	Output	No. of Pins
32 x 8	MCM10139▲	25	ECL output	16
256 x 4	MCM10149▲	30	ECL output	16

TTL PROMs

Organization	Part Number	Access Time (ns max)	Output	No. of Pins
64 x 8	MCM5503/ 5303▲	125	Open collector	24
64 x 8	MCM5504/ 5304▲	125	2K pull-up	24
512 x 4	MCM7620▲	70	Open collector	16
512 x 4	MCM7621▲	70	3-state	16
512 x 8	MCM7640▲	70	Open collector	24
512 x 8	MCM7641▲	70	3-state	24
1024 x 4	MCM7642▲	70	Open collector	18
1024 x 4	MCM7643▲	70	3-state	18
1024 x 8	MCM7680▲	70	Open collector	24
1024 x 8	MCM7681▲	70	3-state	24
2048 x 4	MCM7684*▲	70	Open collector	18
2048 x 4	MCM7685*▲	70	3-state	18
2048 x 4	MCM7686*▲	70	Open collector with latches	20
2048 x 4	MCM7687*▲	70	3-state with latches	20
2048 x 4	MCM7688*▲	----	Open collector with registers	20
2048 x 4	MCM7689*▲	----	3-state with registers	20

EPROMs

MOS EPROMs

Organization	Part Number	Access Time (ns max)	No. of Power Supplies[1]	No. of Pins
1024 x 8	MCM2708C▲	450	3	24
1024 x 8	MCM27A08C▲	300	3	24
1024 x 8	MCM68708C▲	450	3	24
1024 x 8	MCM68A708C	300	3	24
2048 x 8	TMS2716C▲	450	3	24
2048 x 8	TMS27A16C▲	300	3	24
2048 x 8	MCM2716C*▲	450	1	24
2048 x 8	MCM27A16C*▲	350	1	24
8192 x 8	MCM68764C*	450	1	24

Motorola Memories

Motorola has developed a very broad range of MOS and bipolar memories for virtually any digital data processing system application. And for those whose requirements go beyond individual components, Motorola also supplies Memory Systems and Micromodules.

New Motorola memories are being introduced continually. **This selector guide lists all those available as of March, 1979.** For later releases, additional technical information or pricing, contact your nearest authorized Motorola distributor or Motorola sales office.

Data sheets may be obtained from your in-plant VSMF Data Center, distributors, Motorola sales offices or by writing to:

Literature Distribution Center
Motorola Semiconductor Products, Inc.
P.O. Box 20912
Phoenix, AZ 85036.

MOTOROLA INC.

MOTOROLA MEMORIES Selector Guide

RAMs ROMs PROMs EPROMs

March 1979

MOTOROLA INC.

MOTOROLA INC.

RAMs

MOS DYNAMIC RAMs

Organization	Part Number	Access Time (ns max)	No. of Power Supplies[1]	No. of Pins
4096 x 1	MCM**4096**C-6▲	250	3	16
4096 x 1	MCM**4096**C-16▲	300	3	16
4096 x 1	MCM**4096**C-11▲	350	3	16
4096 x 1	MCM**4027**AC-2▲	150	3	16
4096 x 1	MCM**4027**AC-3▲	200	3	16
4096 x 1	MCM**4027**AC-4▲	250	3	16
4096 x 1	MCM6604AC	350	3	16
4096 x 1	MCM6604AC-2	250	3	16
4096 x 1	MCM6604AC-4	300	3	16
4096 x 1	MCM6605AL	300	3	22
4096 x 1	MCM6605AL-2	200	3	22
16,384 x 1	MCM**4116**AC-15▲	150	3	16
16,384 x 1	MCM**4116**AC-20▲	200	3	16
16,384 x 1	MCM**4116**AC-25▲	250	3	16
16,384 x 1	MCM**4116**AC-30▲	300	3	16

TTL BIPOLAR RAMs

Organization	Part Number	Access Time (ns max)	Output	No. of Pins
256 x 4	MCM**93412***▲	45	Open collector	22
256 x 4	MCM**93422***▲	45	3-state	22
1024 x 1	MCM**93415**▲	45	Open collector	16
1024 x 1	MCM**93425**▲	45	3-state	16

