DEC-08-UMPHA-A-D

MPS
MICROPROCESSOR SERIES
USER'S HANDBOOK

1st Edition, July 1974
2nd Printing (Rev), November 1974

Printed in U.S.A.

CONTENTS

CONTENTS (Cont)

CONTENTS (Cont)

CONTENTS (Cont)

CONTENTS (Cont)

ILLUSTRATIONS

ILLUSTRATIONS (Cont)

TABLES

TABLES (Cont)

PREFACE

The user's handbook provides a detailed range of hardware and software information pertinent to the operation of Microprocessor Series (MPS) modules. This information is presented in ten chapters:

Chapter 1 provides an overview of the functions performed by the MPS modules and the programming routines

Chapter 2 presents a functional description of each MPS module based on detail block diagrams

Chapter 3 consists of a detailed presentation of the Processor Module instruction repertoire

Chapter 4 describes, in detail, the PDP-8 host environment as it applies to the use of the applicable program in the software package supplied by Digital to support user-development of MPS system programs

Chapter 5 provides the application programmer with the detailed information necessary to the use of the Microprocessor Language Editor (MLE)

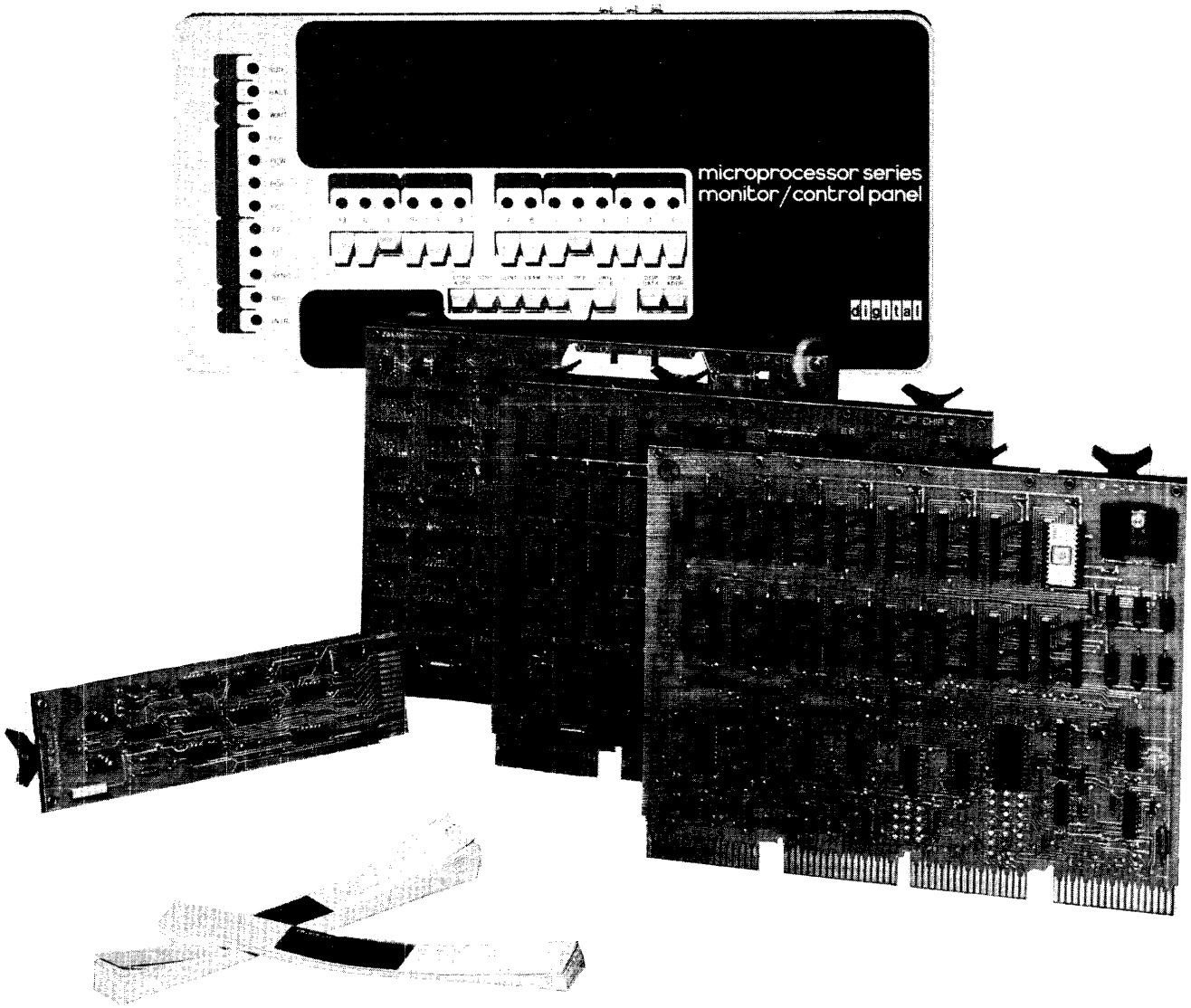
Chapter 6 describes, in detail, the use of the Microprocessor Language Assembler (MLA)

Chapter 7 summarizes the operation of the Microprocessor Program Loader (MPL)

Chapter 8 presents instructions for utilizing the Microprocessor Debugging Program (MDP) which facilitates analysis and alteration of binary programs

Chapter 9 provides operating instructions for reading and writing data and instruction bits into programmable read-only memory (PROM) circuits using the Microprocessor Read-Only Memory Programmer (MRP)

Chapter 10 contains sample programs which might be useful to the user as a reference aid


microprocessor series
monitor/control panel
digital

CHAPTER 1
OPERATING CHARACTERISTICS

1.1 INTRODUCTION

Digital Equipment Corporation's Microprocessor Series (MPS) consists of a group of four M Series modules and an optional operator's control panel, designed to efficiently perform a range of process control and decision-making functions that were previously uneconomic subjects for automation. When used together, these modules can form low-cost digital control systems that exhibit the characteristics normally attributed to more costly minicomputer-based systems. With this capability, systems structured from MPS modules can perform the functions of dedicated controllers, operate as a Central Processor Unit (CPU) in intelligent terminals, perform data aquisition and analysis tasks in the laboratory, and automate a host of industrial processes.

1.2 GENERAL DESCRIPTION

The Microprocessor Series is listed below by model number and name:

- M7341 Processor Module
- M7344-YA 1K Read/Write Memory Module
  M7344-YB 2K Read/Write Memory Module
  M7344-YC 4K Read/Write Memory Module
- M7345 Programmable Read-Only Memory Module
- M7346 External Event Detection Module
- KC341 Monitor/Control Panel

In a systems context, the M7341 Processor Module (PM) acts as the central processor unit with the remaining modules performing supporting functions. Activity in a given system, then, is directed by a unique stored program contained in a read/write and/or a programmable read-only memory and executed by the PM. A major factor in the structuring of an MPS system for a specific application is the development of this unique system program by the user. To support user development of application software, Digital provides the Microprocessor Series Software Tools (MPSST) package that includes the following routines:

- Microprocessor Language Editor (MLE)
- Microprocessor Language Assembler (MLA)
- Microprocessor Read-Only Memory Programmer (MRP)
- Microprocessor Host Loader (MHL)
- Microprocessor Debugging Program (MDP)
- Master Tape Duplicator (MTD)

In addition, the Microprocessor Program Loader (MPL) is available to users of the optional KC341 Monitor Control Panel.

1.3 FUNCTIONAL DESCRIPTION

1.3.1 Microprocessor Series Modules

The discussions that follow present brief descriptions of the functions performed by each module within the context of a generalized MPS system structure.

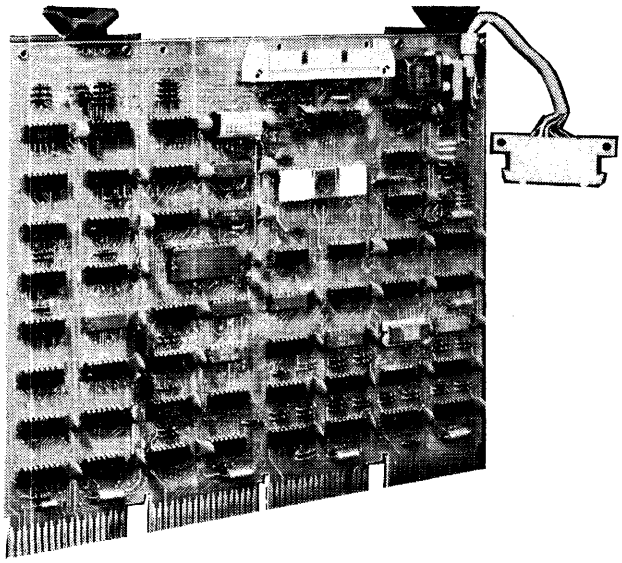


Figure 1-1 Processor Module

Processor Module (Figure 1-1)
The M7341 Processor Module performs the functions of a CPU in a system structured from MPS modules. The module consists of solid-state integrated circuits with input and output lines that are TTL-compatible; its major CPU functions are executed by a single-chip, large-scale, integrated (LSI) microprocessor. Supportive functions such as timing, data and address busing, input multiplexing, gating, buffering, storage and external communication are performed by the remaining logic population on the board.

The processor chip is a parallel, 8-bit control processor unit configured as a single metal oxide silicon circuit packaged in an 18-pin dual in-line package. Through the supportive logic in the M7341 module, the processor can communicate the consequences of program execution with all other MPS modules.

LSI processor internal logic includes an accumulator, two memory address registers, six general-purpose registers, four condition flip-flops, complete instruction control and decoding logic, and a stack. All communication between internal registers and logic and other MPS modules and peripheral devices is conducted through an 8-bit bidirectional data port integral to the processor chip. The internal stack contains the 14-bit program counter and seven other 14-bit registers for nesting up to seven levels of subroutines. This 14-bit addressing capability permits accessing up to 16K memory locations that can be any mixture of RAM or ROM.

The instruction control and decoding logic implement a set of 48 register transfer, arithmetic, control, and logical instructions which are specifically optimized for the process control environment. The processor chip is also equipped with an interrupt line under control of supporting PM logic which allows the enabling or disabling of interrupts. Input to this interrupt recognition logic is generated by the external event detection module which implements the detection of, and response to, application-defined events or power failure conditions. Enabling and disabling interrupts is performed under program control.

Serial communication between the processor and external equipment is furnished by a universal asynchronous receiver/transmitter which is also part of the PM supportive logic. Through this interface, programs can be loaded from an external peripheral device such as a paper-tape loader and MPS systems communicating directly with external data bases.

Read/Write and Programmable Read-Only Memories (Figures 1-2 and 1-3)
These two MPS memory modules provide the user with a wide range of options with respect to the mixing of RAM and ROM memory within a system. The read/write memory module is available in three versions: a 1K module, a 2K module, and a 4K module. All memory circuits in a programmable read-only memory module are socket-mounted so that the storage capacity of a given module can be expanded to 4K by adding memory circuits.

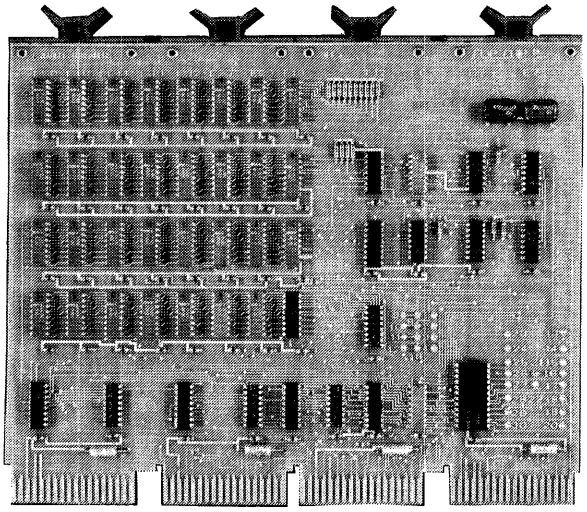


Figure 1-2 Read/Write Module

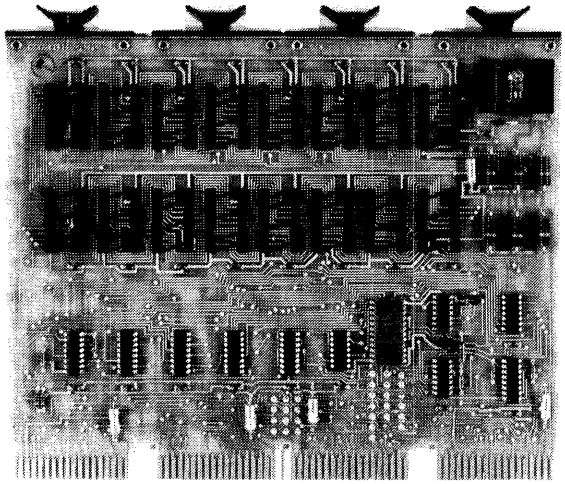


Figure 1-3 Programmable Read-Only Module

This feature permits varying PROM capacity in response to changing system requirements. Each PROM circuit is equipped with a sealed transparent quartz lid which permits erasure prior to reprogramming using an ultraviolet light source.

With both memory modules, the address range for each – within a group of modules forming a system memory – can be determined either by inserting the appropriate jumpers or through backplane selection.

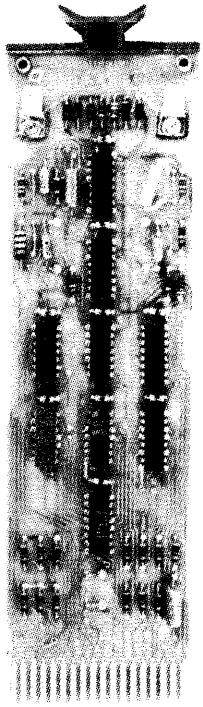


Figure 1-4 External Event Detection Module

External Event Detection Module (Figure 1-4)
The M7346 External Event Detection Module (EEDM) implements nine levels of priority arbitration including application-defined six level priority interrupt schemes, an ac and dc power failure detection capability, and the processor control functions of halt and restart. The EEDM is completely contained on a single-height, extended-length PC board.

Monitor/Control Panel (Figure 1-5)
The KC341 Monitor/Control Panel (MCP) serves as an operator's panel for the processor module. In addition to the conventional panel controls and indicators, the MCP is equipped with controls and visual displays for examining and changing the content of manually accessed read/write memory locations and for performing single-step instruction execution. These functions are supported by a resident memory consisting of a 256 X 8-bit PROM and a 32 X 8-bit RAM completely contained on the MCP. The PROM contains the Microprocessor Program Loader (MPL). The RAM and the PROM are directly addressable as system memory, and the RAM can be used as a scratch pad by user-diagnostics and by operating programs.

The MCP interfaces with the processor module through a dedicated cable of up to 8 feet. Although normally configured for table mounting, the MCP can be panel-mounted in a standard EIA rack panel fitted with a suitable bezel.

1.3.2 MPSST Software Tools Package

This software package (supplied by Digital) aids the user in developing application programs. The support functions performed by each of these routines are presented in the following paragraphs.

Microprocessor Language Assembler (MLA)

MLA is a three-pass symbolic assembler that operates on a PDP-8 to produce either a listing or a binary punched paper tape of an MPS object program from punched paper tape source code. This program has been designed to conform generally to the operational characteristics of other PDP-8 assemblers. Assembled code is generated in punched paper tape form or as a printed listing at the option of the user. Diagnostic messages are also printed out to designate syntax errors and to indicate warnings or actions taken by the assembler.

Microprocessor Language Editor (MLE)

MLE is an on-line symbolic editor that operates on the PDP-8 to create and modify MPS source program punched paper tapes. This editor implements both program entry and on-line program editing. Source text can be entered from a keyboard or from a punched paper-tape reader. After editing, the user may produce a source paper tape ready for input to MLA and/or a source text listing. Listings and tapes of source text can be made in whole or in part as required by the user.

Microprocessor Host Loader (MHL)

MHL is a utility program loaded into core to read binary-coded data from paper tape and to store it in core memory, and used primarily to load system binary object programs.

Microprocessor Read-Only Memory Programmer (MRP)

MRP operates on an MR873 PROM writer in conjunction with a PDP-8/E, /F, or /M to set data and instruction bits into ultra-violet light erasable PROM circuits using object tapes produced by MLA.

Microprocessor Debugging Program (MDP)

MDP operates on the processor module in conjunction with the Monitor/Control Panel from either PROM or RAM memory. This octal debugger permits the following diagnostic actions under control of the MCP panel as directed by an operator:

- Reads and punches paper tape
- Opens specified memory locations for modifications and allows the previous, current, and next locations to be opened, displayed, and closed
- Dumps the contents of program addresses, status flip-flops, and index registers on the Teletype printer
- Allows a program segment to execute for test purposes under MDP control
- Specifies a breakpoint location for program execution
- Loads specified locations in memory with a constant value

Microprocessor Program Loader (MPL)

MPL is a binary paper-tape loader that operates on the processor module and resides in the Monitor/Control Panel PROM memory. This program provides for the loading of a binary punched tape from an external paper-tape reader through the universal asynchronous receiver/transmitter integral to the PM. Operation of MPL is performed from the MCP control panel.

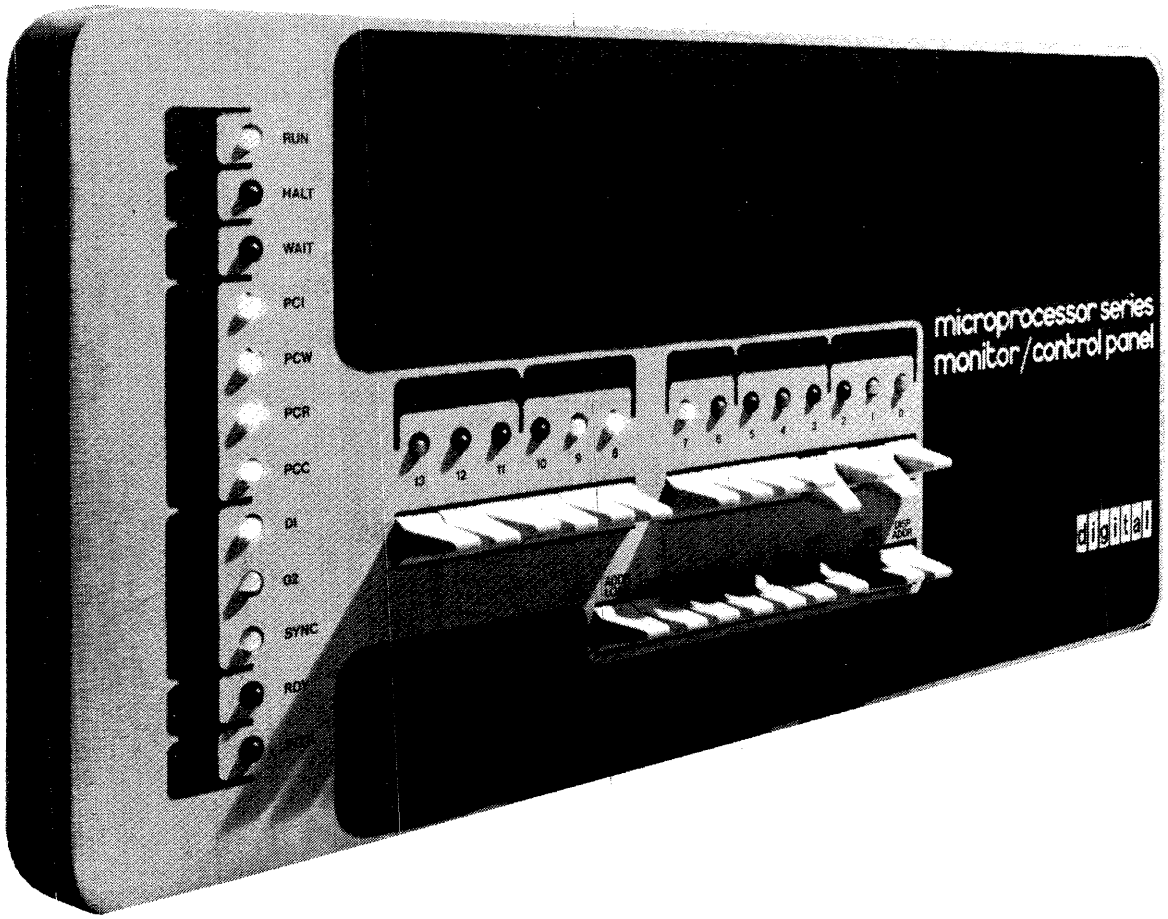


Figure 1-5 Monitor/Control Panel

1.4 SPECIFICATIONS

1.4.1 Performance Specifications

M7341 Processor Module

Operating Speed @ 500 kHz	2 μs
Two-phase clock period	4 μs
Time state	12 to 44 μs
Instruction time	
Word Size	8-bit word
Data	1, 2, or 3 8-bit words
Instruction	14 bits
Address	

Input Data Ports		
Memory data	8 bits	Multiplexed
Peripheral data	8 bits	
Power fail/stop	8 bits	
I/O Interrupt/start	8 bits	
Input/Output Lines		
Memory data	8 bits bidirectional	
Memory address	14 bits, output only	
Peripheral data	8 bits input and output	
Peripheral address	5 bits, output only (may be expanded)	
Communication Lines	2; 20 mA current loop, active or passive	
Baud rate		
With internal clock	110 baud (1.76 kHz)	
With external clock	9600 baud (153.6 kHz) maximum (TTL)	
Instruction Repertoire		
Forty-eight basic instructions		
Instruction Categories		
Register Operation		
Accumulator Operation		
PC and Stack Control		
I/O		
Machine		

KC341 Monitor/Control Panel

Controls	
Switch register	14-switch manual input register
ADDR LOAD	Load address from switch register with PM halted
STRT	Start Processor Module
CONT	Execute one machine cycle if in single cycle mode, or continue program execution at machine speed if not in single cycle mode
EXM	Display content of memory location addressed by either switch register content or incremented switch register content
DEP	Deposit content of switch register into a memory location accessed by a previously loaded address
SING CYCLE	Enter single cycle mode
DISP DATA	Display data contained in location being examined
DISP ADDR	Display address loaded from switch register or address of location being examined

Indicators

RUN	Lights when processor module is operating
HALT	Lights when processor module is in the stopped state
WAIT	Lights when processor module is in wait state
PCI, PCW, PCR, PCC	Each indicator lights when the processor module is executing the corresponding machine cycle
ϕ 2, ϕ1, SYNC	These indicators light to designate the operation of the corresponding processor module timing signal
RDY	Lights when the processor module Ready line is asserted true
INTR	Lights when the processor module Interrupt line is asserted true

MCP-PM Interface

Cable length (max) @ PM clock rate of 500 kHz/phase	8 ft. (2.4m)
Connector/plug types	50-pin PC board connector/header
Cable type	50-conductor, flat, shielded

M7344-YA, -YB, -YC Read/Write Memory Modules

Memory type	Static MOS
Data word size	8 bits
Address word size	14 bits, expandable to 16 bits plus address expansion line
Number of words	1024, 2048, or 4096
Memory read or write cycle time	1.15 μs

M7345 Programmable Read-Only Memory Module

Memory type	Static MOS PROM
Data word size	8 bits
Address word size	14 bits, expandable to 16 bits plus selection line
Number of words	Up to 4K (multiples of 256)
Cycle time	1.0 μs
Erasure method	Ultraviolet light; 256 words erased per circuit exposed
Program write time	2 minutes, typical per 256 words

M7346 External Event Detection Module

Number of event detection input lines	9
Priority encoded	1, lowest
	9, highest
External event response time	12 to 44 μs
Power failure response time	21 ms (from ac loss to power-fail request)
Input polarity	Zero volts true
Output polarity	Zero volts true
Power fail sense input	6.3 Vac

1.4.2 Electrical Specifications

Power supply (all modules)	+5 Vdc, -15 Vdc, ±5%
Input Logic Levels (all modules)	
TTL Logical Low	0.0 to 0.8 Vdc
TTL Logical High	2.0 to 3.6 Vdc
Output Logic Levels (all modules)	
TTL Logical Low	0.0 to 0.4 Vdc
TTL Logical High	2.4 to 3.6 Vdc

Power Consumption

Processor Module	1.62 A @ +5 V, 150 mA @ -15 V, 10.25 W
Monitor/Control Panel	2.43 A @ +5 V, 60 mA @ -15 V, 12 W
Read/Write Memory, M7344-YA	1.2 A @ +5 V, 6.0 W
M7344-YB	1.5 A @ +5 V, 7.5 W
M7344-YC	2.2 A @ +5 V, 11.0 W
Programmable Read-Only Memory, 1K	490 mA @ +5 V, 300 mA @ -15 V; 6.0 W
2K	630 mA @ +5 V, 530 mA @ -15 V; 11.0 W
4K	900 mA @ +5 V, 1.0 A @ -15 V; 19.5 W
External Event Detection Module	250 mA @ +5 V, 50 mA @ 6.3 Vac, 1.5 W

1.4.3 Mechanical Specifications

M7341 Processor Module

Board type	Quad-height, extended-length, single width
Dimensions	10.436 X 8.50 X 0.50 in. (26.5 X 21.6 X 1.27 cm)

KC341-B Monitor and Control Panel

Overall panel dimensions

Width	18 in. (45.7 cm)
Height	8.75 in. (22.2 cm)
Depth	1.75 in. (4.4 cm) excluding switches
	2.50 in. (6.35 cm) including switches

M7344 Read/Write Memory Module

Board type	Quad-height, extended length, single width
Dimensions	10.436 X 8.50 X 0.50 in. (26.5 X 21.6 X 1.27 cm)

M7345 Programmable Read-Only Memory Module

Board type	Quad-height, extended length, single width
Dimensions	10.436 X 8.50 X 0.50 in. (26.5 X 21.6 X 1.27 cm)

M7346 External Event Detection Module

Board type	Single height, extended length, single width
Dimensions	2.4375 X 8.50 X 0.50 in. (6.2 X 21.6 X 1.27 cm)

1.4.4 Environmental Specifications (all modules)

Ambient Temperature	
Operating	5° to 50° C (41° to 122° F)
Nonoperating	-40°to 66° C (-40° to 150° F)
Humidity	10 to 95% noncondensing

CHAPTER 2
FUNCTIONAL DESCRIPTION

2.1 INTRODUCTION

This chapter presents a detailed functional description of each Microprocessor Series module and the operator's monitor/control panel. The discussion conveying these descriptions is based on comprehensive block diagrams which relate input and output signals and internal signal flow to the event sequence within each module.

Each of the detailed block diagrams supporting these discussions graphically represents module throughput as logic circuit blocks that are functionally cohesive. For example, registers, multiplexers, gating networks, clocks, and various control logic are depicted as functional blocks. Data and address buses, control and enabling lines, and internally generated signals are shown as they affect the pertinent functional blocks. The discussions supported by these block diagrams deal with the effect of inputs on the function of throughput, how functional blocks interrelate to implement throughput, and what actions result from outputs. A more comprehensive technical description including pinouts, input and output loading, signal descriptions, and jumper selections is contained in the respective data sheet supplied with each module.

2.2 PROCESSOR MODULE

The M7341 Processor Module (PM) contains a single chip MOS/LSI microprocessor along with the integrated logic and control circuitry necessary to operate as a parallel 8-bit central processing unit. This microprocessor support logic consists of an adjustable 500 kHz variable clock; a four channel input multiplexer; data, memory, and address bus gating; I/O control logic; interrupt recognition logic; and a universal asynchronous receiver/transmitter driven by an integral 844.8 kHz clock. The relationships of the supporting logic to the processor chip are shown in Figure 2-1.

The single chip microprocessor contains a bidirectional data port, complete instruction decoding logic, an arithmetic unit, a state counter, an accumulator, an address stack, six general registers, and memory and I/O timing and control logic.

2.2.1 Processor Module Timing

The basic timing signals shown in Figure 2-1, for the PM are produced by the two-phase clock. These signals, labeled $\phi 1$ and $\phi 2$, are symmetrical, nonoverlapping positive-going clock pulses which drive the processor chip state counter. This state counter controls all activity internal to the processor chip and produces the output signals S0, S1, S2, and SYNC. The timing signals available for external use are $\phi 1$CLK H, $\phi 2$CLK H, and SYNC H. The SYNC H signal, along with S0, S1, and S2 defines processor module instruction execution states.

A typical PM machine cycle involves five sequential time states: TS1, TS2, TS3, TS4, and TS5 (Figure 2-2). During time states TS1 and TS2, system memory is addressed by a lower and an upper address byte respectively to form a 14-bit address; during TS1 the program counter (PC) is incremented. In time state TS3 the instruction addressed during TS1 and TS2 is fetched, and during TS4, or TS4 and TS5, the fetched instruction is executed. The flow chart of time state transitions is shown in Figure 2-3 which simplifies the progression through time states during a machine cycle.

If an interrupt is initiated by an external event, control does not return to TS1 after completing instruction execution but instead reverts to time state TS1I which replaces TS1. During TS1I, the external event is recognized, an interrupt is generated, and incrementing of the PC is suppressed to permit execution of a one-byte instruction generated by the external event.

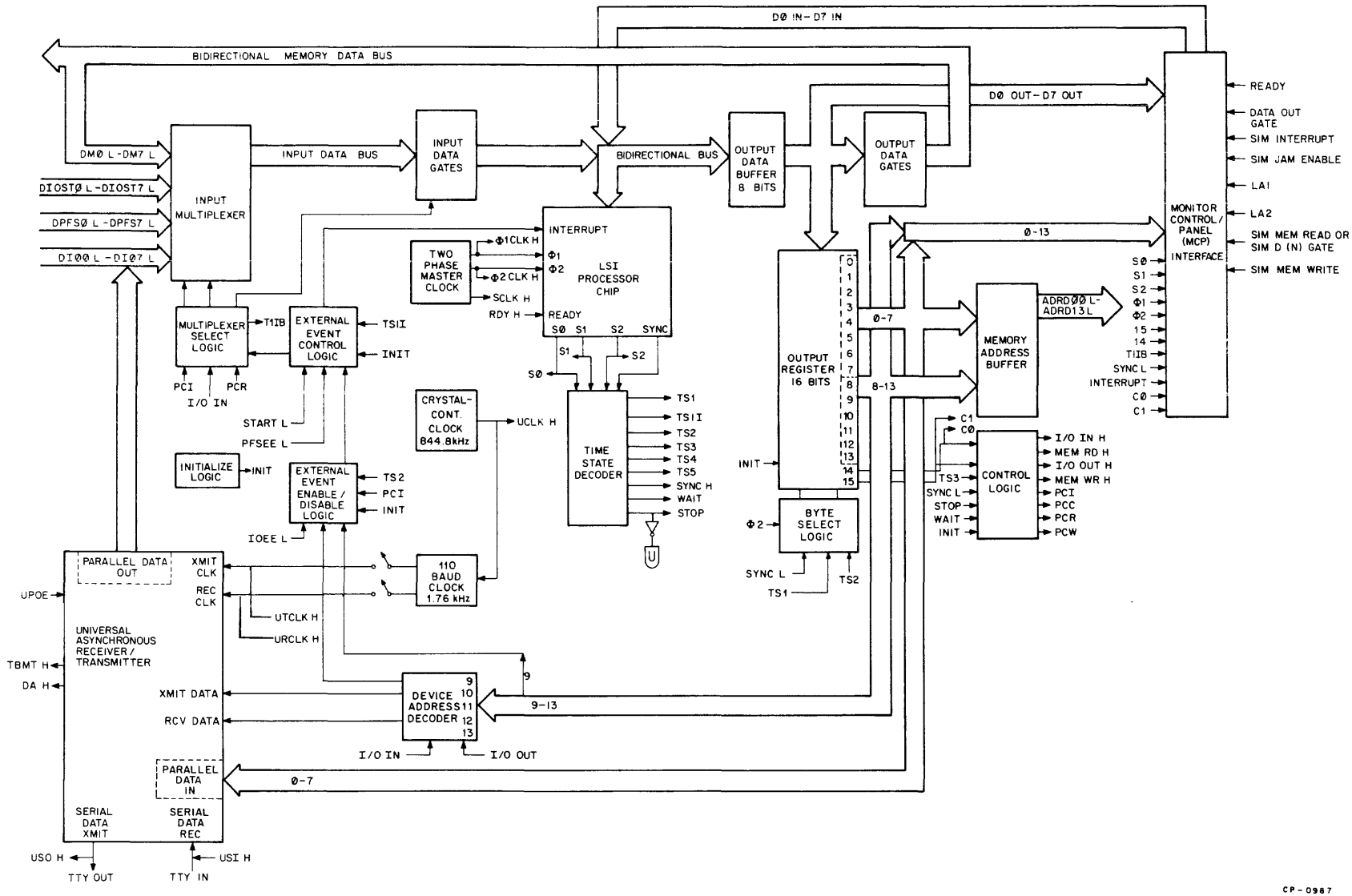
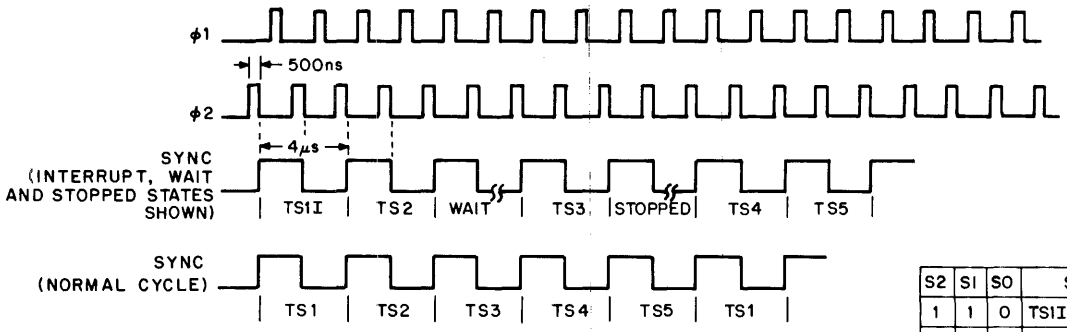


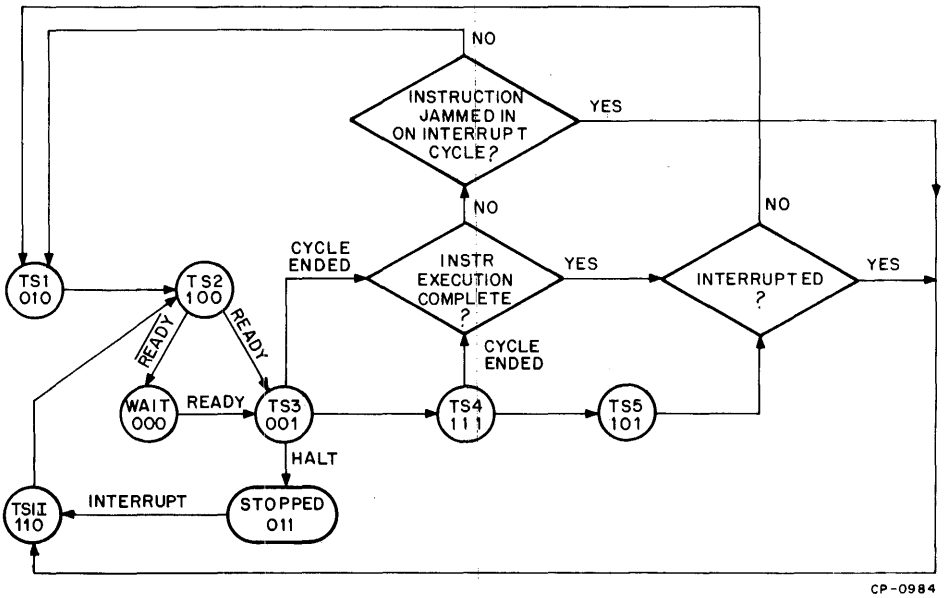
Figure 2-1 PM Block Diagram



S2	SI	SO	STATE
1	1	0	TSII (INTERRUPT)
0	1	0	TS1
1	0	0	TS2
0	0	0	WAIT
0	0	1	TS3
0	1	1	STOPPED
1	1	1	TS4
1	0	1	TS5

CP-0985

Figure 2-2 Instruction Execution States



CP-0984

Figure 2-3 Time State Flow Diagram

At the completion of time state TS2, the processor checks the state of the READY line. If this line is true (High), time state TS3 is entered; if not true (Low), the Wait state is entered. Time state TS3 is entered from the Wait state when the READY line is asserted again. The state of the READY line is available for external use through the signal RDY H.

If the instruction fetched during time state TS3 is a Halt, the processor stops operation at the end of that time state. The processor remains halted until the START line is asserted forcing entry into time state TS1I and execution of a jammed one-byte instruction which can be supplied by the External Event Detection Module. When the PM is operating (STOP H not asserted) the RUN indicator is lit; when the Stop state is entered (STOP H asserted), this indicator will be extinguished.

2.2.2 Processor Module Instruction Cycle

Figure 2-3 shows that a machine cycle can be completed at the end of time states TS3, TS4 or TS5. The instruction cycle for instructions in the PM repertoire is variable depending on the class and function of the specific instruction executed and can consist of one, two, or three machine cycles. The completion point within a machine cycle is also instruction-dependent so that the number of time states encompassed by PM instructions can range from a minimum of three to a maximum of eleven.

The processor module executes four types of machine cycles which are listed and defined below:

PCI Cycle – This is always the first cycle of every PM instruction and initiates an instruction fetch. The two bytes which address memory during this cycle are always taken from the PC.

PCR Cycle – This cycle initiates the addressing of memory by the incremented PC to retrieve a subsequent byte of a two- or three-byte instruction, or to retrieve a data byte addressed by the contents of registers H and L.

PCC Cycle – This cycle initiates the set-up and execution of I/O instructions by placing the address of the peripheral device to be accessed and the content of the accumulator onto the memory address bus and retrieving and/or storing the data at the pertinent peripheral.

PCW Cycle – This cycle initiates the addressing of memory by the content of the H and L registers and implementing the writing of data into that location.

As shown in Figure 2-1, a corresponding signal for each of these machine cycles is available as output from the processor. At time state TS2, the specific signal corresponding to the machine cycle being executed is asserted and latched for external use. These signals are derived from the states of the two high-order bits of the high address byte and are decoded and gated out for external use during time state TS3.

2.2.3 Input Data Paths

The processor chip is equipped with a single time-shared 8-bit bidirectional data port to permit memory addressing, instruction fetching, and data input and output. This port connects to the bidirectional data bus on the processor module. As shown in Figure 2-1, input data in the form of an 8-bit byte is gated onto this internal bidirectional bus from the unidirectional input data bus. Data is multiplexed and gated onto the input data bus from four input ports which are selected as a function of the machine cycle currently being executed.

When a PCI or PCR machine cycle is in process, time state TS3 selects the bidirectional memory port DM0 L to DM7 L to fetch the instruction or data word addressed during TS1 and TS2 of that cycle from memory. During a PCC cycle the signal I/O IN, asserted by the control logic, selects the peripheral data-in port DI0 L to DI7 L to retrieve data from the addressed peripheral device as specified by the read I/O instruction being executed.

At start-up or restart time, or in response to an external event, one of the signals START L or IOEE L is asserted to select the I/O start port DIOST0 L to DIOST7 L for external instruction input.

Figure 2-3 illustrates that when an interrupt occurs in response to an external event, time state TS1I is entered so that normal incrementing of the PC is inhibited.

As a consequence of selecting the I/O start port, an externally supplied one-byte instruction is automatically fetched. This instruction is executed during time states TS4 and TS5. Note that when IOEE L is asserted, the external event recognition logic must be enabled under program control in order for the I/O start port to be selected as a response to an external event. The signal START L, when asserted, bypasses the event recognition to select the I/O start port regardless of program-enabling action.

The signal PFSEE L can be asserted by a system start-stop switch, by a power-fail sensing circuit, or by some other

external logic. A power-fail circuit or external logic connected to the power-fail/stop port DPFS0 L to DPFS7 L can jam a one-byte RST instruction (see section 3.7.2) into this port upon detection of a power failure or in reaction to some external event. This instruction would then be executed to initiate a service routine or sequence.

Activating an external system stop switch would also select the power-fail/stop port; however, in this case, all the data lines into this port would normally be High which is equivalent to a Halt instruction. When the signal PFSEE L is asserted, the power-fail stop port is selected at time state TS3 following entry into time state TS1I which, as shown in Figure 2-3, occurs in the same manner as with a normal interrupt. During time state TS3, the instruction at this port is fetched by the processor for execution.

2.2.4 Output Data Paths

Processor Module output can be in the form of memory addresses, memory and I/O control information, I/O device addresses, data output to memory, and data output to peripherals. With the exception of data output to memory, all of these addresses and data are stored for output in the 16-bit multipurpose output register. Data words are latched into the register by the data selection logic during time states TS1 and TS2 (Figure 2-1).

Memory addresses are issued as two separate words to form a 14-bit address word during machine cycles PCI, PCR, and PCW – the lower word at time state TS1 and the upper byte at TS2 of these cycles. During each one of these time states the corresponding memory address word is loaded into the 16-bit output register by the address selection logic. At the end of time state TS2, the current memory address is present at the output side of the output register. These output lines are buffered to drive the address/data lines ADRD00 L to ADRD13 L which can be bused out to the Microprocessor Series ROMs, RAMs, and I/O devices.

During a PCC cycle, the contents of the accumulator and the instruction register are stored in output register bits 0 through 13 for use as peripheral device output data and address. At time state TS1 of a PCC cycle the content of the accumulator (Register A), which is the data to be output to the addressed peripheral device, is placed in the output register bit positions 0 to 7.

At time state TS2, the content of the instruction register is transferred to output register bit positions 8 through 15, with bits 9 through 13 containing the address of the peripheral device being accessed during the PCC cycle. Once stored in the output register, these address and data fields are available to external peripheral devices over the lines ADRD00 L to ADRD13 L. This device address field permits the addressing of up to eight input devices and 24 output devices.

Data to be written in read/write memory is gated onto the bidirectional memory bus DM0 L to DM7 L during time states TS1 and TS2 of a PCW cycle. Data must be accepted by the memory during time state TS3 of that cycle.

2.2.5 Control Logic

The control logic (Figure 2-1) provides the various control signals necessary to memory accessing and to the performance of input/output operations with associated peripherals. Input to the control logic is the state of bits 14 and 15 of the output register.

During time state TS3 of every machine cycle, the control signals pertinent to the machine cycle currently being executed are asserted. For example, during PCI and PCR cycles where memory is addressed to fetch an instruction or data, the signal MEM RD H is asserted at the end of time state TS2 and throughout TS3. As shown in Figure 2-4, this signal controls the reading of data from that memory location addressed during the pertinent machine cycle.

For a PCC cycle, the signals I/O IN H or I/O OUT H are asserted during time state TS3 to control the storage and retrieval of data at external peripheral devices. When data is to be written into an addressed memory location during a PCW cycle, the signal MEM WR H is asserted by the control logic during time state TS3 of that machine cycle.

In addition to these signals, the control logic also asserts one of the signals PCI L, PCC L, PCR L, or PCW L during time state TS3 of the corresponding machine cycle.

2.2.6 Asynchronous Communications Receiver/ Transmitter Logic

The PM is equipped with a full duplex communication receiver/transmitter implemented by a Universal Asynchronous Receiver/Transmitter (UART). A 1.76 kHz clock, driven by an 844.8 kHz crystal-controlled clock integral to the PM, is input to this device to produce a 110 baud data transfer rate. The UART is addressed during time state TS3 by bits 9 to 13 of the output register. Data is transmitted and received at the module over 20 mA current loop or TTL-compatible lines (USI H and USO H) to interface with Teletype-like lines or to telephone lines through a modem.