MOS STATIC RAMs

Organization	Part Number	Access Time (ns max)	No. of Power Supplies[1]	No. of Pins
128 x 8	MCM6810	450	1	24
128 x 8	MCM68A10	360	1	24
128 x 8	MCM68B10	250	1	24
1024 x 4	MCM**2114**P-20▲	200	1	18
1024 x 4	MCM**2114**P-25▲	250	1	18
1024 x 4	MCM**2114**P-30▲	300	1	18
1024 x 4	MCM**2114**P-45▲	450	1	18
1024 x 4	MCM21L14P-20▲	200	1	18
1024 x 4	MCM21L14P-25▲	250	1	18
1024 x 4	MCM21L14P-30▲	300	1	18
1024 x 4	MCM21L14P-45▲	450	1	18
4096 x 1	MCM6641P-20▲	200	1	18
4096 x 1	MCM6641P-25▲	250	1	18
4096 x 1	MCM6641P-30▲	300	1	18
4096 x 1	MCM6641P-45▲	450	1	18
4096 x 1	MCM66L41P-20▲	200	1	18
4096 x 1	MCM66L41P-25▲	250	1	18
4096 x 1	MCM66L41P-30▲	300	1	18
4096 x 1	MCM66L41P-45▲	450	1	18
4096 x 1	MCM2147C-55*▲	55	1	18
4096 x 1	MCM2147C-70*▲	70	1	18
4096 x 1	MCM2147C-85*▲	85	1	18

CMOS STATIC RAMs

Organization	Part Number	Access Time (ns max)	No. of Power Supplies	No. of Pins
256 x 4	MCM145101-1▲	450	1	22
256 x 4	MCM145101-3▲	650	1	22
256 x 4	MCM145101-8▲	800	1	22
4096 x 1	MCM**146504**▲	450	1	18
1024 x 1	MCM**146508***▲	460	1	16
1024 x 1	MCM**146508**-1*▲	300	1	16
1024 x 1	MCM**146518***▲	460	1	18
1024 x 1	MCM**146518**-1*▲	300	1	18

ECL BIPOLAR RAMs

Organization	Part Number	Access Time (ns max)	Output	No. of Pins
8 x 2	MCM10143	15	ECL output	24
256 x 1	MCM**10144**▲	26	ECL output	16
16 x 4	MCM**10145**▲	15	ECL output	16
1024 x 1	MCM**10146**▲	29	ECL output	16
128 x 1	MCM**10147**▲	15	ECL output	16
256 x 1	MCM10152▲	15	ECL output	16

ROMs

MOS STATIC ROMs

Character Generators[2]

Organization	Part Number	Access Time (ns max)	No. of Power Supplies	No. of Pins
128 x (7 x 5)	MCM6670P	350	1	18
128 x (7 x 5)	MCM6674P	350	1	18
128 x (9 x 7)	MCM66700P▲	350	1	24
128 x (9 x 7)	MCM66710P▲	350	1	24
128 x (9 x 7)	MCM66714P▲	350	1	24
128 x (9 x 7)	MCM66720P▲	350	1	24
128 x (9 x 7)	MCM66730P▲	350	1	24
128 x (9 x 7)	MCM66734P	350	1	24
128 x (9 x 7)	MCM66740P▲	350	1	24
128 x (9 x 7)	MCM66750P▲	350	1	24
128 x (9 x 7)	MCM66760P▲	350	1	24
128 x (9 x 7)	MCM66770P	350	1	24
128 x (9 x 7)	MCM66780P	350	1	24
128 x (9 x 7)	MCM66790P	350	1	24

Binary ROMs

Organization	Part Number	Access Time (ns max)	No. of Power Supplies	No. of Pins
1024 x 8	MCM68A30P8	350	1	24
1024 x 8	MCM68A308P7	350	1	24
2048 x 8	MCM68A316P91	350	1	24
1024 x 8	MCM68B30AP▲	250	1	24
1024 x 8	MCM68A30AP▲	350	1	24
1024 x 8	MCM68B308P▲	250	1	24
1024 x 8	MCM68A308P▲	350	1	24
2048 x 8	MCM68A316EP▲	350	1	24
2048 x 8	MCM68A316AP▲	350	1	24
4096 x 8	MCM68A332P▲	350	1	24
4096 x 8	MCM68A332P2*	350	1	24
8192 x 8	MCM68A364P*▲	350	1	24
8192 x 8	MCM68A364P3*	350	1	24
8192 x 8	MCM68B364P3*	250	1	24

CMOS ROM

Organization	Part Number	Access Time (ns max)	No. of Power Supplies	No. of Pins
256 x 4	MCM14524	1200	1	16

*To be introduced.
▲Second source.
Heavy black type denotes industry standard part numbers.
Operating temperature ranges:

MOS	0°C to 70°C
CMOS	−40°C to +85°C and −55°C to +125°C
ECL	Consult individual data sheets.
TTL	Military −55°C to +125°C, Commercial 0°C to 70°C

[1] MOS power supplies:
3 +12, ±5 V
1 +5 V
All MOS outputs are 3-state except the 6570 and 6580 Series which are open-collector.

[2] Character generators include shifted and unshifted characters, ASCII, alphanumeric control, math, Japanese, British, German, European and French

MOTOROLA
SEMICONDUCTORS
3501 ED BLUESTEIN BLVD., AUSTIN, TEXAS 78721

MCM6664

Product Preview

MOS

(N-CHANNEL, SILICON-GATE)

65,536-BIT
DYNAMIC RAM

65,536-BIT DYNAMIC RAM

The MCM6664 is a 65,536 bit, high-speed, dynamic Random-Access Memory. Organized as 65,536 one-bit words and fabricated using HMOS high-performance N-channel silicon-gate technology. This new breed of 5-volt only dynamic RAM combines high performance with low cost and improved reliability.

By multiplexing row- and column-address inputs, the MCM6664 requires only eight address lines and permits packaging in standard 16-pin dual-in-line packages. Complete address decoding is done on chip with address latches incorporated. Data out is controlled by $\overline{\text{CAS}}$ allowing for greater system flexibility.

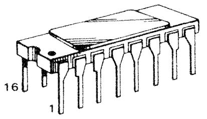
All inputs and outputs, including clocks, are fully TTL compatible. The MCM6664 incorporates a one-transistor cell design and dynamic storage techniques. In addition to the $\overline{\text{RAS}}$-only refresh mode, refresh control function available on pin 1 provides automatic and self-refresh modes.

- Organized as 65,536 Words of 1 Bit
- Single +5 V Operation
- Fast 150 ns Operation
- Low Power Dissipation
  250 mW Maximum (Active)
  30 mW Maximum (Standby)
- Three-State Data Output
- Internal Latches for Address and Data Input
- Early-Write Output Capability
- 16K Compatible 128-Cycle, 2 ms Refresh
- Control on Pin 1 for Automatic and Self Refresh
- $\overline{\text{RAS}}$-only Refresh Mode
- $\overline{\text{CAS}}$ *Controlled Output Providing Latched or Unlatched Data*
- Upward Pin Compatible from the 16K RAM (MCM4116)

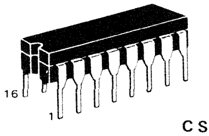
OUTPUT BUFFER TRUTH TABLE

Internal Early Write	$\overline{\text{CAS}}$	Refresh Control (CAS Internal)	Output Buffer
H	X	X (X)	Hi-Z
X	H	X (X)	Hi-Z
L	L	L (H)	Maintains Previous Data
L	L	H (L)	Active

This is advance information and specifications are subject to change without notice.