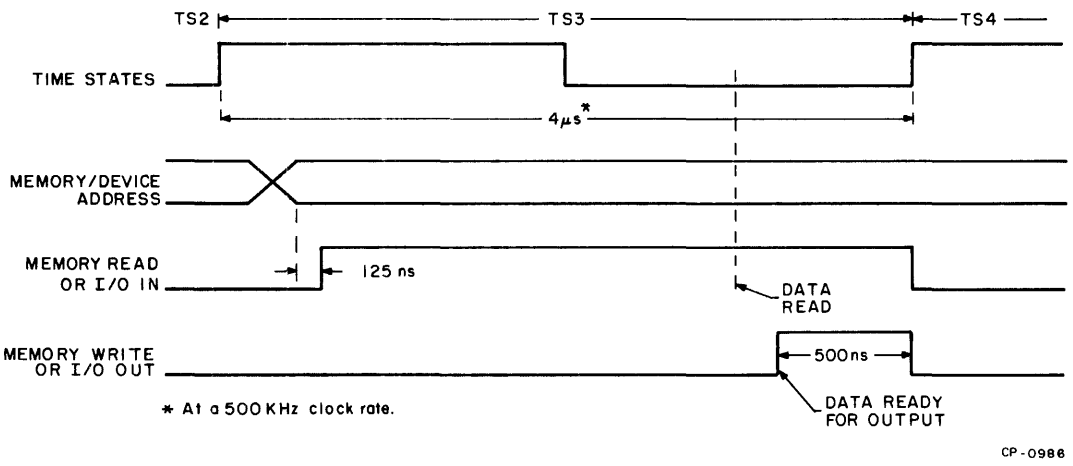


Figure 2-4 I/O Timing Diagram

Higher baud rates can be obtained by using an external clock input (URCLK or UTCLK) that can be derived from external logic which divides the basic crystal clock frequency. The maximum data transfer rate is 9600 baud for TTL lines and 4800 baud for current loop lines (at limited lengths). Internal switches on the PM permit selection of operation under external clock control; the number of stop bits used (one or two) can be selected through switch action. Active or passive operation of current loop lines is jumper-selectable.

Odd parity, even parity, or no parity is selected by the PM input line UPOE. Also, the number of data bits in a word can be selected to vary from five to eight bits. Both TBMT (Transmitter Buffer Empty) and DA (Data Available) are available for external interrupt drive capability.

When an input instruction is being executed, the signal I/O IN H is asserted by the control logic. This signal and the receiver/transmitter device receive address are decoded to gate data from the device onto peripheral bus lines DI0 L to DI7 L and into the peripheral data-in port.

Data for transmission is written from the output register into the UART transmission buffer during time state TS3 of a PCC cycle when the signal I/O OUT H is asserted (Figure 2-4). This signal and the transmit data address are decoded to strobe data from the PM output register into the transmission buffer.

Status information, which includes receiver/transmitter error conditions and transmit and receive buffer status, is retrievable through execution of an input instruction with the assigned status device address. As with reading data from the receiver/transmitter, the signal I/O IN H is asserted as a result of a PCC cycle execution. This signal is gated with the device address to, in turn, gate device status into the peripheral data-in port DI0 L through DI7 L.

2.2.7 Interrupt Control

The interrupt control logic drives the input data multiplexer to select one of two input ports. If the interrupt results from a power failure or a stop command, the power-fail/stop port DPFS0 L to DPFS7 L is selected. Similarly, if the IOEE L line or the system START L line is asserted, the I/O start port DIOST0 L to DIOST7 L is selected. The instruction jammed into these ports as a consequence of a power failure or an I/O or restart interrupt is furnished by the External Event Detection Module or external logic as determined by the specific application.

I/O interrupts can be enabled or disabled under program control by the PM external event enable/disable logic. Interrupts are disabled by executing the instruction IOF and enabled by executing ION (Paragraph 3.7.3).

Since interrupts will be enabled or disabled one instruction time after execution of an ION or IOF, one instruction can

be executed after ION or IOF before interrupts are actually enabled or disabled. The external event recognition logic is automatically disabled after every interrupt.

2.3 READ/WRITE MEMORY MODULE

The M7344 Read/Write Memory Module provides a 1K, 2K, or 4K X 8-bit random access memory capacity along with all necessary timing, control, and decoding logic (Figure 2-5). This module is completely contained on a single quad extended-length board. The module memory matrix is formed by up to 32 1024 X 1-bit static MOS MSI memory circuits. The nature of these MOS circuits precludes the need for external refresh logic.

The M7344 Read/Write Memory Module is available in three versions to satisfy varying memory capacity requirements. Model numbers identifying these memory versions are listed below:

- M7344-YA 1K X 8-bit capacity
- M7344-YB 2K X 8-bit capacity
- M7344-YC 4K X 8-bit capacity

All versions of the M7344 can be accessed by up to 16-bits of address data and are equipped with an address expansion line to implement multi-module memory systems having potential capacities of up to 128K 8-bit words. M7344 Read/Write Memory Modules also contain a jumper network which can be configured to permit assignment of a module within an application-defined address space.

Operation of each memory circuit in either the read mode or write mode is determined by a read/write (R/W) line. Each of the 32 MSI circuits in the memory matrix connect to a data input line and a data output line with the significance of the line corresponding to the position of a circuit with a 1K X 8-bit group. Data input lines are wire-ORed to each memory circuit from the data input register. Similarly, data output lines are wire-ORed to the data output gates from each memory circuit.

2.3.1 Memory Read Timing

The timing and control logic (Figure 2-5) furnishes the signals necessary to time memory read and write operations. The processor module asserts the level MEM RD H after providing the address of the memory location to be

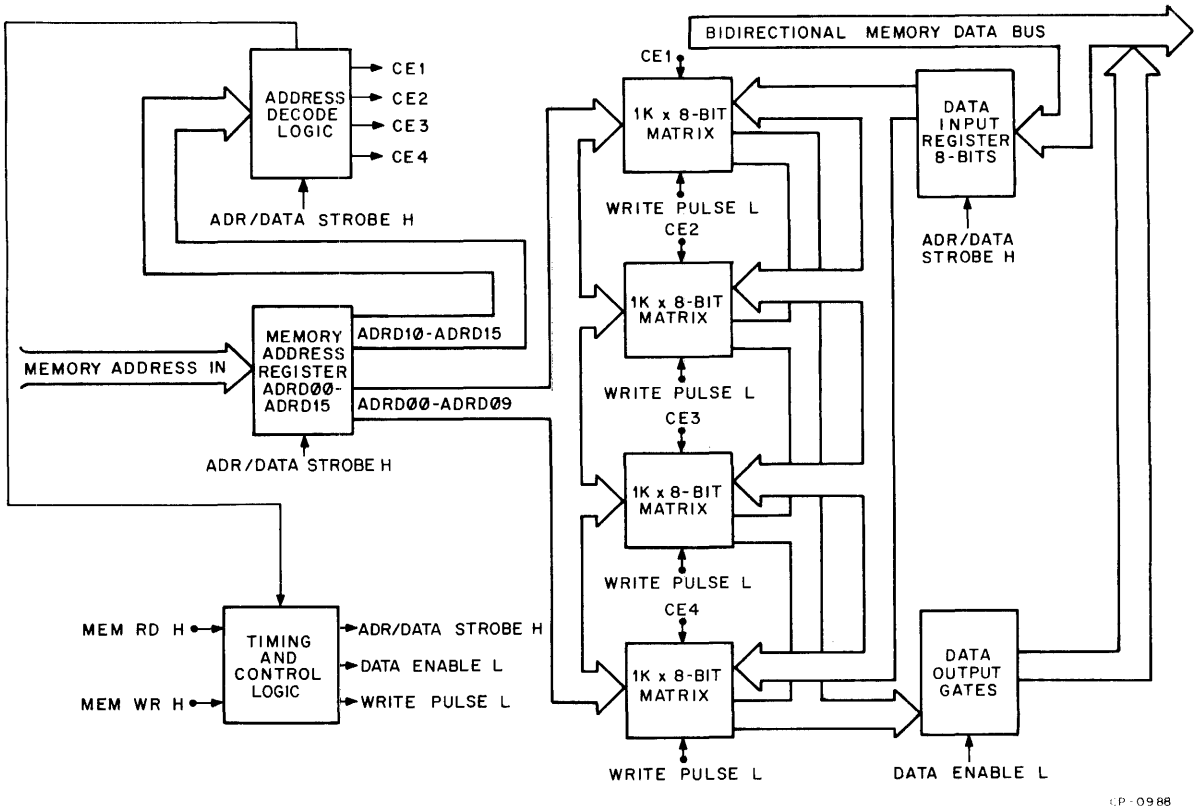


Figure 2-5 M7344 Block Diagram

read. At the read/write memory module, the signal MEM RD H generates the internal signals ADR/DATA STROBE H and DATA ENABLE L.

The signal ADR/DATA STROBE clocks the memory address register to store the address currently on the memory address bus ADRD00 L to ADRD15 L. This action initiates address decoding which enables assertion of DATA ENABLE L along with addressing of the memory location being read. DATA ENABLE L, when asserted, enables the output gating network to place the data from the addressed location onto the bidirectional memory data bus DM0 L to DM7 L. Figure 2-6 shows that data becomes valid on this bus 1.15 μs after the assertion of MEM RD H.

2.3.2 Memory Write Timing

For a write operation, the address of the memory location to be written into is placed on the address bus and must be stable for at least 150 ns prior to the assertion of MEM WR H. Data to be written into the addressed location is placed on the bidirectional memory data bus coincident with the signal MEM WR H.

As shown in Figure 2-6, the signal MEM WR H has a minimum period of 250 ns and is asserted by the processor module. Pulse-stretching circuitry in the M7344 control logic uses the leading edge of MEM WR H to set a latch to store the signal. Approximately 200 ns after the receipt of MEM WR H, this signal is ANDed with the decoded address to generate the internal 1 μs signal WRITE PULSE L. This pulse then enables the data stored in the data input register by the assertion of ADR/DATA STROBE to be written into the addressed memory location.

2.3.3 Address Decoding

Input to the memory address decoding logic is an address loaded into the memory address register from the address bus by the assertion of ADR/DATA STROBE. Both the address bus and the memory address register can accommodate a 16-bit address to permit memory system capacities up to 64K.

As shown in Figure 2-5, the 10 low-order bits (ADRD00 to ADRD09) of the memory address access the same location in each of the four 1K memory segments. The next high-order two bits (ADRD10 and ADRD11) define the

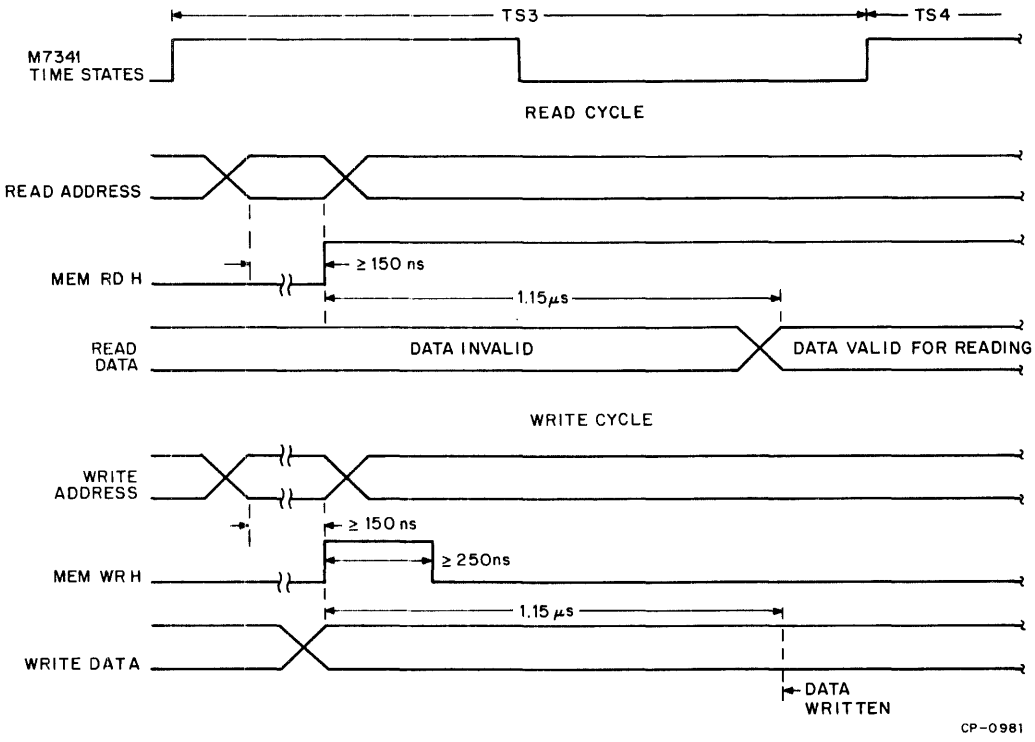


Figure 2-6 Memory Timing Diagram

final magnitude of the addressed location and are decoded by the segment address decoding logic to assert one of the four signals CE1, CE2, CE3, or CE4. These signals enable the 1K segment, which contains the location pointed to by the low-order 12 bits of a 14-bit address. Note that each of the signals CE1, CE2, CE3, and CE4 is jumpered to permit a module to contain multiples of 1K memory locations within the total memory system address space.

A 16-bit address field, all of decoded states of bits ADRD13 L to ADRD15 L, can be jumper-configured. As a consequence, each 4K module in a multiple module memory system can be uniquely jumpered to be assigned as a given set of 4K memory locations within a consecutive set of up to 16K locations.

This jumper network is configured to permit allocation of address space in 1K intervals within an address range of 4K to 8K formed by either one or two M7344 modules. In addition, each module can be assigned an address space of up to 4K within a total 64K address set. Each side of this jumper network is also brought out to the module edge fingers to permit address space allocation to be supplemented by wire wrap on the connector block.

Since the M7344 Read/Write Memory is electrically and logically compatible with the M7345 Programmable Read-Only Memory, these modules can be used together to form a contiguous RAM-PROM memory space.

2.4 PROGRAMMABLE READ-ONLY MEMORY MODULE

The M7345 Programmable Read-Only Memory (PROM) Module provides a variable read-only data storage capacity for systems structured from modules in this family. The functional logic blocks comprising this module (Figure 2-7) consist of plug-in socket space for up to four 1K memory matrices, an address buffer, address and control decoding logic, and an output gating network. Each of these functional blocks is discussed in terms of functions performed and how these functions interrelate. Figure 2-7 also provides a graphic reference for data and signal flow within the module.

2.4.1 Memory Organization

The M7345 PROM module is an 8-bit electrically programmable and erasable read-only memory contained on a single quad board. Maximum PROM capacity is formed by 16 MSI memory circuits mounted in plug-in sockets and organized as 16 separate matrices each containing 256 8-bit words (Figure 2-8). As a consequence, memory capacity can range from 256 to 4K words in 256 word increments. PROM circuits can be removed at will to satisfy changing system requirements and for erasure and reprogramming (Chapter 9).

A transparent quartz lid on each PROM circuit permits exposure to an ultraviolet light source for erasing an existing bit pattern. Then a new bit pattern can be electrically written.

2.4.2 Address and Control Decoding

Address input to the PROM module consists of the full 16 bits of the address bus (ADRD00 L to ADRD15 L) with 14 of these bits (ADRD00 L to ADRD13 L) relevant to the PM. The remaining two bits provide for memory address expansion. As shown in Figure 2-7, the low-order eight bits directly address each PROM circuit in the memory matrix through the address buffer. The remaining bits on the address bus, ADRD08 to ADRD15, are input to the address and control decoding logic which is enabled by the signal MEM RD L. This signal is asserted by the processor module during time state TS3 of a PCI or PCR machine cycle.

Address bits ADRD10 and ADRD11 are decoded to determine the 1K group associated with an addressed location within a 4K group, and bits ADRD08 and ADRD09 are decoded to point to the 256 X 8-bit PROM circuit within that 1K group containing the location being accessed. The result of this decoding is the assertion of one of 16 chip enable signals which causes the addressed PROM circuit to output data from the addressed location within 1.15 μs after the assertion of MEM RD H (Figure 2-9). This data is present at the data output gates. All chip enable signals are wire-ORed to assert the internal signal DATA ENABLE which gates the data at the output data gates onto the bidirectional memory data bus DM0 L to DM7 L. This same logic also asserts the external synchronizing signals MEM SYNC L and DATA READY L.

Bus address bits ADRD12 and ADRD13 are decoded to implement the addressing of multiple module PROM systems having up to 16K locations. Address bits ADRD14 and ADRD15 are decoded to permit expansion of multiple module PROM systems beyond 16K locations up to 64K locations.

Each of the 16 chip enable signals derived as a consequence of decoding address bits ADRD08 through ADRD11 are jumper-connected for assertion to permit depopulation of a given PROM module down to 256 locations by multiples of 256. In addition, the results of decoding address bits ADRD12 and ADRD13 can be jumper-configured to permit the selection of 1K contiguous addresses of PROM memory

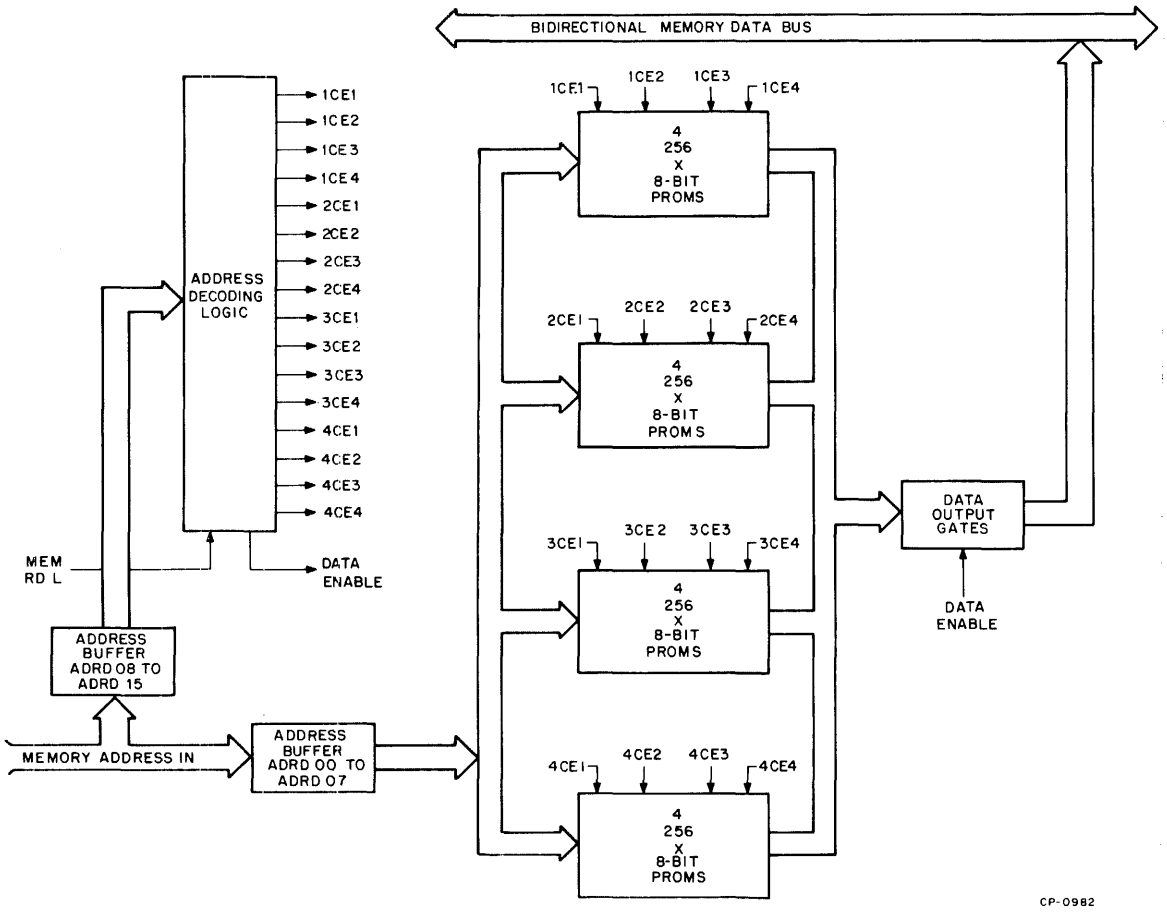


Figure 2-7 M7345 Block Diagram

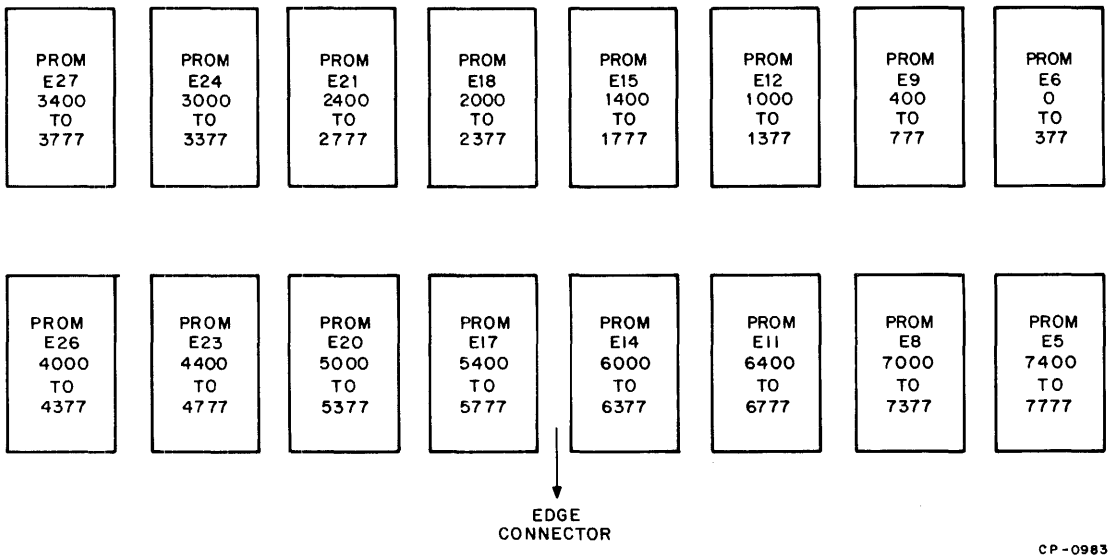


Figure 2-8 Physical Location and Octal Address of M7345 PROMs

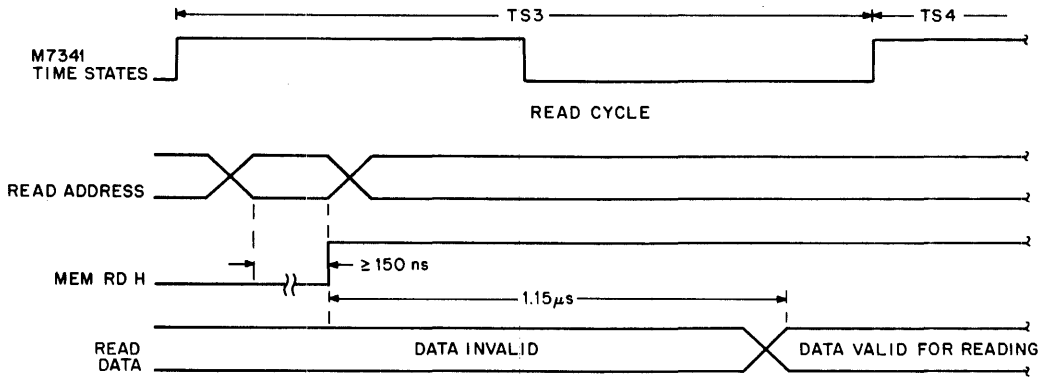


Figure 2-9 Memory Timing Diagram

within 8K location. This feature allows implementing memory systems that require the intermixing of 1K ROM and RAM memory sections within an overall set of contiguous locations.