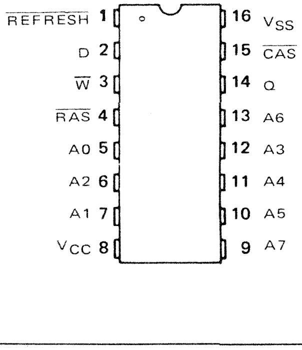


L SUFFIX
CERAMIC PACKAGE
CASE 690



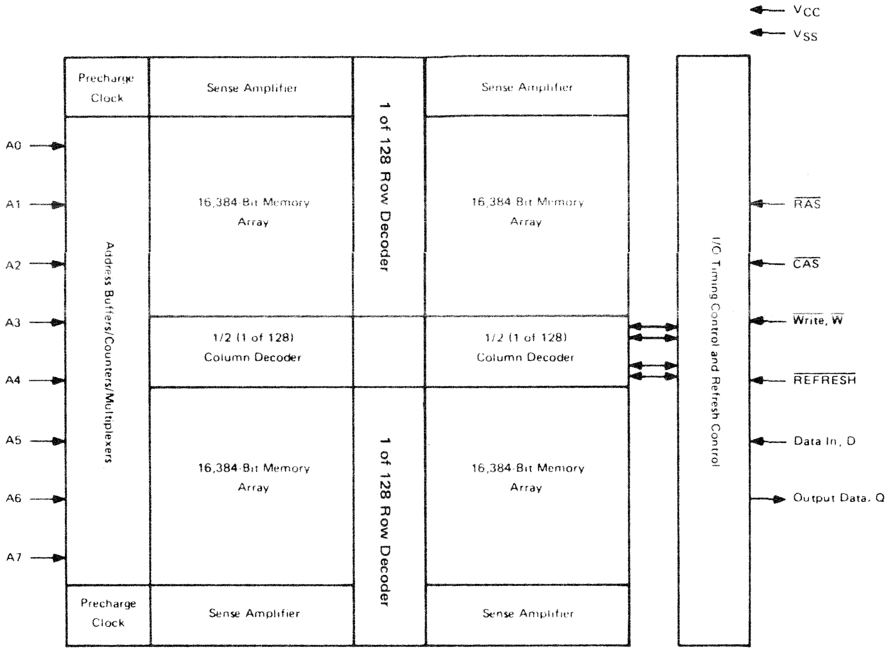
C SUFFIX
FRIT-SEAL
CERAMIC PACKAGE
CASE 620

PIN ASSIGNMENT



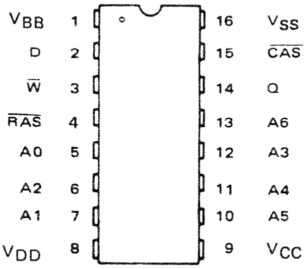
This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum rated voltages to this high impedance circuit.

BLOCK DIAGRAM

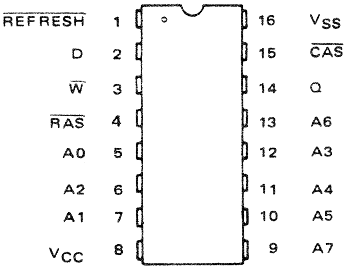


4116 TO 6664 COMPARISON

MCM4116



MCM6664



PIN VARIATIONS

PIN NUMBER	MCM4116	MCM6664
1	V_{BB} (−5 V)	$\overline{\text{REFRESH}}$
8	V_{DD} (+12 V)	V_{CC} (+5 V)
9	V_{CC} (+5 V)	A7

On-Chip Refresh Features/Benefits

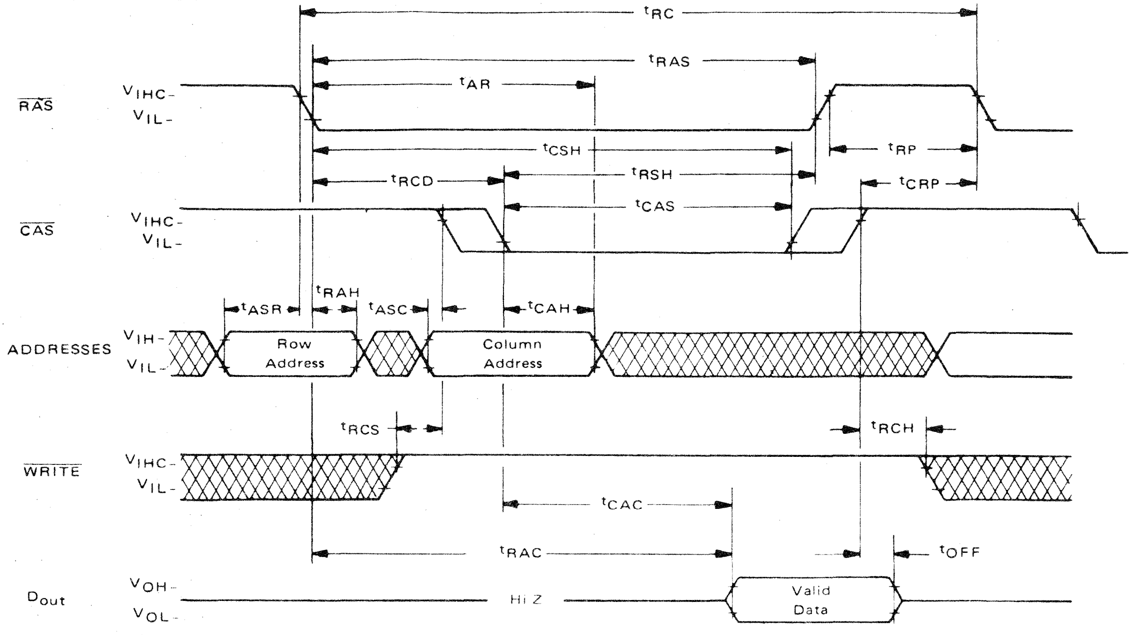
Reduce System Refresh Controller Design Problem