2.5 EXTERNAL EVENT DETECTION MODULE

The M7346 External Event Detection Module (EEDM) is a multi-purpose Microprocessor Series module designed to implement priority interrupt schemes, provide power failure detection, and processor start/restart/stop control. This module is contained on a single-height, extended-length PC board. Both the priority arbitration logic and the power failure detection circuit are present on this module and can be selected as required, for use in a system.

Separate input lines to the EEDM provide for encoding up to six levels of external application-defined event priority. Each of these lines, when asserted, initiates an attempt to jam a 1-byte unconditional call (RST) instruction into the M7341 Processor Module external event port.

The EEDM priority logic arbitrates all assertions and selects the highest level asserted, then jams the corresponding instruction into the processor module. The jammed RST instruction associated with each priority level (zero through five) constitutes an unconditional call on one of six 8-byte subroutines located in the first 48 words of an MPS system memory.

The eight output lines which propagate the instructions that are jammed, are connected in common to the processor module external event port and the power-fail port.

To jam an RST instruction, the EEDM asserts a signal to enable multiplexing the external event port and to initiate an external event interrupt. If interrupts are enabled at the processor module, the RST instruction is fetched and executed. If not, the RST instruction is ignored.

The seventh priority level is asserted by either a manually initiated start signal or automatically as a consequence of power-up. In either case, an RST instruction is generated which makes an unconditional call on eight reserved memory words headed by location 48 (60_8).

Priority level eight is asserted by the power failure detection circuit which monitors ac power inputs to an MPS system. When a power failure condition is detected, an RST instruction is automatically generated making an unconditional call on location 70_8 (56_{10}). Since level eight is the highest arbitrated priority, a power failure takes precedence over any of the six levels of application-defined event priority as well as the seventh or start level.

Level nine implements the halt function for MPS systems which can be initiated manually or automatically, and overrides all other priorities. When initiated, a halt instruction is placed on the lines to the processor module, and the power fail/stop port is enabled for multiplexing. When fetched, this instruction forces the processor module into the stopped state.

2.5.1 Priority Arbitration Logic

EEDM priority aribtration logic (Figure 2-10) accepts nine levels of ascending priority. Eight of these levels result in a memory reference and the ninth, and highest, is executed by the processor module without referencing memory.

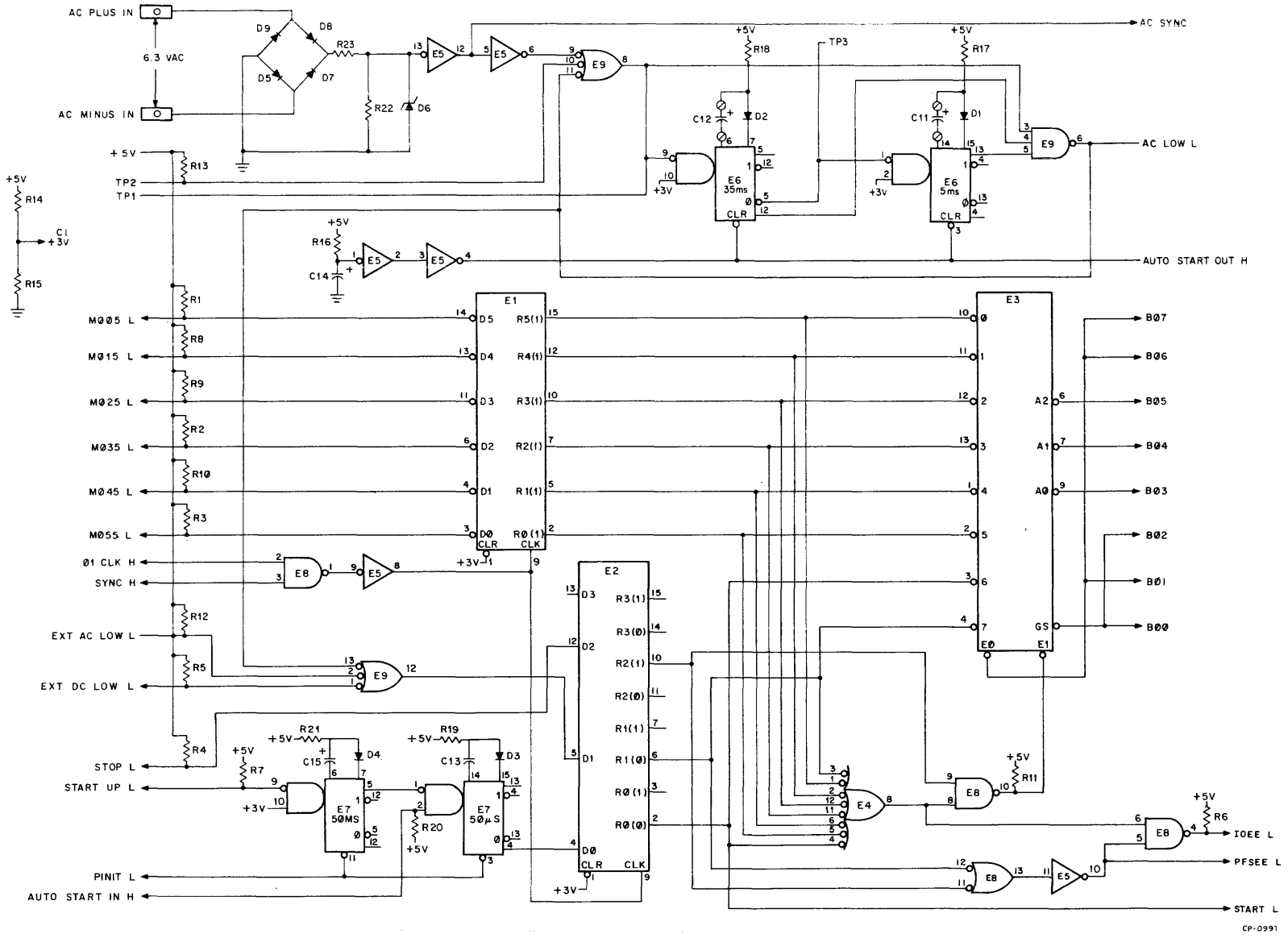


Figure 2-10 M7346 Logic Diagram

Levels 0 through 5, designated by the input signal lines M005 L, M015 L, M025 L, M035 L, M045 L, and M055 L, constitute the six lowest priority levels. These lines are reserved for implementing six levels of application-defined interrupt priority arbitration.

The seventh level is reserved for the automatic and manual restart function, and the eighth and highest arbitrated level, for power failure detection. The ninth, and highest absolute level is reserved for the stop function which automatically jams an HLT instruction into the processor module to stop operation.

EEDM priority arbitration logic is formed by the nine-stage external event storage latch E1 and E2 and the octal encoder E3. Assertions of any of the nine levels are stored during the combined periods of the processor module synchronous timing signals SYNC H and Φ1CLCK H. These signals are ANDed to strobe the external event storage latch E1 and E2 once during each processor time state (every 4 μs) to store asserted events. The output of the first eight stages of the external event storage latch are input to the octal priority encoder E3. These eight input lines are ORed by E3 to enable the encoder. The outputs of the encoder, E0 and GS, are wired to assert the code for an RST instruction on module output lines B00 to B02 and B06 and B07. The octal number corresponding to the external event input line asserted is simultaneously placed in the address field (B03 to B05) of the RST instruction generated by encoder E3.

In parallel with the generation of the RST instruction, the outputs of the first eight stages of latch E1, E2 are ORed by E3 to assert the signal IOEE L at the processor module. This signal, when asserted, generates an interrupt if interrupts are enabled, and initiates multiplexing of the RST instruction on EEDM output lines B00 to B07 at the processor module external event port DIOST0 L to DIOST7 L for fetching.

The octal number placed in the address field of the RST instruction generated is in the range 0 to 7. By decoding this instruction, this value is mapped to address one of the eight locations 10, 20, 30, 40, 50, 60, and 70 (decimal 0, 8, 16, 24, 32, 40, 48, and 56 respectively) in an MPS system memory. The result is a correspondence between a given external event input line and an unconditional call on a dedicated memory location. Through this mechanism, application-defined routines can be developed to implement priority interrupt schemes, start-up routines, and power-failure handling routines.

2.5.2 Start Circuit

The start function can be initiated in two ways: automatically by an integral EEDM circuit, or manually by an external switch. The automatic start output signal AUTO START OUT H is generated by the RC circuit R16, C14, and E5 when +5 V power is turned on and is maintained high as long as +5 V is maintained. The output of this circuit also clears the two monostable multivibrators E6 in the power failure detection circuit. This automatically-generated start signal, when fed back to the EEDM, becomes the input signal AUTO START IN H. When asserted, this signal triggers the monostable multivibrator E7 to produce a 50 μs signal at pin 4. This signal is stored in the seventh stage of the external event latch E1, E2 on the next assertion of φ1CLK H and SYNC H from the processor module. As a consequence, an RST instruction making an unconditional call on memory location 60 (48_{10}) is generated and placed on output lines B00 to B07. Simultaneously, the signal START L is asserted at the processor module to initiate a demand interrupt and to multiplex the RST instruction into the external event port DIOST0 L to DIOST7 L for fetching. Manual restarts are implemented through the EEDM input signal START L which can be derived from an external switch. This signal is debounced by the monostable multivibrator E7 which produces a 50 ms negative-going debouncing level at pin 5 for input to E7 at pin 1. From this point, the circuit path is exactly the same as with an automatic restart. Both automatic and manual start signals take priority over all other external events except power failure detection and stop functions.

2.5.3 Power Failure Detection Circuit

The EEDM contains a complete ac power failure detection circuit (Figure 2-10) which samples a 6.3 V, 50 or 60 Hz, ac input derived from the local line voltage by an external transformer. This sampled ac voltage, which is received through two FASTON tabs on the handle end of the module, is rectified by a full-wave diode bridge to produce a signal having a frequency twice that of the ac input frequency. This signal is input to the frequency integrators E5, E9, and E6 which detect the absence of line voltage for two complete cycles. The signal period triggered by a power failure is approximately 35 ms and is determined by the value of capacitor C12 which, together with R18, forms the RC for the monostable multivibrator E6 at pins 6 and 7. Note that C12 is connected to the circuit with split lugs. This manner of connection permits the value of C12 to be changed to accommodate different application requirements.

When an absence of two or more ac cycles is detected, the multivibrator E6 is triggered at pin 1 to assert a 5 ms signal to be gated out by E9 at pin 6. This period is determined by the RC circuit R17, C11 where capacitor C11 is connected to the circuit with split lugs to allow the value of C11 to be changed to accommodate different application requirements. The 5 ms output signal from the frequency integrator is ORed with the externally generated signals AC LOW L and DC LOW L for input to level seven of the external event storage latch E1, E2. This input is stored in the latch on the assertion of the processor module signals SYNC H and ϕ1CLK H. Since both of these signals are continuous regardless of processor state, storage of a power failure detection signal for fetching by the processor is assured even when the processor is in the Wait state.

The external inputs AC LOW L and DC LOW L allow use of application-defined power failure detection circuitry whose outputs are TTL levels.

NOTE

Many commercially available power supplies provide such output signals.

Power failure detection by the integral EEDM circuit or assertion of one of these external signals takes priority over all external events except assertion of the external signal STOP L.

A detection of power failure results in the automatic generation of an RST instruction which makes an unconditional call on memory location 48 (60_8). Simultaneously, the signal PFSEE L is asserted by E8, pin 13 of the priority arbitration logic to generate a mandatory interrupt at the processor module and to enable multiplexing of the RST instruction into the power-fail port DPFS0 L to DPFS7 L for fetching.

2.5.4 Stop Function

A halt instruction is generated by the EEDM when the input signal STOP L is asserted. When this occurs, the signal is stored in the ninth stage of the external event latch E1, E2 in the same manner as other external events. This stage disables encoder E3 to generate and place a halt instruction (HLT) on the lines B00 to B07.

As with detection of a power failure, the signal PFSEE L is asserted at the processor module power fail/stop port by the EEDM to initiate multiplexing and fetching of the halt instruction.

2.6 MONITOR/CONTROL PANEL

The KC341 Monitor/Control Panel (MCP) interfaces directly with the processor module (PM) over a 50-wire dedicated interface cable to provide an on-line control and program diagnostic capability for systems configured from Microprocessor Series modules. Specifically, the KC341 MCP serves as an address and data input station for the PM to provide a visual display of data as well as display of machine states and PM operating status. The MCP also contains a resident memory formed by a random access scratch pad memory and a programmable read-only memory. Together, these memories provide for program loading as well as other application-defined requirements.

2.6.1 Monitor/Control Panel Cable Connections

Data/Control Interface Cable (BC05-W)

Connect the flat Data/Control Cable between the 50-pin connector on the MCP and the 50-pin connector on side 1 (component side) of the M7341 Processor Module as shown in Figure 2-11. Be sure that the flat cable is not twisted between the two units.

Power Cable (BC05-Y)

The power cable connections to the MCP must be made to the appropriate FASTON tabs as listed below:

Black	Ground
Blue	−15 V
Red	+5 V

2.6.2 Monitor/Control Panel Functions

The discussion that follows relates the functions performed by the MCP to data, address, and signal flow, both within the module and between the MCP and the PM. This discussion is centered on two categories: panel functions and diagnostic memory. Discussions of MC panel functions and the diagnostic memory are based on the detailed block diagrams which graphically depict data address and signal flow as related to the functional logic blocks comprising the monitor/control panel.

In Figure 2-12 the MCP consists of a 14-bit switch register for entry of addresses and data with corresponding display lights, nine function switches, and 12 signal, status and condition display lights. The switch register, together with corresponding bit display lights, is marked off for quick visual observation of octal as well as address blocking notation. The line of function switches located below the switch register is divided into two groups with one group containing seven switches and the other containing two switches. The seven-switch group provides the mechanism

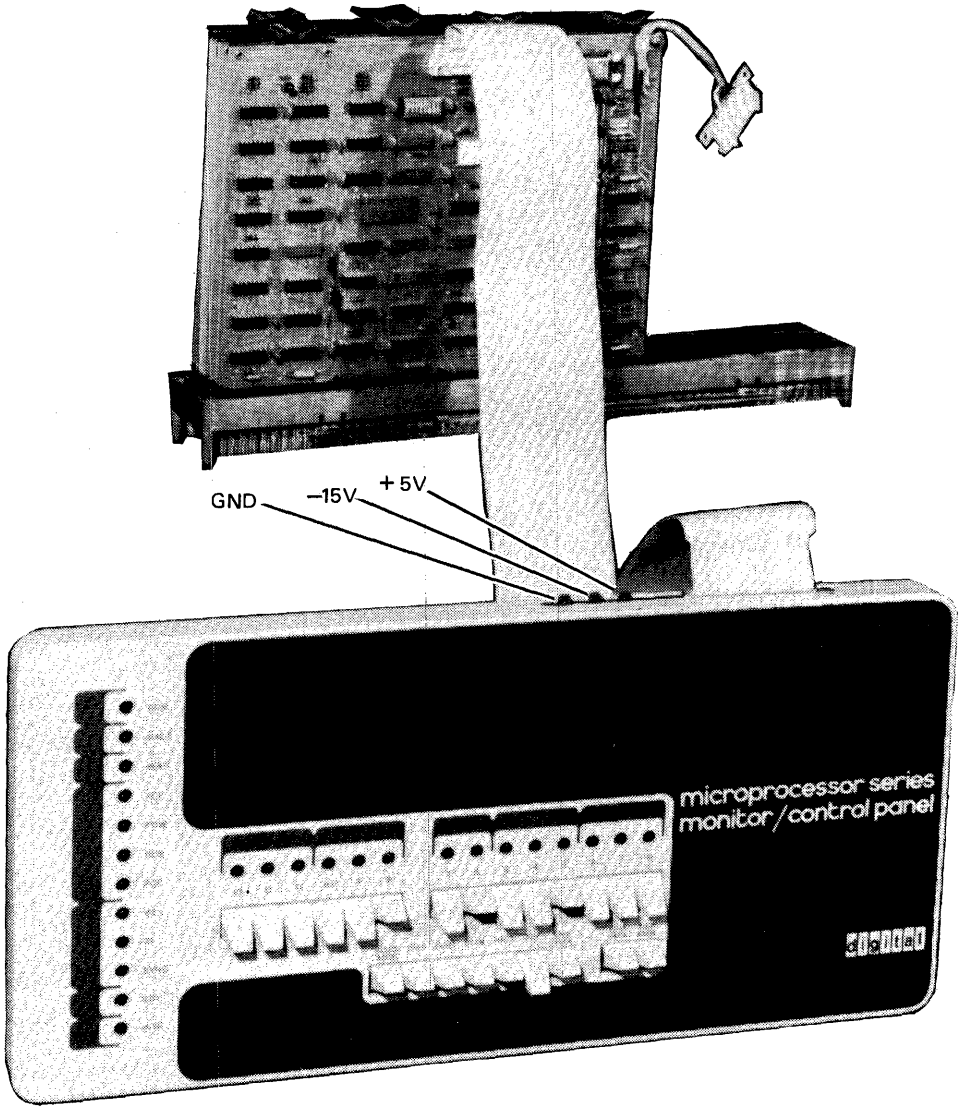


Figure 2-11 MCP Cable Connections

for controlling the input of addresses and data and for monitoring and control of PM operation. The two-switch group controls the display of addresses and data. Each of the seven function switches connects to an implementing logic circuit (Figure 2-13) which performs the action specified by the switch. The ADDR LOAD and DEP switches control address and data entry, with the EXM switch initiating the visual display of the data contained in the memory location accessed by the address entered through the switch register. Data, addresses, and signals pertinent to, or resulting from, these panel actions are received from or sent to the processor module over the 50-wire interface cable.

Use of the ADDR LOAD, DEP, and EXM switches requires that the PM be in the halt state (HLT switch must be on). The remaining four switches permit the on-line control of, and intervention in the operation of the PM. Intervention in this case refers to the single-cycle execution, on a sequential basis, of processor module program instructions performed

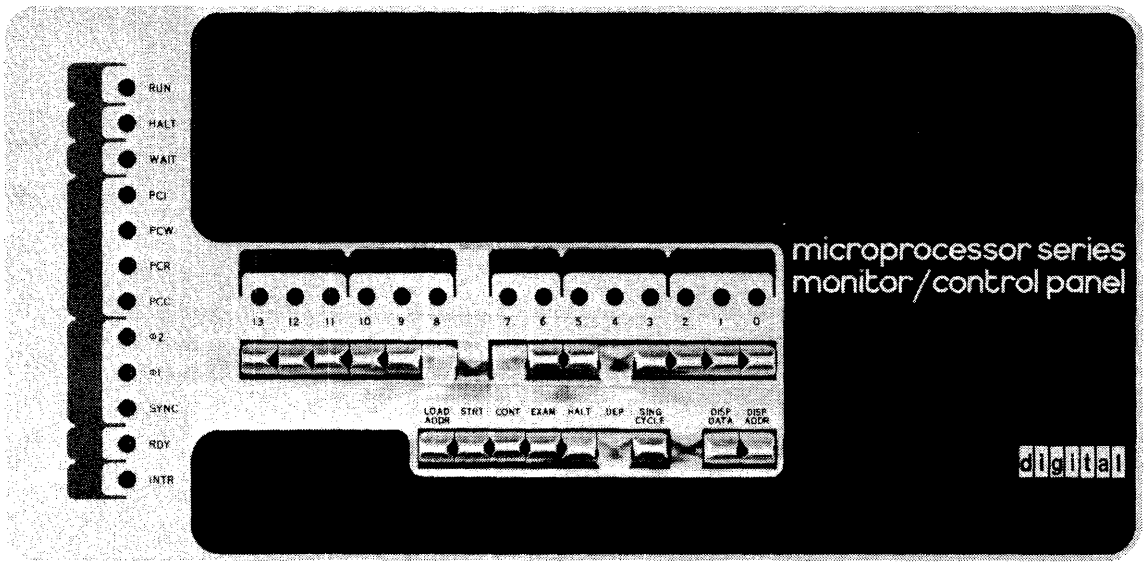


Figure 2-12 MCP Front Panel

by the SING CYCLE and CONT switches. Each of these seven switch functions is discussed in detail with all discussion based on Figure 2-13.

Load Address Function

The load address function permits the manual insertion of a 14-bit memory address through the switch register to deposit data in the location accessed to or examine its content. Once an address has been loaded, it is displayed automatically. Prior to implemeting the load address function, the HLT switch must be on.

Pressing the ADDR LOAD switch causes this switch action to be stored in the load address flip-flop. Flip-flop output is then gated with the signals SSYNC H and ϕ1B H derived from the PM synchronous signals SYNC L and ϕ1, to assert the signal LOAD. Assertion of LOAD causes the manually inserted content of the switch register to be loaded into the address counter. Address counter output is direct input to the address multiplexer. Note that the two most significant bits of the high address byte (ADRD14 H and ADRD15 H) are hardwired to logic ONE (+3 V). This assures that the PM will address memory under the control of a PCI machine cycle so that the memory data-in port at the PM input data multiplexer is selected for subsequent fetching. The signal LOAD is also ORed with the signal COUNT DOWN to clock a second flip-flop whose output is gated with SSYNC H and ϕ2B H to assert LA1. (ϕ2B H is derived from the PM synchronous signal ϕ2.) Note that the signals S0B and S1B control selection of the low and high address bytes for multiplexing onto the 8-bit output data bus. During the period of LA1, S0B is high and S1B, which is the reset side of the flip-flop whose set side is gated with SSYNC H and ϕ2B H to assert LA1, is already low. As a consequence, low address byte is multiplexed onto the data bus.

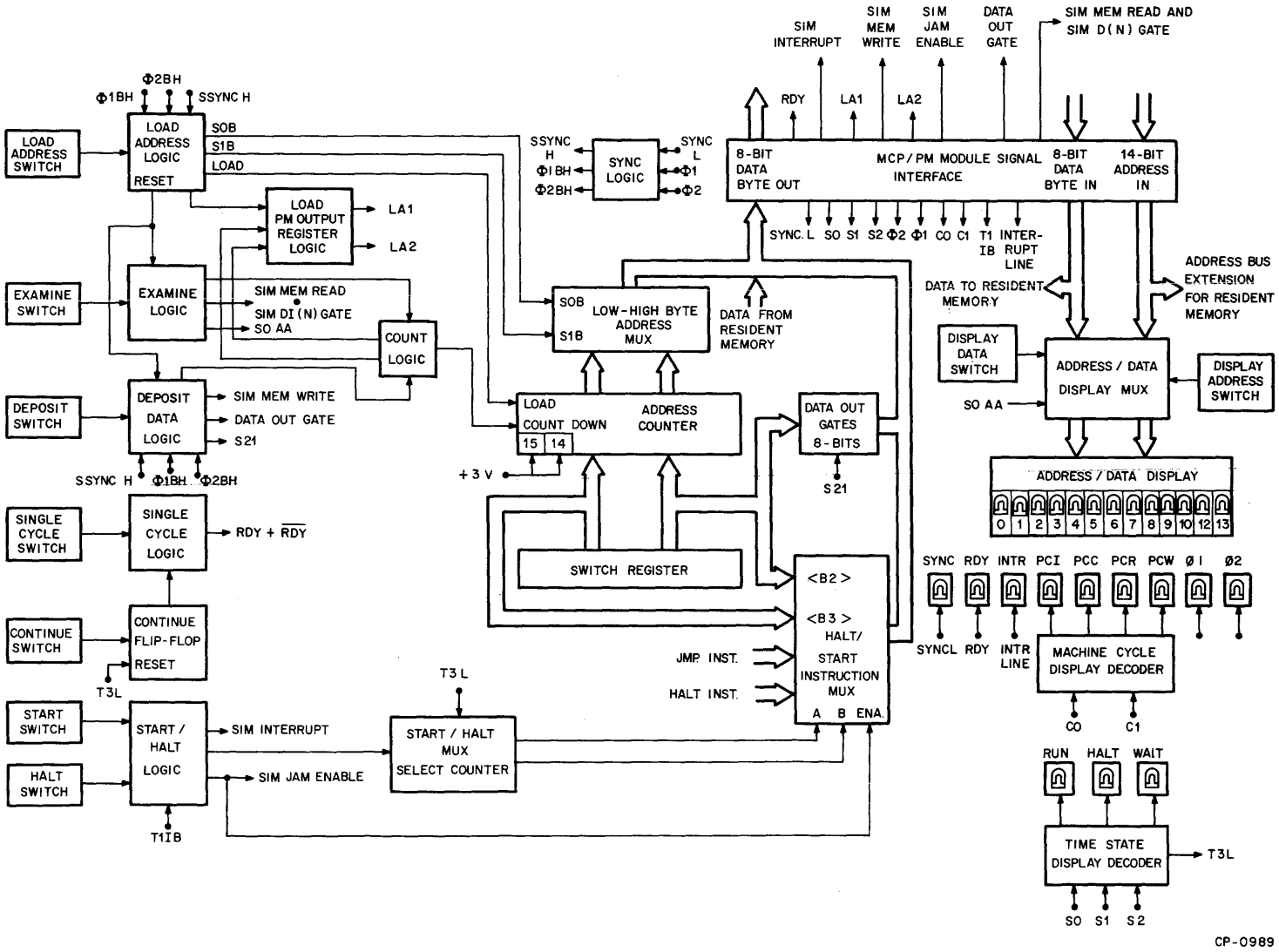


Figure 2-13 MCP Block Diagram

At the PM, LA1 loads this low address byte into the output register (Figure 2-1) as though processor time state TS1 had commenced.

The trailing edge of LA1, as determined by ϕ2B H, clocks a third flip-flop causing S0B to go low along with S1B so that the high address byte is multiplexed onto the output data bus. The set side of this flip-flop is gated with ϕ2B H to assert LA2. At the PM, LA2 causes the high address byte to be loaded into the output register as though the processor time state TS2 had commenced. The assertion of LA2 also initiates a reset cycle to prepare this condition-sensitive logic for the next load address cycle.

At this point, the output register at the PM contains a 14-bit address and this address is present on the address bus to system memory.

Deposit Function

The deposit function permits an 8-bit data byte to be written into the memory location currently accessed by a 14-bit address inserted into the switch register and placed on the PM memory bus by pressing the ADDR LOAD switch. Each data byte inserted into the switch register for deposit can be displayed after being deposited by pressing the DISP DATA button.

Since the content of the address counter is incremented after each deposit, and this new address is placed on the PM memory bus, sets of data bytes can be deposited into consecutive memory locations of ascending magnitude merely by inserting and depositing each byte. The data byte to be deposited is inserted into the eight least significant switches (labeled 0 to 7).

Pressing the DEP switch causes the switch action to be stored by the two deposit flip-flops which are configured to form a two-phase selector. During the first phase, which is selected by the initial pressing of the DEP switch, the resulting signal is stored in an associated flip-flop, then gated with the AND of signals SSYNC H and ϕ1B H. The result of this gating is then stored in a second associated flip-flop to assert the simultaneous signals S21 and DATA OUT GATE. Data inserted into the eight least significant positions of the switch register, in addition to being the low address input to the address multiplexer, is also input to the data out gates.

The assertion of S21, then, gates the 8-bit data byte to be deposited out of the switch register onto the output data bus and to the PM. At the same time, DATA OUT GATE enables the output data gates at the PM to gate this data onto the bidirectional memory data bus.

The assertion of S21 and DATA OUT GATE is gated with the next ϕ2B H pulse to assert SIM MEM WRITE at the PM. As shown in Figure 2-1, this signal activates the PM control logic to assert MEM WR H causing the data on the bidirectional memory bus to be written into memory. If a new address were loaded at this point, the two-phase selector would be reset and the next pressing of the DEP switch would cause the deposit cycle, just discussed, to be repeated.

When the DEP switch is pressed a second time with no intervening address loading or examining, the second phase of the two-phase selector is entered causing assertion of the signal COUNT DOWN. This signal updates the address counter by one and is ORed with the signal LOAD to initiate the load address sequence as described in the discussion under Load Address Function. After a 200 ns delay, to permit gating of the incremented address onto the PM memory bus, a deposit cycle is initiated to write the new data inserted in the switch register into the memory location accessed by the incremented address.

Examine Function

The examine function, as the name implies, permits the examination of the content of that location accessed by the current content of the PM address bus. The content of this location is automatically displayed in the Address/Data display as a result of pressing the EXM switch. The address currently being examined can be displayed by pressing the DISP ADDR button. As with the Load Address and Deposit functions, the processor module must be in the Halt state prior to examining a memory location.

If a sequential set of addresses of ascending magnitude is to be examined, only the starting address need be entered into the MCP switch register. With each subsequent pressing of the EXM switch following the first pressing, the initial address is incremented by one to access the next sequential location for examination.

After an address has been loaded, pressing the EXM switch causes the switch action to be stored by the two examine flip-flops which are configured to form a two-phase selector. During the first phase, which is selected by the initial pressing of the EXM switch, the resulting signal is stored in a third flip-flop to assert the simultaneous signals SIM MEM READ and SIM D (N) GATE at the PM. SIM MEM READ is input to the PM control logic causing that logic to assert the PM signal MEM RD H. As a consequence, the data contained in the location accessed by the MCP (see discussion of Load Address Function) is placed on the memory bus as input to the PM input multiplexer memory

data port DM0 L to DM7 L. Since this port is always selected when the PM is halted, the content of the addressed location is present at the PM input data gates.

These gates are enabled by the signal SIM D (N) GATE placing the data on the input data bus and hence onto the dedicated bidirectional data lines to the MCP. At the MCP, this 8-bit data byte to be examined is input to the eight low-order positions of the Address/Data display. The display is enabled along with the assertion of SIM MEM READ and SIM D (N) GATE, thereby automatically displaying the retrieved data for examination.

When the EXAMINE switch is pressed a second time, the second phase of the two-phase examine selector is entered. During this phase, which is maintained until a new address is manually entered into the switch register, each pressing of the EXM switch asserts the COUNT DOWN signal. The assertion of COUNT DOWN updates the MCP address counter by one, placing the next sequential address to be examined as input to the address multiplexer. COUNT DOWN is also ORed with LOAD to initiate the load address sequence. The load address sequence generates a 200 ns delay to inhibit the examine sequence until the incremented address has been gated onto the data output bus and placed on the PM memory address bus. At the end of this 200 ns delay, the examine sequence is initiated to read and display the data contained in the memory location accessed by the incremented address.

When a new address is entered into the switch register following an examine sequence, the resulting load address sequence will reset all condition-sensitive examine logic including the two-phase selector in the same manner as with the Deposit function.

Single Cycle and Continue Function

The switches implementing these functions, the SING CYCLE and CONT switches, permit the examination on a cycle-by-cycle basis of the PM memory address bus and the bidirectional data port content. Pressing the SING CYCLE switch serves to pull the RDY (Ready) line to the PM to ground causing the processor to enter the Wait state. Entrance into the Wait state by the PM is designated when the WAIT indicator lights. With the processor in the Wait state, each pressing of the CONT switch clocks the continue flip-flop to assert the RDY line at +3 volts causing the PM to escape the Wait state and begin execution starting at time state TS3.

As soon as time state TS3 begins, the states of S0, S1, and S2 at the MCP assert the internal signal T3L. This signal then resets the Continue flip-flop pulling the RDY line back to ground within a time frame which assures that the PM will enter the Wait state following execution of the next time state TS2.

As a consequence each time the CONT switch is pressed, the machine cycle, which is the current constituent of the instruction under execution, is performed starting at PM time state TS3, continues into the next constituent machine cycle, and stops in the Wait state. At that point, the identity of the next machine cycle to be executed will be displayed by the pertinent indicator (PCI, PCC, PCR, or PCW). In addition, the address of the locations containing the bytes constituting the instruction being executed is automatically displayed as each byte is accessed. The actual content of each address can be displayed by pressing the DISP DATA button.

Start Function

The Start function permits an operator to begin executing a program at any location within that program merely by inserting the address of the desired memory location into the MCP switch register and pressing the STRT switch. This address could be, for example, the starting location of the bootstrap routine contained in the MCP PROM resident memory.

Pressing the STRT switch will light the RUN indicator. Use of the Start function is always based on the processor module initially in the halted state. The MCP HALT switch must be in the off position (down) to initiate the Start Function.

Pressing the STRT switch stores the switch action in the start flip-flop causing the output of that flip-flop to assert SIM INTERRUPT at the PM which interrupts the processor. As a result, the processor enteres time state TS1I after completing the current machine cycle. At the start of the next time state TS3 following TS1I, the PM asserts the signal T1IB to the MCP which is gated with the start flip-flop to enable the 2-bit Start/Halt multiplexer select counter and to assert the signal SIM JAM ENABLE to the PM. The signal T1IB represents the first occurrence of time state TS3 at the PM following entry into time state TS1I. The signal SIM JAM ENABLE inhibits the PM input data gates to prevent any extraneous data out of the input data multiplexer from entering the PM bidirectional data port. (Figure 2-1). On the first assertion of TS3 during a start sequence, the initial byte of a JMP (jump unconditionally)

instruction, which is hardwired at the MCP, is selected. This first byte is multiplexed onto the MCP output bus to the PM and directly into the PM data port simultaneously with the assertion of SIM JAM ENABLE. On the second assertion of TS3, the low byte of the jump address (<B2>) previously inserted into the switch register is multiplexed and gated onto the output data bus and into the PM. On the third assertion of TS3, the high byte of the jump address (<B3>) is multiplexed and gated in the same manner. The next instruction executed, which would be the first instruction of a start-up routine, would be fetched from this address. During the jamming of this 3-byte JMP instruction, all system memories including the MCP resident memory are disabled until the jammed instruction has been fetched by the PM.

Halt Function

The Halt function permits a user to arbitrarily halt operation of the processor module through a single switch action. When using one of the panel functions such as load address, examine, or deposit, this switch must be actuated to perform any of these operations.

When the HLT switch is actuated, this action is stored in the Halt flip-flop. The output of this flip-flop then asserts the signal SIM INTERRUPT to the PM thereby interrupting the PM and causing the INTR indicator at the MCP to light. As a consequence of the interrupt, a PCI machine cycle is initiated and T1IB is issued by the PM to set the Start/Halt multiplexer select counter to a zero count thereby selecting the hardwired halt instruction for gating onto the output data bus. Simultaneously, T1IB is gated with start flip-flop output to enable the Start/Halt multiplexer, placing the HLT instruction at the PM data port to be fetched and executed. At that point, the RUN indicator will be extinguished and the HLT indicator will light. All condition-sensitive circuits in the Start/Halt logic are also reset.

2.6.3 Resident Memory

The MCP Resident memory is a semiconductor memory matrix formed by a fully decoded bipolar 32-word X 8-bit random access scratch pad memory (RAM) and an MOS 256-word X 8-bit programmable read-only memory (PROM). In *all* systems configured from Microprocessor Series modules, this memory occupies the last 288 memory locations within a 16K address set regardless of actual system memory size. The octal address set 37340 to 37777 is hardwired-dedicated to the MCP resident memory. The resident memory may be removed from a system and this address space used by a system memory by cutting jumper W1 and installing a soldered wire connection between the adjoining split lugs. Resident memory is discussed under the two memory categories – the RAM and the PROM. Each discussion is based on the block diagram shown in Figure 2-14.

Resident RAM

The Resident RAM is configured for addressing as 16 upper words and 16 lower words. To access this memory, the address decoding logic decodes the state of bus address lines ADRD04 H through ADRD13 H to determine that the current address is in the range 37340_8 to 37357_8 (lower words) or 37360_8 to 37377_8 (upper words).

When one of the lower bytes is addressed, the signal SELECT LOWER RAM is asserted to select the lower word locations for accessing. Similarly, when one of the upper words is addressed, the signal SELECT UPPER RAM is asserted to select upper word locations for accessing. The lines ADRD00 H through ADRD03 H are the four low-order address bits in the 14-bit address and are wire-ORed to all 32 memory locations to address one of 16 locations in both the upper and lower memory word locations. Resident memory timing is performed by the signals MEMORY READ and MEMORY WRITE in conjunction with the signals SYNC L and $\phi 2$ from the PM and T3L as derived at the MCP. Both signals, MEMORY READ and MEMORY WRITE, are derived by the MCP from the signals C0 and C1 from the PM. These signals are asserted simultaneously with the corresponding PM signals MEM RD H and MEM WR H and are, therefore, equivalent.

Two memory-enabling signals are associated with each select signal to implement reading and writing operations. These are: ME UPPER, ME LOWER, WE UPPER and WE LOWER. The signals ME UPPER and ME LOWER are asserted in parallel with the corresponding select signal to initiate a memory-read operation. However, for a write operation, the pertinent select signal is gated with MEMORY WRITE to enable the addressed location for writing of data present on the MCP input data bus. During a read operation, data from the addressed location is gated onto the bidirectional data bus by the signal MEMORY READ.

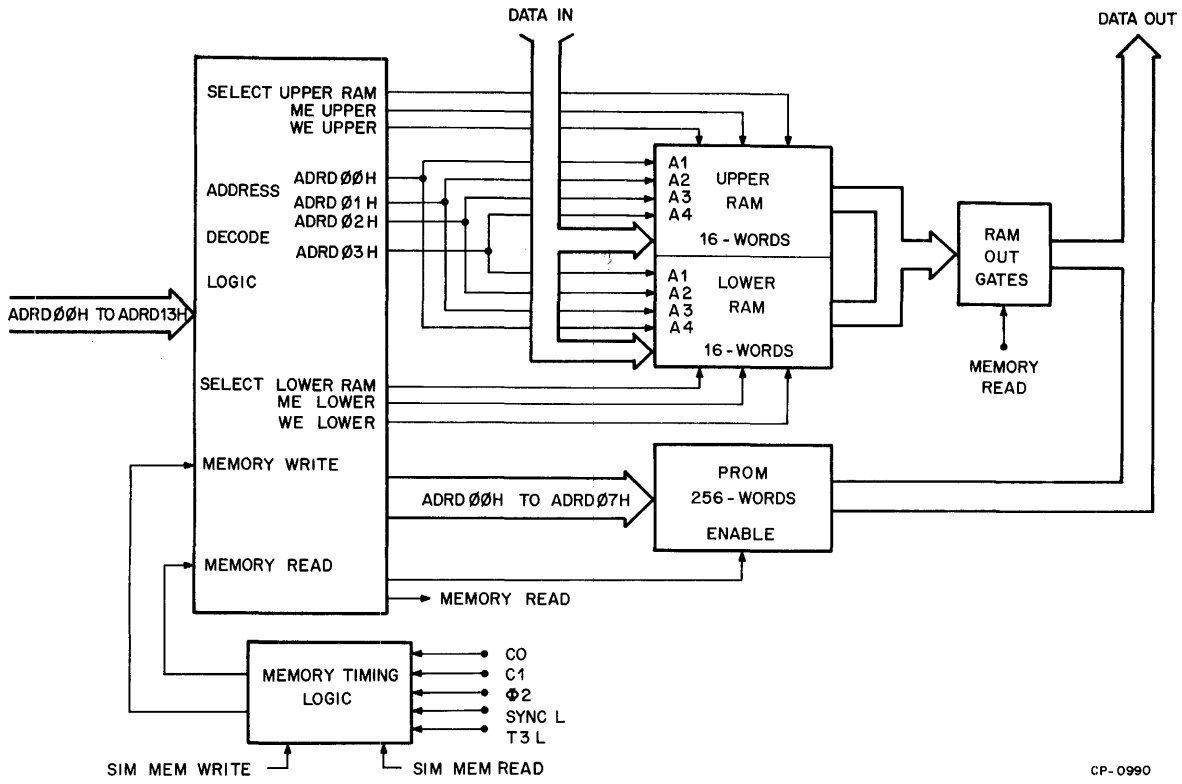


Figure 2-14 MCP Resident Memory Block Diagram

Resident PROM

The Resident PROM (Figure 2-14) is a 256 word X 8-bit electrically programmable and ultra violet erasable read-only memory accessed by octal addresses in the range 37400 to 37777. The signal MEMORY READ is gated with address bits ADRD09 H through ADRD13 H when the magnitude of these bits is decoded to be equal to or greater than 37400_8 to enable the PROM for reading. The content of the location accessed by the eight low-order bits of an asserted address (ADRD00 H to ADRD07 H) is placed on the MCP output data bus as a direct consequence of addressing.

Note that the resident PROM is formed by a single socket mounted dual in-line integrated circuit which contains the Microprocessor Program Loader (MPL). MPL is a bootstrap loader which permits the user to read in object paper tapes from a Teletype.

CHAPTER 3
MICROPROCESSOR SERIES INSTRUCTION SET

3.1 INTRODUCTION

This chapter defines the instruction set for the Microprocessor Series (MPS) M7341 Processor Module, the central control element of the system described in this manual. This instruction set is highly optimized for process control applications. Chapter 3 and Chapter 6 are intended to be used together by system users who develop application programs. Chapter 3 presents the instruction set itself in terms of its graphic and mnemonic representation, the number of bytes in the instruction, the number of time states, the types of machine cycles executed, and examples of program usage. Chapter 6 describes the Microprocessor Language Assembler (MLA) (which assembles the instructions defined in Chapter 3), presents the MLA character set and syntax, describes pseudo-instructions defined for MPS, and summarizes operating procedures and possible error messages.

3.2 INSTRUCTION FUNCTIONS AND FORMATS

Instructions are presented in this chapter in five functional categories:

- Index register instructions
- Accumulator (arithmetic/logical) instructions
- Program counter and stack control instructions
- Input/output instructions
- Machine instructions

Table 3-1 summarizes the conventions used in describing the instruction set, and translates terms used frequently as shorthand descriptions of individual instructions.

The following list of registers and codes applies both to source and destintion registers:

Register	Code
A	000
B	001
C	010
D	011
E	100
H	101
L	110
M (memory addressed by H and L)	111

Data is stored and handled in the form of 8-bit words; all data transfers between registers and memory occur in this format. The instruction syntax shown in the paragraphs which follow includes the number of machine states executed by the instruction. To obtain the amount of time in seconds, use the following computation:

$$\text{seconds} = \frac{1}{\text{clock frequency}} * 2 * \text{state times executed}$$

where clock frequency is expressed as Hz. If the frequency is given in megacycles, the result will be expressed in microseconds.