Reduce System Parts Count

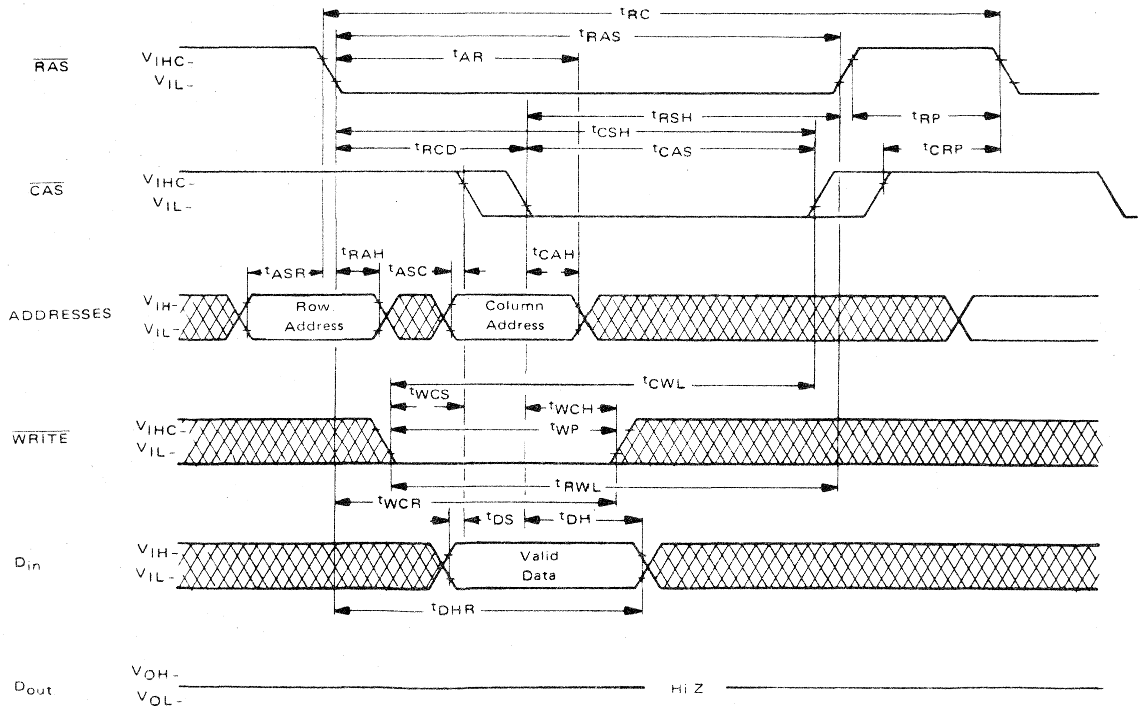
Reduce System Noise Increasing System Reliability