Table 3-1
Instruction Set Notation

Symbol	Meaning
<B2>	Second byte of an instruction
<B3>	Third byte of an instruction
r r(1) r(2)	One of the 8-bit registers A, B, C, D, E, H, L
A	Register used as the accumulator
B,C,D,E,H,L	Scratchpad registers
H,L	Registers used as memory-address registers
c	One of the status flip-flops (C, Z, S, or P)
C(4)C(3)	Condition flip-flop codes: **Meaning** — **Code** — **Truth Status** Carry (C) — 00 — Overflow, underflow Zero (Z) — 01 — Result is zero Sign (S) — 10 — Most significant bit of result is set Parity (P) — 11 — Number of bits set in result is even
M	Memory location referenced by the contents of registers H and L (code for memory is 111)
( )	Contents of register, memory location, or status flip-flop
$\wedge$	Logical AND
$\forall$	Exclusive OR
V	Inclusive OR
A(m)	Bit m of the accumulator (register A)
stack	Pushdown registers storing nested subroutine return addresses
P	Program counter register containing the address of the next instruction to be executed
←	Is replaced by
XXX	Can be any value
SSS	Source register code
DDD	Destination register code

3.3 INDEX REGISTER INSTRUCTIONS

Index register instructions have been implemented to perform the following functions:

- Load data into index registers or memory
- Load constant immediately after the instruction into index registers or memory
- Increment an index register
- Decrement an index register

The registers maniplulated by these instructions include the following:

- Accumulator or A register
- Scratchpad registers B, C, D, and E
- Memory address registers H and L
- Any addressable read/write or read-only memory location

3.3.1 Loading Data into Index Registers or Memory

Data can be loaded into any of the index or memory registers or can be moved among these registers. Loads of this kind are one-byte instructions, and their execution does not affect the condition flip-flops in any way. Data can be loaded in any of the following ways:

- Load a register with the contents of another register
- Load a register with the contents of a memory location
- Load a memory location with the contents of a register

In all of these instructions, data is loaded from a source (SSS) to a destination (DDD) register; the source register remains intact.

The format for loading a register with the contents of another register is:

Form	Lr(1)r(2) 11 DDD SSS r(1)←r(2)
Examples	LAB LDE
Time States/ Machine Cycles	5,PCI

The contents of r(2), the source register, are transferred to r(1), the destination register. The contents of r(2) remain unchanged. If the source and destination registers are the same, this is considered a NOP (no operation) instruction.

To load a register with the contents of a memory location, the following is issued:

Form	LrM 11 DDD 111 (r)←(M)
Examples	LAM LDM
Time States/ Machine Cycles	8,PCI,PCR

The contents of a memory location (M), addressed by registers H and L, are transferred to r, the destination register. If the code of the destination register is 111, an HLT instruction is executed.

To load a memory location with the contents of a register, use the following instruction:

Form	LMr 11 111 SSS (M)←(r)
Examples	LMA LMC
Time States/ Machine Cycles	7,PCI,PCW

The contents of r, the source register, are transferred to a memory location (M), addressed by registers H and L. The contents of r remain unchanged. If the code of the source register is 111, an HLT instruction is executed.

3.3.2 Loading Data Immediate

These instructions are executed to load the byte of data immediately following the instruction into a register or memory location. Condition flip-flops are not affected. Loads of this kind are two-byte instructions. Data can be loaded as follows:

- Load data into a register
- Load data into a memory location

To load byte two of an instruction into a register, use the following format:

Form	LrI 00 DDD 110 <B2> (r)←<B2>
Examples	LAI A+B LLI 340
Time States/ Machine Cycles	8,PCI,PCR

The data contained in byte two of this instruction will be loaded immediately into r, the destination register.

To load byte two of an instruction into a memory location addressed by the contents of registers H and L, use the following:

Form	LMI 00 111 110 <B2> (M)←<B2>
Example	LMI 104
Time States/ Machine Cycles	9,PCI,PCR,PCW

The data contained in byte two of this instruction will be loaded immediately into M, the memory location addressed by registers H and L.

3.3.3 Incrementing an Index Register

The one-byte instruction is used to increment an index register by one. All condition flip-flops are affected except the carry. Registers B, C, D, E, H, and L can be incremented, but the accumulator (register A) and memory cannot. The instruction format is:

Form	INr 00 DDD 000 (r)←(r)+1
Examples	INB INL
Time States/ Machine Cycles	5,PCI

The contents of r, the destination register, are incremented by one, and the result is stored in r. If the code of the destination register is 000, an HLT instruction is executed.

3.3.4 Decrementing an Index Register

An index register can be decremented by one by means of the one-byte instruction. All condition flip-flops are affected except the carry. Registers B, C, D, E, H, and L can be decremented, but the accumulator and memory cannot. The instruction format is:

Form	DCr 00 DDD 001 (r)←(r)-1
Examples	DCB DCC
Time States/ Machine Cycles	5,PCI

The contents of r, the destination register, are decremented by one, and the result is stored in r. If the code of the destination register is 000, an HLT instruction is executed.

3.4 ACCUMULATOR INSTRUCTIONS

The instructions summarized in this paragraph are used to perform arithmetic, logical, and rotation operations usually between the accumulator and a register or memory location. Accumulator instructions can be divided into the following areas:

- Arithmetic/logical index register instructions
- Arithmetic/logical operations with memory

- Arithmetic/logical immediate instructions
- Rotate instructions

These instructions use the contents of the accumulator as one argument, and an index register, a memory location, or the second byte of the instruction as the other argument. Instructions in this category affect the condition flip-flops in a variety of ways:

1. If a carry or borrow is generated by the instruction, the carry flip-flop (C) is set to one; if no carry or borrow is generated, the carry flip-flop is set to zero.

2. If the result of a comparison with the accumulator is zero, the zero flip-flop (Z) is set to one; if the result of the comparison is nonzero, the zero flip-flop is set to zero.

3. If bit 7 of a result is one, the sign flip-flop (S) is set to one; if bit 7 is not one, the sign flip-flop is set to zero.

4. If a result contains an even number of ones, the parity flip-flop (P) is set to one; if the result contains an odd number of ones, the parity flip-flop is set to zero.

Depending on the specific instruction being executed, one or more of the condition flip-flops can be set as a consequence of instruction execution.

Multiple-precision binary arithmetic is performed using the carry flip-flop; logical operations always reset the carry flip-flop to zero. Rotate instructions affect only the carry flip-flop leaving other condition flip-flops unchanged. Subsequent paragraphs define flip-flop consequences of executing other instructions.

3.4.1 Index Register Instructions

The eight instructions described in this paragraph are used to perform arithmetic and logical operations between the accumulator (register A) and the contents of one of the index registers. The results of the operations affect the accumulator but do not change the contents of any other index register (SSS in boxes below). All of the operations described are one-byte instructions.

To add the contents of a register to the contents of the accumulator, use the following:

Form	ADr
	10 000 SSS
	(A)←(A)+(r)
Examples	ADC
	ADD
Time States/ Machine Cycles	5,PCI

The contents of r, the source register, are added to the contents of the accumulator and the sum is stored in the accumulator. The result of executing this instruction can affect any of the condition flip-flops.

To add the contents of a register and the carry flip-flop to the accumulator, issue the following:

Form	ACr
	10 001 SSS
	(A)←(A)+(r)+(carry)
Examples	ACB
	ACD
Time States/ Machine Cycles	5,PCI

The contents of the source register (r) and the carry flip-flop are added to the contents of the accumulator and the sum is stored in the accumulator. When used in conjunction with the ADr instruction, this instruction facilitates multiple-precision addition of register and accumulator data. Any of the condition flip-flops can be affected by executing this instruction.

To subtract the contents of a register from the contents of the accumulator, use the following:

Form	SUr
	10 010 SSS
	(A)←(A)-(r)
Examples	SUB
	SUE
Time States/ Machine Cycles	5,PCI

The contents of r, the source register, are subtracted from the contents of the accumulator and the difference is stored in the accumulator. Subtraction is performed using two's complement arithmetic. Any of the condition flip-flops can be affected by executing this instruction.

To subtract and borrow use the following:

Form	SBr 10 011 SSS (A)←(A)-(r)-(carry)
Examples	SBB SBD
Time States/ Machine Cycles	5,PCI

The contents of the source register (r) and the carry flip-flop are subtracted from the contents of the accumulator and the difference is stored in the accumulator. Subtraction is performed using two's complement arithmetic. When used in conjunction with the SUr instruction, this instruction facilitates multiple-precision subtraction. Any of the condition flip-flops can be affected by executing this instruction.

To perform a logical AND operation on the contents of the accumulator and a register, use the following:

Form	NDr 10 100 SSS (A)←(A) $\wedge$ (r)
Examples	NDB NDD
Time States/ Machine Cycles	5,PCI

Each bit of r, the source register, is ANDed with each bit of the accumulator. The logical product is stored in the accumulator.

To perform an exclusive OR operation on the contents of the accumulator and a register, use the following:

Form	XRr 10 101 SSS (A)←(A) $\veebar$ (r)
Examples	XRA XRD
Time States/ Machine Cycles	5,PCI

The contents of r, the source register, are exclusively ORed with the contents of the accumulator. The result is stored in the accumulator.

To perform an inclusive OR operation on the contents of the accumulator and a register, use the following:

Form	ORr 10 110 SSS (A)←(A) $\vee$ (r)
Examples	ORA ORB
Time States/ Machine Cycles	5,PCI

Each bit of r, the source register, is ORed with each bit of the accumulator. The result is stored in the accumulator.

To compare the contents of a register with the contents of the accumulator, use the following:

Form	CPr 10 111 SSS (A)-(r)
Examples	CPB CPD
Time States/ Machine Cycles	5,PCI

The contents of r, the source register, are compared with the contents of the accumulator. The accumulator remains unchanged. After the instruction has been executed, if the contents of r are greater than the contents of the accumulator, the carry flip-flop is set to one; if not, it is reset to zero. If the two values are the same, the zero flip-flop is set to one; if not, it is reset to zero. The sign and parity flip-flops are set as if the subtraction had actually occurred.

3.4.2 Operations With Memory

The eight instructions described in this paragraph are used to perform arithmetic and logical operations between the accumulator (register A) and the memory byte of data addressed by the contents of registers H and L. The results of the operations affect the accumulator but do not change the contents of the memory location (M in all models below). All of the operations described are one-byte instructions.

To add the contents of a memory location to the contents of the accumulator, use the following:

Form	ADM 10 000 111 (A)←(A)+(M)
Example	ADM
Time States/ Machine Cycles	8,PCI,PCR

The contents of M are added to the contents of the accumulator and the sum is stored in the accumulator. Any of the condition flip-flops can be affected by executing this instruction.

To add the contents of a memory location and the carry flip-flop to the contents of the accumulator, use the following:

Form	ACM 10 001 111 (A)←(A)+(M)+(carry)
Example	ACM
Time States/ Machine Cycles	8,PCI,PCR

The contents of the specified memory location and the carry flip-flop are added to the contents of the accumulator; the sum is stored in the accumulator (A). When used in conjunction with the ADM instruction, this instruction facilitates multiple-precision addition of memory and accumulator data. Any of the condition flip-flops can be affected by executing this instruction.

To subtract the memory location from the contents of the accumulator, use the following:

Form	SUM 10 010 111 (A)←(A)-(M)
Example	SUM
Time States/ Machine Cycles	8,PCI,PCR

The contents of the specified memory location are subtracted from the accumulator and the difference is stored in the accumulator. Subtraction is performed using two's complement arithmetic. Any of the condition flip-flops can be affected by executing this instruction.

To subtract the contents of a memory location and the carry flip-flop from the contents of the accumulator, use the following:

Form	SBM 10 011 111 (A)←(A)-(M)-(carry)
Example	SBM
Time States/ Machine Cycles	8,PCI,PCR

The contents of the specified memory location and the carry flip-flop are subtracted from the contents of the accumulator. Subtraction is performed using two's complement arithmetic. When used in conjunction with the SUM instruction, this instruction facilitates multiple-precision subtraction. Any of the condition flip-flops can be affected by executing this instruction.

To perform a logical AND operation on the contents of the accumulator and the memory location use the following:

Form	NDM 10 100 111 (A)←(A) $\wedge$ (M)
Example	NDM
Time States/ Machine Cycles	8,PCI,PCR

Each bit of the memory location is ANDed with each bit of the accumulator. The logical product is stored in the accumulator.

To perform an exclusive OR operation on the contents of the accumulator and the memory location, use the following:

Form	XRM 10 101 111 (A)←(A) $\veebar$ (M)
Example	XRM
Time States/ Machine Cycles	8,PCI,PCR

The contents of M are exclusively ORed with the contents of the accumulator and the result is stored in the accumulator.

For an inclusive OR operation on the contents of the accumulator and the memory location, issue the following:

Form	ORM 10 110 111 (A)←(A) $\vee$ (M)
Example	ORM
Time States/ Machine Cycles	8,PCI,PCR

The contents of M are inclusively ORed with the contents of the accumulator and the result is stored in the accumulator.

To compare the contents of the memory byte with the contents of the accumulator, use the following:

Form	CPM 10 111 111 (A)-(M)
Example	CPM
Time States/ Machine Cycles	8,PCI,PCR

The contents of M are compared with the contents of the accumulator; the accumulator remains unchanged. After the instruction has been executed, if the contents of M are greater than the contents of the accumulator, the carry flip-flop is set to one; if not, it is reset to zero. If the two values are equal, the zero flip-flop is set to one; if not, it is reset to zero. The sign and parity flip-flops are set as if the subtraction had actually occurred.

3.4.3 Immediate Instructions

The eight instructions described in the paragraph are used to perform arithmetic and logical operations between the accumulator and the byte of data immediately following the instruction. The results of the operations described below affect the accumulator but do not change the contents of the immediate byte. All of the operations described are two-byte instructions.

To add the contents of byte two to the contents of the accumulator, use the following:

Form	ADI 00 000 100 <B2> (A)←(A)+<B2>
Example	ADI 2
Time States/ Machine Cycles	8,PCI,PCR

The second byte of this instruction, <B2>, is added to the contents of the accumulator and the sum is stored in the accumulator. Any of the condition flip-flops can be affected by executing this instruction.

To add byte two and the carry flip-flop to the contents of the accumulator, issue the following:

Form	ACI 00 001 100 <B2> (A)←(A)+<B2>+(carry)
Example	ACI 104
Time States/ Machine Cycles	8,PCI,PCR

The second byte of this instruction (<B2>) and the carry flip-flop are added to the contents of the accumulator and the sum is stored in the accumulator. In conjunction with the ADI instruction this instruction facilitates multiple-precision addition of instruction and accumulator data. Any of the condition flip-flops can be affected by executing this instruction.

To subtract byte two from the contents of the accumulator, use the following:

Form	SUI 00 010 100 <B2> (A)←(A)-<B2>
Example	SUI 1
Time States/ Machine Cycles	8,PCI,PCR

The second byte of this instruction, <B2>, is subtracted from the contents of the accumulator and the difference is stored in the accumulator. Subtraction is performed using two's complement arithmetic. Any of the condition flip-flops can be affected by executing this instruction.

To subtract byte two and the carry flip-flop from the accumulator, use the following:

Form	SBI 00 011 100 <B2> (A)←(A)-<B2>-(carry)
Example	SBI 6
Time States/ Machine Cycles	8,PCI,PCR

The second byte of this instruction, (<B2>), and the carry flip-flop are subtracted from the contents of the accumulator and the difference is stored in the accumulator. Subtraction is performed using two's complement arithmetic. In conjunction with the SUI instruction, this instruction facilitates multiple-precision subtraction.

To perform a logical AND operation on the contents of the accumulator and byte two, use the following:

Form	NDI 00 100 100 <B2> (A)←(A) $\wedge$ <B2>
Example	NDI 100
Time States/ Machine Cycles	8,PCI,PCR

The second byte of this instruction, <B2>, is ANDed with the contents of the accumulator. The logical product is stored in the accumulator.

To perform an exclusive OR operation on the contents of the accumulator and byte two, use the following:

Form	XRI 00 101 100 <B2> (A)←(A) $\veebar$ <B2>
Example	XRI 340
Time States/ Machine Cycles	8,PCI,PCR

The second byte of the instruction, <B2>, is exclusively ORed with the contents of the accumulator and the result is stored in the accumulator.

For an inclusive OR operation on the contents of the accumulator and byte two, issue the following:

Form	ORI 00 110 100 <B2> (A)←(A) $\vee$ <B2>
Example	ORI 102
Time States/ Machine Cycles	8,PCI,PCR

The second byte of the instruction, <B2>, is inclusively ORed with the contents of the accumulator and the result is stored in the accumulator.

To compare the contents of byte two with the contents of the accumulator, use the following:

Form	CPI 00 111 100 <B2> (A)-<B2>
Example	CPI 4
Time States/ Machine Cycles	8,PCI,PCR

The second byte of the instruction, <B2>, is compared with the contents of the accumulator; the accumulator remains unchanged. After the instruction has been executed, if the contents of <B2> are greater than the contents of the accumulator, the carry flip-flop is set to one; if not, it is reset to zero. If the two values are the same, the zero flip-flop is set to one; if not, it is reset to zero. The sign and parity flip-flops are set as if the subtraction had actually occurred.

3.4.4 Rotate Instructions

The four instructions described in this paragraph are used to rotate the contents of the accumulator, one bit per instruction execution, in one of the following ways:

- Left and into the carry flip-flop
- Right and into the carry flip-flop
- Left and through the carry flip-flop
- Right and through the carry flip-flop

These instructions affect only the carry flip-flop; all other condition flip-flops remain unchanged. All rotates described are one-byte instructions.

To rotate the contents of the accumulator one bit to the left and into the carry flip-flop, use the following:

Form	RLC 00 000 010 A(m+1)←A(m) A(0)←A(7) (carry)←A(7)
Example	RLC
Time States/ Machine Cycles	5,PCI

The contents of the accumulator are rotated to the left by one bit. Bit 7 is moved to the bit position of bit 0, and bits 0 through 6 are moved to bit positions 1 through 7. Bit 7 is also stored in the carry flip-flop. The following diagrams show the bit positions before and after the rotate. The original contents of the accumulator are as follows:

0	0	0	0	0	0	1	0
7	6	5	4	3	2	1	0

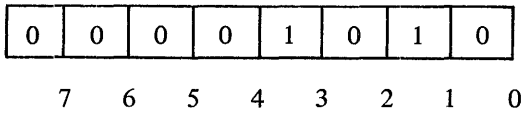
After the rotate, the following is the case:

	0	0	0	0	0	1	0	0
carry	7	6	5	4	3	2	1	0

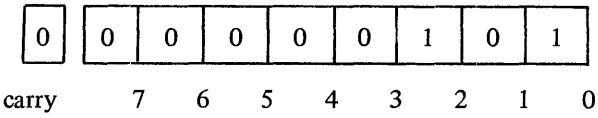
To rotate the contents of the accumulator one bit to the right and into the carry flip-flop, use the following:

Form	RRC 00 001 010 A(m)←A(m+1) A(7)←A(0) (carry)←A(0)
Example	RRC
Time States/ Machine Cycles	5,PCI

The contents of the accumulator are rotated to the right by one bit. Bit 0 is moved to the bit position of bit 7, and bits 7 through 1 are moved to bit positions 6 through 0. Bit 0 is also stored in the carry flip-flop. The original position is:



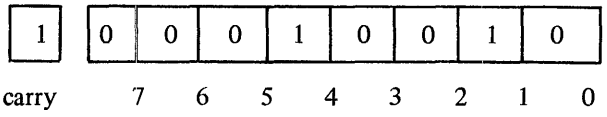
The next diagram represents the contents of the accumulator after an RRC:



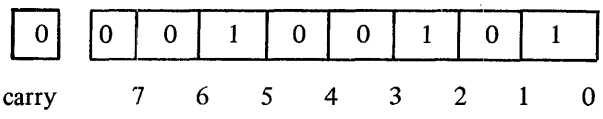
To rotate the contents of the accumulator one bit to the left and through the carry flip-flop, use the following:

Form	RAL 00 010 010 A(m+1)←A(m) A(0)←(carry) (carry)←A(7)
Example	RAL
Time States/ Machine Cycles	5,PCI

The contents of the accumulator are rotated to the left by one bit. Bit 7 is stored in the carry flip-flop; the contents of the carry flip-flop are stored in A(0). Bits 0 through 6 are moved to bit positions 1 through 7. The original contents of the accumulator and the carry flip-flop are:



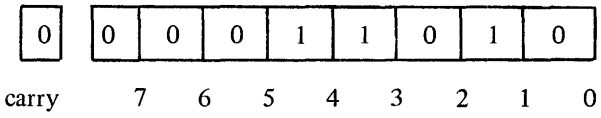
After the rotate, the contents are:



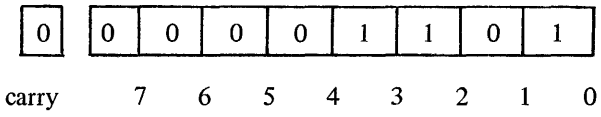
To rotate the contents of the accumulator one bit to the right and through the carry flip-flop, use the following:

Form	RAR 00 011 010 A(m)←A(m+1) A(7)←(carry) (carry)←A(0)
Example	RAR
Time States/ Machine Cycles	5,PCI

The contents of the accumulator are rotated to the right by one bit. Bit 0 is stored in the carry flip-flop; the contents of the carry flip-flop are stored in bit 7. Bits 7 through 1 are moved to bit positions 6 through 0. The original contents of the accumulator and the carry flip-flop are:



After the rotate, the following is the case:



3.5 PROGRAM COUNTER AND STACK CONTROL INSTRUCTIONS

The instructions summarized in this paragraph are used to transfer control from one address to another. These instructions are divided into three categories:

- Jump instructions
- Call instructions
- Return instructions

All of these instructions make use of the processor stack and/or program counter. The stack is a set of eight 14-bit registers. Seven of these registers are organized as a last-in first-out (LIFO) pushdown stack which is used to store nested subroutine addresses. The eighth register serves as the current program counter (P) and always contains the address of the next instruction to be fetched.