Reduce System Power During Refresh

READ CYCLE TIMING

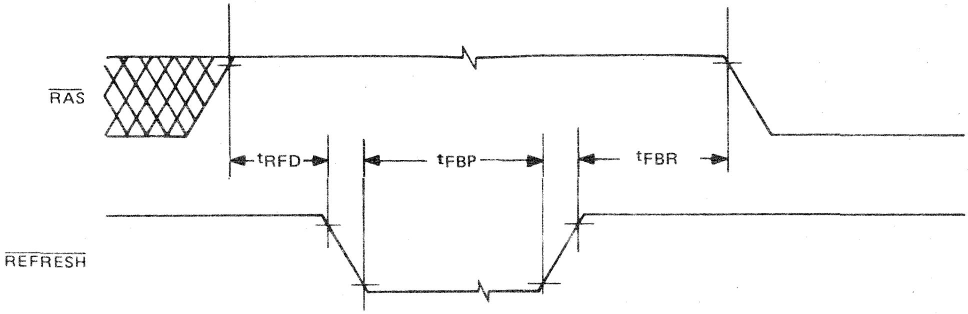


WRITE CYCLE TIMING



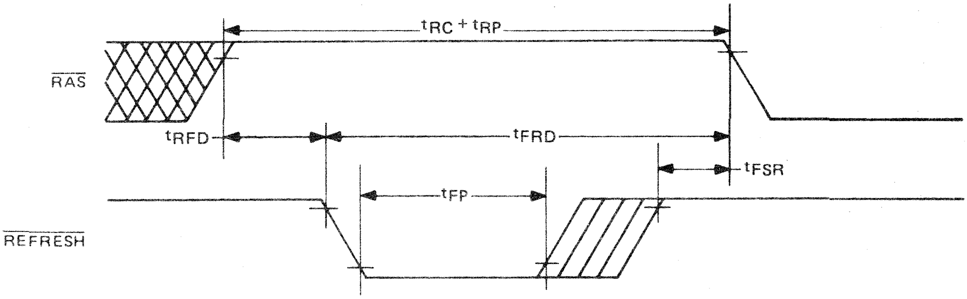
SELF REFRESH MODE (Battery Backup)

(CAS[1], Addresses, Data-In, and Write are Don't Care)



AUTOMATIC PULSE REFRESH CYCLE

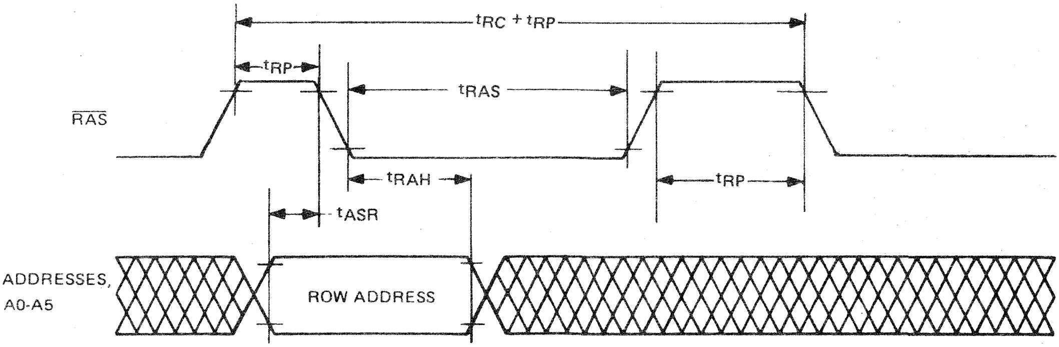
(CAS[1], Addresses, Data-In, and Write are Don't Care)