Each of the three instruction types described uses the stack or program counter in a different way. The second and third bytes, <B2> and <B3>, of each *jump instruction* (JMP, JFc, JTc) contain an address. This address points to the memory location from which the next instruction is to be fetched if the jump is an unconditional one or if the conditions of the jump are satisfied. The address contained in the jump instruction is stored in the program counter (P) for use during execution. <B2> and <B3> are 8-bit bytes which form the 14-bit address to be stored. <B2> contains the eight low-order bits of the address, and <B3> contains the six high-order bits. Bits 6 and 7 of <B3> are not used.

Subroutine call instructions (CAL, CFc, CTc) store the current program counter (P) in the pushdown stack and then store the starting address of the subroutine to be called in the program counter. This address is contained in the second and third bytes of the call instruction. Calls can be performed unconditionally in a CAL instruction, or conditionally in the case of CFc and CTc.

Subroutine return instructions (RET, RFc, RTc) cause the current program counter to be replaced by the last inserted address in the stack, and for all remaining addresses in the stack to be "popped up" one level. These actions can be performed unconditionally in the case of a RET instruction, or conditionally in the case of RFc and RTc. Because the stack is an eight-register LIFO pushdown stack, is popped up one level at a time, and has a program counter as its eighth register, subroutines can be nested to seven levels.

With conditional jumps, calls, and returns, any of the four condition flip-flops (carry, zero, sign, and parity) can be tested to determine the condition on which the instruction will be executed.

Each of the instruction groups introduced is explained in detail. The following example shows a call to a subroutine, a conditional jump, and an unconditional return.

3.5.1 Jump Instructions

Jump instructions are three-byte instructions which are used to alter the normal flow of a program by branching conditionally or unconditionally to another location. The address to which control is passed is specified by bytes two and three of the jump instruction. Byte two (<B2>) contains the eight low-order bits of the address and byte three (<B3>) contains the six high-order bits. Because the processor uses a 14-bit address, bits six and seven of <B3> are ignored. There are three jump instructions:

- Jump unconditionally
- Jump if condition is false
- Jump if condition is true

To perform an unconditional jump, issue the following:

Form	JMP
	01 XXX 100
	<B2>
	<B3>
	(P)←<B3><B2>
Examples	JMP CKDONE
	JMP AL
Time States/ Machine Cycles	11,PCI,PCR,PCR

<B2> and <B3> make up a 14-bit address which is stored in the program counter to initiate an unconditional transfer of program control to that address. <B2> and <B3> therefore represent the next instruction to be executed after the JMP.

```
NXTBLK, CAL     GETBYT
        .
        .
/SUBROUTINE TO GET ONE BYTE OF DATA FROM PAPER TAPE
/AND UPDATE CHECKSUM
GETBYT, INP1                    /INPUT STATUS
        NDI     40              /MASK "DA" (IGNORE ERRORS)
        JTZ     GETBYT          /WAIT FOR "DA"
        INP0                    /GET CHAR
        LEA                     /SAVE CHAR
        ADD                     /ADD CHECKSUM TO INPUT BYTE
        LDA                     /SAVE NEW CHECKSUM
        LAE                     /RESTORE CHAR
        RET                     /RETURN UNCONDITIONALLY
```

To jump on a false condition use the following:

Form	JFc
	01 0C(4)C(3) 000 <B2> <B3>
	If (c)=0 (P)←<B3><B2> Otherwise (P)=(P)+3
Examples	JFZ NXTBLK JFP B2
Time States/ Machine Cycles	If (c)=0 11,PCI,PCR,PCR Otherwise 9,PCI,PCR,PCR

If the condition flip-flop represented by c is false (reset to zero), the address specified by <B3> <B2> is stored in the program counter and the next instruction to be executed is fetched from this address. If the relevant status flip-flop is true (set to one), the program counter is incremented by three and the branch is not taken. In this case, time states T4 and T5 of the second PCR cycle are skipped. For example, if instruction JFC is executed, a branch will occur if the carry condition flip-flop is zero.

To jump on a true condition, use the instruction following:

Form	JTc
	01 1C(4)C(3) 000 <B2> <B3>
	If (c)=1 (P)←<B3> <B2> Otherwise (P)=(P)+3
Examples	JTS ER JTC DONE
Time States/ Machine Cycles	If (c)=1 11,PCI,PCR,PCR Otherwise 9,PCI,PCR,PCR

If the condition flip-flop represented by c is true (set to one), the address specified by <B3> <B2> is stored in the program counter and the next instruction to be executed is fetched from this address. If the relevant condition flip-flop is false (reset to zero), the program counter is incremented by three and the branch is not taken. In this case, time states T4 and T5 of the second PCR cycle are skipped. For example, if instruction JTP is executed, a branch will occur if the parity condition flip-flop is set.

3.5.2 Call Instructions

Call instructions are three-byte instructions which are used to alter the normal flow of a program by branching conditionally or unconditionally to a subroutine. Subroutine calls may be nested to seven levels. Byte two (<B2>) contains the low-order eight bits of the address, and byte three (<B3>) contains the high-order six bits. Bits six and seven of byte three are ignored. A call causes <B3> <B2> to be stored in the current program counter (P) and for the previous contents of P to be inserted at the top of the pushdown stack. There are three call instructions:

- Call unconditionally
- Call if condition is false
- Call if condition is true

To perform an unconditional call to a subroutine, issue the following:

Form	CAL 01 XXX 110 <B2> <B3>
	(stack)←(P) (P)← <B3> <B2>
Examples	CAL GETBYT CAL D0A1
Time States/ Machine Cycles	11,PCI,PCR,PCR

The contents of P are shifted into the stack and the starting address of the subroutine, <B3> <B2>, is stored in the program counter. This causes the next instruction executed to be the starting address of the subroutine whose name is included in the CAL.

To perform a call on a false condition, use the instruction following:

Form CFc

01 0C(4)C(3) 010
<B2>
<B3>

If (c)=0
(stack)←(P)
(P)← <B3> <B2>
Otherwise
(P)=(P)+3

Examples CFZ ALL
CFP ADDUP

Time States/ Machine Cycles If (c)=0
11,PCI,PCR,PCR
Otherwise
9,PCI,PCR,PCR

If the condition flip-flop represented by c is false (reset to zero), the address specified by <B3> <B2> is stored in the program counter and the next instruction to be executed is fetched from this address. If the relevant condition flip-flop is true (set to one), the program counter is incremented by three and the subroutine is not called. In this case, time states T4 and T5 of the second PCR cycle are skipped. For example, if instruction CFS is executed, a subroutine call will be issued if the sign condition flip-flop is not set.

To call a subroutine on a true condition, issue the following:

Form CTc

01 1C(4)C(3) 010
<B2>
<B3>

If (c)=1
(stack)←(P)
(P)← <B3> <B2>
Otherwise
(P)=(P)+3

Examples CTS CKDONE
CTC CX

Time States/ Machine Cycles If (c)=1
11,PCI,PCR,PCR
Otherwise
9,PCI,PCR,PCR

If the condition flip-flop represented by c is true (set to one), the address specified by <B3> <B2> is stored in the program counter and the next instruction to be executed is fetched from this address. If the relevant condition flip-flop is false (reset to zero), the program counter is incremented by three and the subroutine is not called. In this case, time states T4 and T5 of the second PCR cycle are skipped. For example, if instruction CTZ is executed, a subroutine call will be issued if the zero condition flip-flop is set.

3.5.3 Return Instructions

The instructions described in this section are one-byte instructions which are used to exit unconditionally or conditionally from a subroutine entered via a call instruction and to return to the next sequential instruction after the call. Returns cause the pushdown stack to be popped up one level at a time. The popped entry in the pushdown stack is stored in the program counter (P). The following example illustrates both an unconditional and a conditional subroutine return:

```
        CAL     INCHL
        .
        .
INCHL,  INL             /INCREMENT LOW BYTE OF MEMORY ADDRESS
        RFZ             /RETURN IF NO OVERFLOW
        INH             /OVERFLOW-INCREMENT HIGH BYTE
        RET             /RETURN
```

To perform an unconditional return from a called subroutine, issue the following:

Form	RET
	00 XXX 111
	(P)←(stack)
Example	RET
Time States/ Machine Cycles	5,PCI

The stack is popped up one level and the popped entry in the pushdown stack is stored in the program counter. P now points to the next instruction after the call.

To perform a return on a false condition, use the following:

Form	RFc
	000C(4)C(3) 011
	If (c)=0 (P)←(stack) Otherwise (P)=(P)+1
Example	RFZ
Time States/ Machine Cycles	If (c)=0; 5,PCI Otherwise; 3,PCI

If the condition flip-flop represented by c is false (reset to zero), the stack is popped up one level and the popped entry in the pushdown stack is stored in the program counter. P now points to the next instruction after the call. If the relevant condition flip-flop is true (set to one), the program counter is incremented by one and the return is not performed. In this case, time states T4 and T5 are skipped. For example, if instruction RFP is executed, a return will be issued if the parity condition flip-flop is not set.

To return on a true condition, issue the following:

Form	RTc
	00 1C(4)C(3) 011
	If (c)=1 (P)←(stack) Otherwise (P)=(P)+1
Example	RTS
Time States/ Machine Cycles	If (c)=1; 5,PCI Otherwise; 3,PCI

If the condition flip-flop represented by c is true (set to one), the stack is popped up one level and the popped entry in the pushdown stack is stored in the program counter. P now points to the next instruction after the call. If the relevant condition flip-flop is false (reset to zero), the program counter is incremented by one and the return is not performed. In this case, time states T4 and T5 are skipped. For example, if instruction RTZ is executed, a return will be issued if the zero condition flip-flop is set.

3.6 INPUT/OUTPUT INSTRUCTIONS

The one-byte instructions described in this section are used to perform input or output operations. With these instructions, data can be transferred between the accumulator (register A) of the Processor Module and any peripheral device associated with the system. The data transfer is performed in 8-bit bytes at the rate of one byte per I/O instruction executed. It is possible to access eight different input devices by specifying the appropriate address field in the INP instruction. By supplying the proper address in an OUT instruction, 24 different output devices can be accessed.

The states of the condition flip-flops are not affected by executing the I/O instructions described. Because INP moves data into the accumulator from an input device and OUT moves data from the accumulator to an output device, it is the programmer's responsibility to load data into register A before issuing an OUT instruction, and to extract data from A after executing an INP.

3.6.1 Input Instruction

To read one byte of data into the accumulator (register A) from an input device, use the following:

Form	INP
	01 00M MM1
	(A)←(input data lines)
Example	INP + 10*
Time States/ Machine Cycles	8,PCI,PCC

The contents of the accumulator are placed on the peripheral device input bus during time state T1 of the PCC cycle. The device address field, represented by MMM, is placed on the device address bus to select the appropriate device during time state T2 of that cycle. During time state T3 of the PCC cycle, data contents from the selected device are removed from the input bus and loaded into the accumulator.

*Where 10_8 represents input device 4 multiplied by 2.
**Where 60_8 represents output device 30 multiplied by 2.

Eight input devices may be referenced by the INP instruction. The contents of the accumulator is latched during time state T1 of the PCC cycle to facilitate expansion of the number of input-only devices that can be connected to an MPS system.

3.6.2 Output Instruction

To write data to an output device from the accumulator, use the following:

Form	OUT
	01 RRM MM1 (RR=0)
	(output data line)←(A)
Example	OUT + 60**
Time States/ Machine Cycles	6,PCI,PCR

The contents of the accumulator are placed on the peripheral device output bus at time state T1 of the PCC cycle. The device address field, represented by RRMMM, is placed on the device bus at time state T2 of the same cycle. RRMMM must be a nonzero number in the range 10 (octal) through 37 (octal).

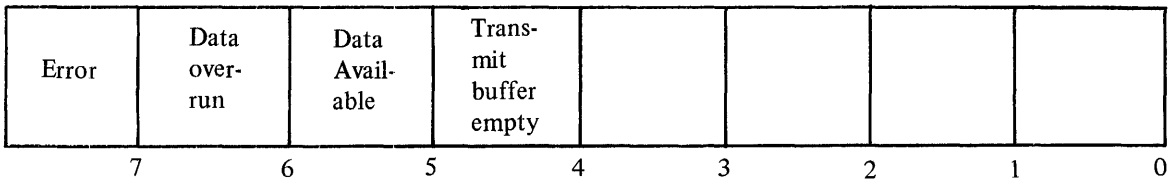
OUT can reference 24 output devices.

3.6.3 Reserved INP and OUT Instructions

Three input/output instructions are reserved for UART (Universal Asynchronous Receiver/Transmitter) control on the Processor Module. The UART is an I/O interface on the Processor Module which handles data from a serial port. Control of the UART is accomplished as follows:

Instruction	Function
INP0 01 000 001	Read data from UART
OUT0 01 010 001	Output data to UART
INP1 01 000 011	Read status from UART

The status register looks like the following:



3.7 MACHINE INSTRUCTIONS

The four instructions in this category perform the basic machine control functions halt, restart, interrupt enable, and interrupt disable. All four of these instructions have a one-byte format.

3.7.1 Halt Instruction

The halt instruction is issued by the following:

Form	HLT
	00 000 00X or 11 111 111
Example	HLT
Time States/ Machine Cycles	4,PCI

When the HLT is executed, the processor enters the stopped state after completion of time state T3. The program counter is incremented by one, and the contents of all condition flip-flops, registers, and memory are unchanged.

3.7.2 Restart Instruction

The restart instruction can be used as a one-byte unconditional call to any of eight specified locations in the first 64 words of memory. The called addresses are the following octal locations:

0
10
20
30
40
50
60
70

Each of these eight locations can be used as the starting address of an eight-word subroutine.

The restart is a one-byte instruction which is issued by the following:

Form	RST
	00 AAA 101
	(stack)←(P) (P)←(000000 00AAA000)
Example	RST + a0*
Time States/ Machine Cycles	5,PCI

The contents of the program counter, P, are shifted into the top entry of the pushdown stack. Bits 3–5 of the instruction, AAA in the model, are moved into positions 3–5 of P. All other bit positions of the program counter are zeroed.

The restart is sometimes used in place of a call instruction because it is at least twice as fast and uses only one-third as much memory. For example, instead of issuing a CAL INCHL directly in the assembly language code, the programmer might place the INCHL subroutine in low memory to be called with an RST.

NOTE

Users may employ OPDEF (Paragraph 6.10.6) to define their specific instructions involving RST, INP, and OUT.

3.7.3 Interrupt Enable and Disable Instructions

The interrupt enable (ION) and interrupt disable (IOF) instructions are specialized input/output instructions which are defined as machine instructions because of the interrupt control functions performed.

*Where a represents an octal digit from 0 to 7

To enable external events to interrupt normal program sequence, issue the following:

Form	ION
	01 010 011
	Enable external event interrupt
Example	ION
Time States/ Machine Cycles	8,PCI,PCC

To disable external events from interrupting, issue the following:

Form	IOF
	01 010 101
	Disable external event interrupt
Example	IOF
Time States/ Machine Cycles	6,PCI,PCC

External event control is similar to interrupt control, but the condition flip-flops cannot be saved and restored, and the registers cannot be saved and restored with sufficient generality. The STRT switch on the programmable module performs a simulated interrupt and jumps to the address set in the Monitor/Control Panel (MCP) Switch Register. As in an actual interrupt, the interrupt recognition logic is automatically disabled after every interrupt. Therefore, if a program is to recognize interrupts, an ION instruction must be issued in the initialization routine. After the ION is executed, one more instruction can be executed before interrupts are enabled. The ION instruction should be the last executable instruction before the RET command in an interrupt service routine.

The IOF instruction is used to disable interrupts from external events excluding power fail. This instruction should therefore be used with extreme caution. Typically it is used when a particular operation is being performed and it is not appropriate to allow interrupts for the duration of the operation.

A typical program sequence is shown on the opposite page.

```
        *50
        JMP DOFF        /ASSUME DOFF IS CONNECTED TO PIN J1

        *70
        JMP PFR         /ASSUME POWER FAIL CONNECTED TO U1

        *100
        XRA             /CLEAR STATUS FLIP-FLOPS AND AC
            .           /OTHER INITIALIZATION CODE
            .
            .

        ION             /ENABLE INTERRUPTS
        JMP .           /LOOP HERE UNTIL INTERRUPT OCCURS
            .           /OR COULD BE VERY LOW PRIORITY
            .           /ROUTINES
            .

DOFF,       .           /DOFF INTERRUPT SERVICE ROUTINE
            .           /CLEAR DOFF INTERRUPT FLAG
            .

        ION             /ENABLE INTERRUPT
        RET             /RETURN TO PROGRAM SEQUENCE BEFORE
            .           /INTERRUPT OCCURRED
            .
            .

PFR,        .           /ROUTINE TO SHUT DOWN OR STABILIZE
            .           /SYSTEM BEFORE ALL POWER IS LOST.
            .           /AFTER POWER IS RESTORED, SYSTEM
                        /SHOULD BE RESTARTED AT THE BEGINNING

        HLT
```

CHAPTER 4
THE PDP-8 HOST ENVIRONMENT

The Digital Equipment Corporation PDP-8 computer system has been selected as the host machine for preparing and processing programs designed for Microprocessor Series (MPS) use. This chapter describes the hardware and software environment in which programs will be developed and assembled, defines both minimum operational requirements and expanded capabilities and options, and outlines the characteristics and use of major PDP-8 hardware and software modules, some of which are described in far greater functional detail in subsequent chapters of this handbook.

4.1 INTRODUCTION TO THE PDP-8

The PDP-8 is a small, economical, and efficient computer designed for effective program development, assembly, and execution. It is a single-address parallel machine which operates on 12-bit binary numbers using two's complement arithmetic. An effective PDP-8 can function with only 4K of core memory and no peripheral devices whatsoever. Yet this minimum configuration can expand to support as much as 32K of core and a variety of devices. PDP-8 users of Microprocessor Series Modules are assumed to possess only the minimum hardware configuration described in the next paragraph. They will receive a set of system programs in paper tape form, sufficient to load and copy paper tapes and to edit and assemble programs developed for the Processor Module. This software system will be described in detail in subsequent paragraphs of this chapter.

4.2 PDP-8 HARDWARE ENVIRONMENT

The minimum configuration of PDP-8 equipment for support of the software defined for this product consists of the following:

- PDP-8 central processing unit, programmer's console, and 4K (4096 decimal words) of core memory
- Keyboard/printer terminal, often a Teletype®
- Low-speed (associated with the Teletype) paper-tape reader/punch
- If Teletype is not the terminal device, high-speed paper-tape reader/punch

Other peripheral devices may be supported by an installation's PDP-8 configuration; these will normally be ignored while processing MPS programs. The only exception occurs when both high-speed and low-speed paper-tape reader/punches are availabe. With most programs, the user must specify which paper tape unit is available by setting a switch on the PDP-8 console. One variety of paper tape unit is a necessary component of the PDP-8 minimum configuration, but both are supported by the system and described in this chapter. The following paragraphs describe the use of each hardware module just defined.

4.2.1 Central Processing Unit (CPU)

The PDP-8 computer system is available in many models and configurations. The CPU most frequently utilized by MPS users is the PDP-8/E. In many of the paragraphs which follow, specific references to switches or keys may be most relevant to the PDP-8/E. However, users of other PDP-8 computer models (e.g., PDP-8/F, PDP-8/I, PDP-8/L, PDP-8/M) normally can use the software supplied with this system with equal facility.

4.2.2 Programmer's Console

The PDP-8 programmer's console provides switches and indicator lamps, faciliating manual control of the computer by allowing a programmer to examine or alter the contents of memory locations and to determine the status of a program in execution. In Figure 4-1 you can see a photograph of a PDP-8/E console. Table 4-1 summarizes the functions of all switches and indicators on this console. Note that some of these functions are specific to the PDP-8/E and may not be relevant to all user machines.

®Teletype is a registered trademark of the Teletype Corporation.