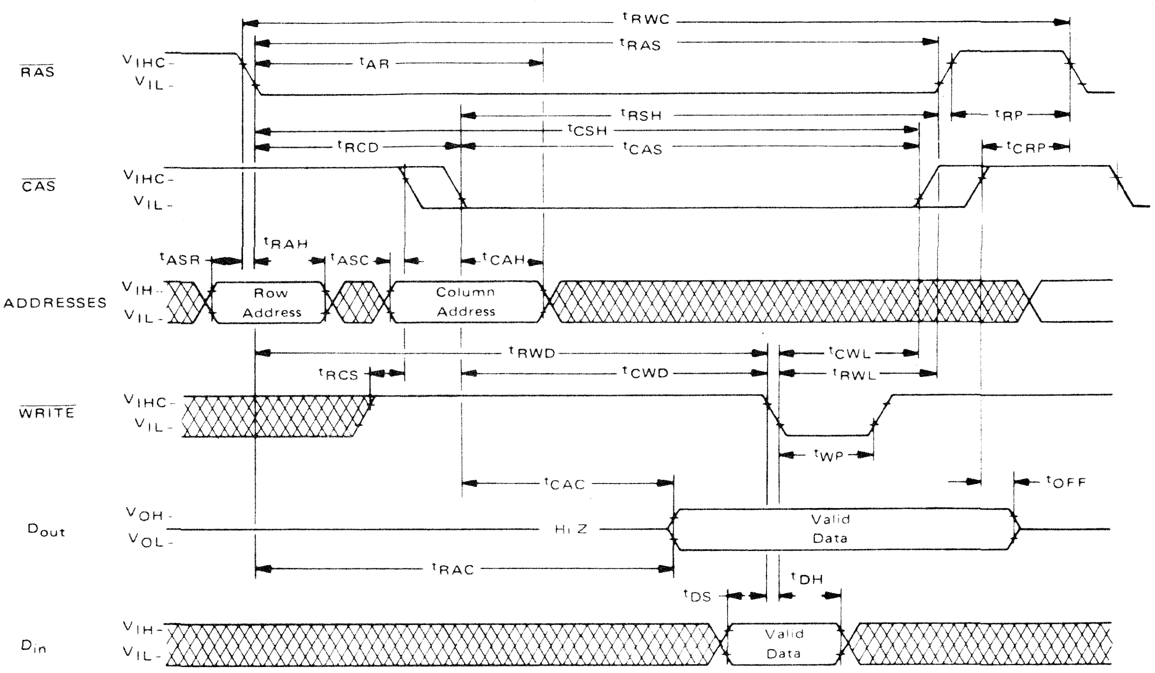
[1] CAS controls the output data. If CAS remains low the previous output will remain valid. When CAS is brought high, the output will assume a high-impedance state.

RAS-ONLY REFRESH CYCLE

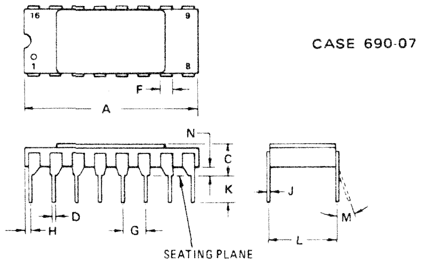
(Data-in and Write are Don't Care, CAS is HIGH)



READ-WRITE/READ-MODIFY-WRITE CYCLE



PACKAGE DIMENSIONS

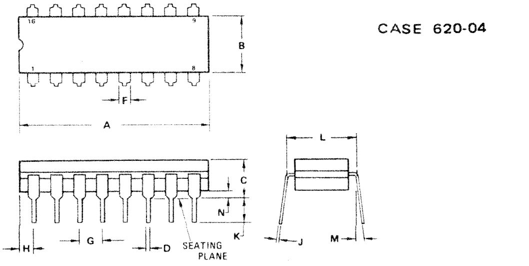


CASE 690-07

DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	18.80	19.23	0.740	0.757
C	2.67	3.94	0.105	0.155
D	0.41	0.51	0.016	0.020
F	1.14	1.40	0.045	0.055
G	2.54 BSC		0.100 BSC	
H	0.51	0.71	0.020	0.028
J	0.20	0.31	0.008	0.012
K	3.56	4.83	0.140	0.190
L	7.62 BSC		0.300 BSC	
M	–	10⁰	–	10⁰
N	0.64	1.14	0.025	0.045

NOTE

1. LEADS WITHIN 0.13 mm (0.005) RADIUS OF TRUE POSITION AT SEATING PLANE AT MAXIMUM MATERIAL CONDITION.



CASE 620-04

DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	19.05	19.94	0.750	0.785
B	6.10	7.49	0.240	0.295
C	–	5.08	–	0.200
D	0.38	0.53	0.015	0.021
F	1.40	1.78	0.055	0.070
G	2.54 BSC		0.100 BSC	
H	0.51	1.14	0.020	0.045
J	0.20	0.30	0.008	0.012
K	2.54	–	0.100	–
L	7.49	8.89	0.295	0.350
M	–	15⁰	–	15⁰
N	0.51	1.02	0.020	0.040

NOTES:

1. LEADS WITHIN 0.13 mm (0.005) RADIUS OF TRUE POSITION AT SEATING PLANE AT MAXIMUM MATERIAL CONDITION.
2. PKG. INDEX: NOTCH IN LEAD NOTCH IN CERAMIC OR INK DOT.
3. DIM "A" AND "B" DO NOT INCLUDE GLASS RUN-OUT.
4. DIM "L" TO INSIDE OF LEADS (MEASURED 0.51 mm (0.020) BELOW BODY)