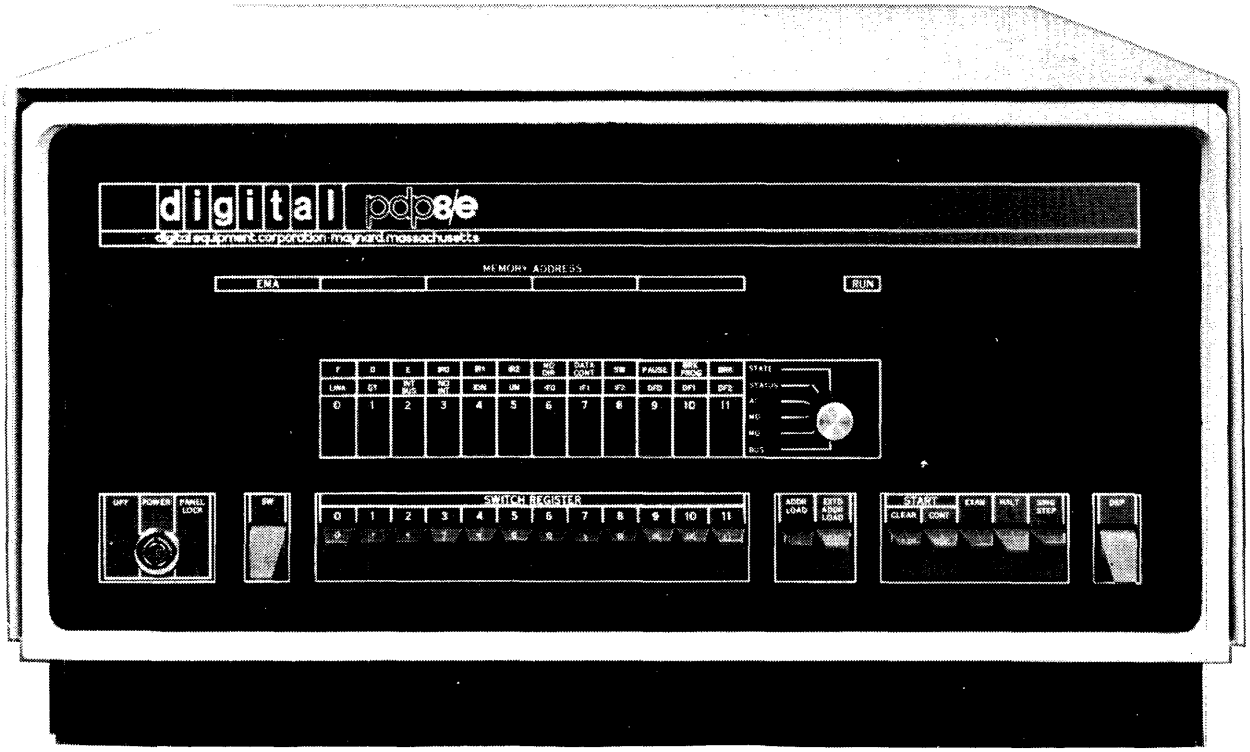


Figure 4-1 PDP-8E Programmer's Console

Table 4-1
Programmer's Console Control and Indicator Functions

Control or Indicator	Function
OFF/POWER/PANEL LOCK	In the counter-clockwise, or OFF position, this key-operated switch disconnects all primary power to the computer. In the POWER, or vertical position, it applies power to the computer and all manual controls. In the PANEL LOCK, or clockwise position, it applies power to the computer, the Switch Register and the RUN light only. In this position, a running program is protected from inadvertent switch operation.
SW	When this switch is up, the OMNIBUS SW line is high (logical 1). When it is down, the SW line is low. This switch is used by special peripheral routines.
SWITCH REGISTER	The Switch Register (SR) may be loaded with a 12-bit binary number by setting each of the twelve switches either up for a 1, or down for a 0.
ADDR LOAD	Pressing the ADDRess LOAD switch loads the contents of the SR into the central processor MA register and forces the processor to enter a fetch state. This causes the contents of the core memory location designated by the SR to be loaded into the MB register.
EXTD ADDR LOAD	Pressing the EXTendeD ADDRess LOAD switch loads the contents of SR bits 6–8 into the instruction field register and the contents of SR bits 9–11 into the data field register.

Table 4-1 (Cont)
Programmer's Console Control and Indicator Functions

Control or Indicator	Function
CLEAR	Pressing the CLEAR switch loads a binary 0 into bits 0–11 of the accumulator, the link, all I/O device flag registers, and the interrupt request flag register. This is equivalent to executing a CAF (Clear All Flags) instruction.
CONT	Pressing the CONTinue switch sets the run flip-flop and issues a memory start to begin program execution at the address specified by the current contents of the central processor MA register.
EXAM	Pressing the EXAMine switch loads the contents of core memory at the address specified by the MA register into the MB register and then increments the MA register and the PC. Repeated operation of this switch permits the contents of sequential core memory locations to be examined.
HALT	Pressing HALT clears the run flip-flop and causes the computer to stop at the beginning of the next fetch state. Operating the computer with HALT depressed causes one complete instruction to be executed whenever the CONTinue switch is pressed.
SING STEP	Pressing SINGle STEP clears the run flip-flop and causes the computer to halt at the next machine cycle. Operating the computer with the SINGle STEP switch depressed causes only one machine cycle to be executed whenever the CONTinue switch is pressed.
DEP	Lifting the DEPosit switch loads the contents of the SR into the MB register and into core memory at the address specified by the current contents of the central processor MA register, then increments the PC and the MA registers. This facilitates manual storage of information in sequential core memory locations.
EMA	The 3-bit Extended Memory Address register displays the memory field designation of the memory field currently being accessed.
MEMORY ADDRESS	The MEMORY ADDRESS register displays the contents of the central processor MA register. It combines with the EMA register to provide the 15-bit address of the next core location to be accessed.
RUN	The RUN indicator is lit whenever all machine timing circuits are activated and capable of executing instructions.
Indicator Selector Switch	This six-position rotary knob designates which of six possible registers (or combinations of registers) is to be loaded into the adjacent 12-bit display.
Setting this knob to:	
BUS	Displays the logical state of the data gating lines which connect the major registers.

Table 4-1 (Cont)
Programmer's Console Control and Indicator Functions

Control or Indicator	Function
MQ	Displays the contents of the multiplier quotient register.
MD	Displays the contents of the MB register. This indicates the last information read from or written into core memory.
AC	Displays the contents of the accumulator.
STATUS	Each display light is turned on to indicate the designated condition:
Indicator Light/Bit Position	**Turned On to Indicate**
0	The link contains a binary 1.
1	The Greater Than Flag (GTF) is raised.
2	The interrupt request line is asserted.
3	A processor condition, which prevents program interrupts, has been initiated by software.
4	The interrupt enable flip-flop is on.
5	The user mode line is asserted.
6–8	Displays the contents of the instruction field register.
9–11	Displays the contents of the data field register.
STATE	With the Indicator Selector knob in the STATE position, each display light is turned on to indicate the following condition:
Indicator Light/Bit Position	**Turned On to Indicate**
0	Currently in fetch state.
1	Currently in defer state.
2	Currently in execute state.
3–5	Displays the contents of the instruction register.
6	The MD DIR line is asserted.
7	The BREAK DATA CONT line is asserted.
8	The SW line is asserted.
9	The PAUSE I/O line is asserted.
10	The BREAK IN PROG line is asserted.
11	The BREAK CYCLE line is asserted.

NOTE

The function of the various transmission lines cited and their associated control logic is documented in Digital's *Small Computer Handbook.*

4.2.3 Keyboard/Printer Terminal

The PDP-8 user interacts with many of the system programs described in this chapter in a command-oriented way using a terminal as the input/output device. The following terminals may be used with the PDP-8 system described.

- LT33 Teletype
- VT05 Display Terminal
- LA30 DECwriter Data Terminal

The VT05 and LA30 are much faster than the Teletype, which prints at a maximum rate of ten characters per second; but the Teletype offers the advantage of a built-in low-speed paper-tape reader/punch unit. The VT05 has a video display screen; the other two terminals supply hard copy. Figure 4-2 is an illustration of the LT33 Teletype, the basic I/O device assumed for users of the system.

Certain specialized Teletype knobs and keys require some clarification. The control knob of the LT33 Teletype has the following three positions:

Position	Meaning
LINE	The Teletype console is energized and connected to the computer as an input/output device under computer control.
OFF	The Teletype console is de-energized.
LOCAL	The Teletype console is energized for off-line operation under control of the Teletype keyboard and switches exclusively.

These three positions will be referenced in this and subsequent chapters when discussing the handling of paper tape functions.

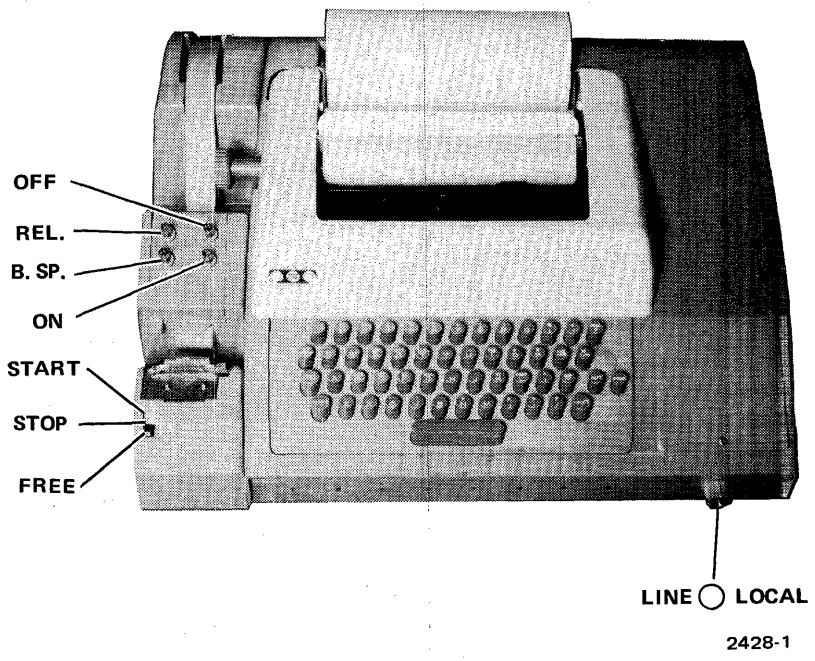


Figure 4-2 LT33 Teletype Console

The Teletype keyboard shown in Figure 4-3 combines standard typewriter characters with special functions which are summarized in Table 4-2.

4.2.4 Low-Speed Paper-Tape Reader/Punch

The Teletype paper-tape reader (also called the low-speed reader) is used to read data punched on paper tape into core memory. The data is read from an eight-channel, perforated paper tape at a maximum rate of ten characters per second. Operation is controlled by a three-position switch, shown in Figure 4-2.

Setting	Meaning
START	Activates the reader; reader sprocket wheel is engaged and operative.
STOP	De-activates the reader; reader sprocket wheel is engaged but not operative.
FREE	De-activates the reader; reader sprocket wheel is disengaged.

The paper-tape punch is used to perforate eight-channel, rolled, oiled paper tape at a maximum rate of ten characters per second. The punch controls are shown in Figure 4-2 and described below:

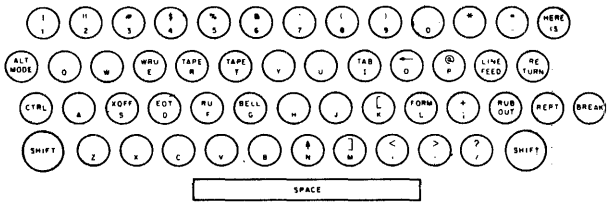


Figure 4-3 Teletype Keyboard

Setting	Meaning
REL.	Disengages the tape to allow tape removal or loading.
B.SP.	Backspaces the tape one space for each firm depression of the B.SP. button.
ON	Activates the paper-tape punch.
OFF	De-activates the paper-tape punch.

Table 4-2
Special Keyboard Functions

Key	Function	Use
SPACE	Space	Used to combine and delimit symbols or numbers in a symbolic program.
RETURN	Carriage Return	Used to terminate a line of input.
HERE IS	Blank Tape	Used to generate leader/trailer tape. Effective in LOCAL control mode only.
RUBOUT	Rubout	Used for deleting erroneous characters. Punches all eight channels.
CTRL/SHIFT/ REPT/P	Code 200	Used for leader/trailer on BIN format tapes. Keys must be released in reverse order: P, REPT, SHIFT, CTRL.
LINE FEED	Line Feed	Follows carriage return to advance terminal printer one line.
SHIFT		Used to type the characters and symbols which appear on the upper portion of certain keys.

The use of the low-speed paper-tape reader/punch for performing specific editing and assembling functions is described in chapters on those modules. The following list of instructions is supplied only as an example of off-line, low-speed, paper tape usage. It allows a PDP-8 user to generate off-line a symbolic tape to be used as input to the Assembler when, for some reason, tapes cannot be created by the Editor.

1. Set the Teletype control knob to LOCAL and turn the paper-tape punch ON.

2. Press the HERE IS key on the Teletype keyboard to produce several inches of leader tape.

3. Type the program on the Teletype keyboard. To correct an error, press B.SP. until the error is under the print/punch station; then press RUBOUT until the error and all subsequent characters have been deleted. The erroneous character and all subsequent characters may now be retyped.

4. Press the HERE IS key to produce several inches of trailer following the symbolic program; remove the tape by tearing it against the plastic cover of the punch.

The following procedure is employed to obtain an off-line listing of an ASCII-coded (USA Standard Code for Information Interchange) symbolic tape:

1. Set the paper tape reader switch to STOP or FREE.

2. Release the plastic cover of the reader unit and place the tape over the read station with the small sprocket holes over the sprocket wheel. Close the cover.

3. Set the Teletype control knob to LOCAL.

4. Push the paper-tape reader switch to START and release. A printed copy of the tape will be produced on the Teletype. If the paper-tape punch is ON, a duplicate of the tape will also be generated.

There are three basic paper tape formats used by PDP-8 system programs provided with this system:

1. ASCII format, used for source text output from the Editor or input to the Assembler.

2. BIN (binary) format, used for almost all of the system programs which execute on the PDP-8.

3. RIM (read-in mode) format, used for the Microprocessor Host Loader.

In addition to these basic formats, the MLA Assembler (Chapter 6) punches paper tape in a binary format suitable for execution on the Processor Module. While the BIN tape just mentioned has a PDP-8 format, the binary tape produced by the Assembler has a format which is used by MPS. This format is described in detail in Chapter 6.

Paper tapes punched in ASCII format use all eight channels of the tape to represent a single character (letter, number, symbol). An example of source text output by the Editor is shown in Figure 4-4.

Paper tapes input to the MLA contain mnemonic instructions and symbolic addresses punched in ASCII format. These are translated into binary instructions and absolute addresses during assembly and are punched out into binary format for execution on the Processor Module.

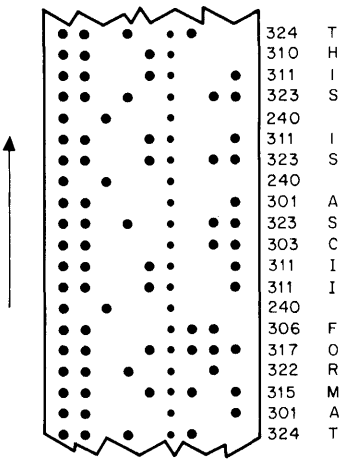


Figure 4-4 ASCII Format

System tapes containing assembled programs in binary format are usually loaded into core under program control, using the Microprocessor Host Loader (MHL), a system program provided to users of this system. This binary loader must be placed in core before any binary format tape is loaded. The Microprocessor Host Loader itself is the only system tape punched in RIM (read-in mode) format. To read a tape in RIM format, the RIM Loader, a 17-instruction program which must be keyed-in manually, must be in core (see the paragraph on the RIM Loader). RIM format uses pairs of adjacent columns to represent 12-bit binary words directly.

Channels 1 through 6 are used to represent either addresses or information to be stored. A channel-7 punch indicates that the current column and the following column are to be interpreted as an address specifying the location at which the information contained in the following two columns is to be stored. The tape leader and trailer for RIM format tape must be punched in channel-8 only (octal 200). Figure 4-5 is an example of tape punched in this way.

BIN (binary) format tape is similar to RIM format tape, except only the first address in a series of consecutive addresses is specified. A channel-7 punch indicates that the current column and the following column are to be interpreted as an address. Successive pairs of columns are stored in sequential locations following this address until another channel-7 punch is encountered. A channel-7 and a channel-8 punch designate the current column as a memory field specification. Leader/trailer tape must be punched in channel-8 only. Figure 4-6 is an example of binary format.

4.2.5 High-Speed Paper-Tape Reader/Punch

Loading long series of paper tape programs into core memory with the low-speed reader of the LT33 Teletype unit can be time-consuming. Punching a long assembled program on paper tape can also be slow. If handling lengthy paper tapes is required frequently, much computer time is wasted while low-speed input/output devices read or punch data. The high-speed paper-tape reader/punch unit performs paper tape input and output at a considerably faster rate than the low-speed reader and punch. It is of great value in any system that requires a great deal of tape handling.

The high-speed paper-tape reader/punch unit is available in two versions: The rack-mounted PC8-EA illustrated in Figure 4-7 and the table-top PC8-EB. Both units consist of a PR8-E high-speed paper-tape reader and a PC8-E high-speed paper-tape punch mounted on a single chassis. The reader and punch are also available separately. Figure 4-7 illustrates the reader/punch unit.

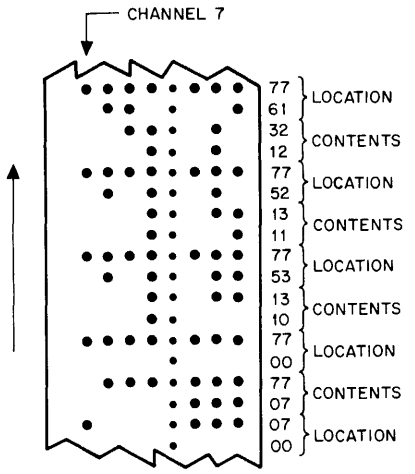


Figure 4-5 RIM Format

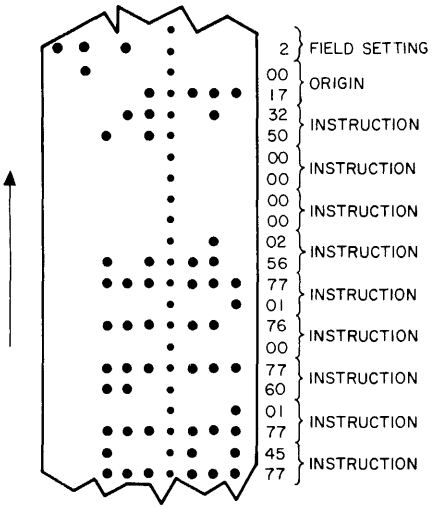


Figure 4-6 BIN Format

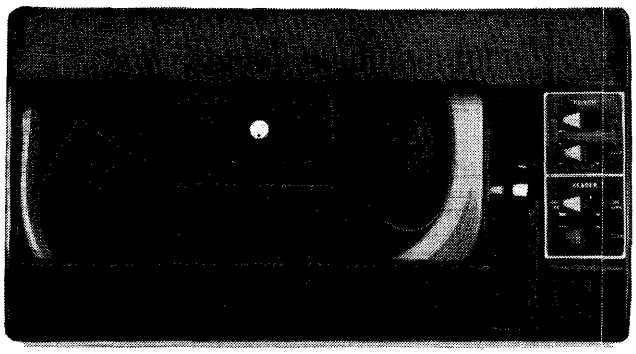


Figure 4-7 High-Speed Paper-Tape Reader/Punch

The high-speed reader accepts input data from eight-channel, fan-folded, non-oiled paper tape at a maximum rate of 300 characters per second, or thirty times the LT33 maximum input rate. The high-speed punch records output data at a maximum rate of 50 characters per second.

The reader and punch are each supplied with an ON/OFF rocker switch which applies power to the respective units in the ON position and disconnects power in the OFF position. Each device is also provided with a FEED switch which advances the tape without reading, in the case of the reader, or advances tape with only the feed holes punched, in the case of the punch unit. The reader is supplied with a control knob which may be turned clockwise to raise the tape retaining lever and free the tape, or counter-clockwise to lower this lever and engage the sprocket wheel.

The following procedure is employed to position tapes in the high-speed reader:

1. Turn the control knob to raise the tape retaining lever.
2. Place a fan-folded tape in the right-hand bin.
3. Place several folds of leader in the left-hand bin and position the tape so that the sprocket wheel engages the feed holes.
4. Turn the control knob to lower the tape retaining lever.
5. Press the FEED switch briefly to ensure that the tape is properly positioned.
6. Tape is advanced and read during program execution.

4.3 PDP-8 SOFTWARE ENVIRONMENT

The minimum software environment required for convenient and efficient program development, assembly, and duplication is quite limited. The few necessary system programs are designed to be used with ease, and are documented in full detail in this and subsequent chapters. These programs are grouped together in kit form, designated Microprocessor Series Software Tools (MPSST), DEC No. QF500-AB. All software components of MPSST will be provided in the form of binary paper tapes ready to be loaded on the user's PDP-8. Table 4-3 summarizes the binary tapes which compose the minimum necessary PDP-8 software system.

Table 4-3
PDP-8 System Programs

Program	Function	DEC Number
Microprocessor Host Loader (MHL)	Loads binary-coded tapes	DEC-08-UMPLA-A-PM
Microprocessor Language Editor (MLE)	Modifies or generates source text from Teletype commands by reading and writing paper tapes	DEC-08-UMPEA-A-PB
Microprocessor Language Assembler (MLA)	Assembles source text into binary format by reading and writing paper tapes and listing at user's option	DEC-08-UMPAA-A-PB
Master Tape Duplicator/ Verifier (MTD)	Copies paper tapes and verifies their contents	DEC-08-UMPDA-A-PB
Microprocessor ROM Programmer (MRP)	Copies and modifies paper tapes and PROMs	DEC-08-UMPPA-A-PB

These system programs run on the PDP-8 and are described in greater detail below. Two other programs, the Microprocessor Debugging Program (MDP) (DEC-08-UMPMA-A-PB), and the Microprocessor Program Loader (MPL) are supplied for use on the Processor Module itself and will be described here. MDP is provided in paper tape form; MPL is supplied as part of the MPS hardware. An additional component, the RIM Loader, is also described; this Loader is not a binary tape but a series of instructions to be entered manually into the PDP-8 before any of the system paper tapes can be used.

4.3.1 The RIM Loader

The RIM (Read-In-Mode) Loader is used to load into core programs punched in RIM format. For purposes of this system, the only program entered in this format is the Microprocessor Host Loader, described here. Unless the PDP-8 being used has a PDP-8/E Hardware Bootstrap Option, the RIM loader must be loaded manually ("toggled"), using the switches located on the programmer's console.

There are two versions of the RIM Loader: one program is designed to be used when tapes are to be loaded from the low-speed (Teletype) paper-tape reader, and the other is intended for input from the high-speed reader. Table 4-4 lists the octal instructions for these programs. The loading and verifying procedures are detailed in the flowcharts in the two figures which follow. After loading RIM, it is good programming practice to verify that all instructions have been entered properly.

Table 4-4
RIM Loader Programs

Location	Instruction	
	Low-Speed Reader	High-Speed Reader
7756	6032	6014
7757	6031	6011
7760	5357	5357
7761	6036	6016
7762	7106	7106
7763	7006	7006
7764	7510	7510
7765	5357	5374
7766	7006	7006
7767	6031	6011
7770	5367	5367
7771	6034	6016
7772	7420	7420
7773	3776	3776
7774	3376	3376
7775	5356	5357
7776	0000	0000

When loaded, the RIM loader occupies absolute locations 7756 through 7776. The following procedures are used to load. (Figure 4-8)

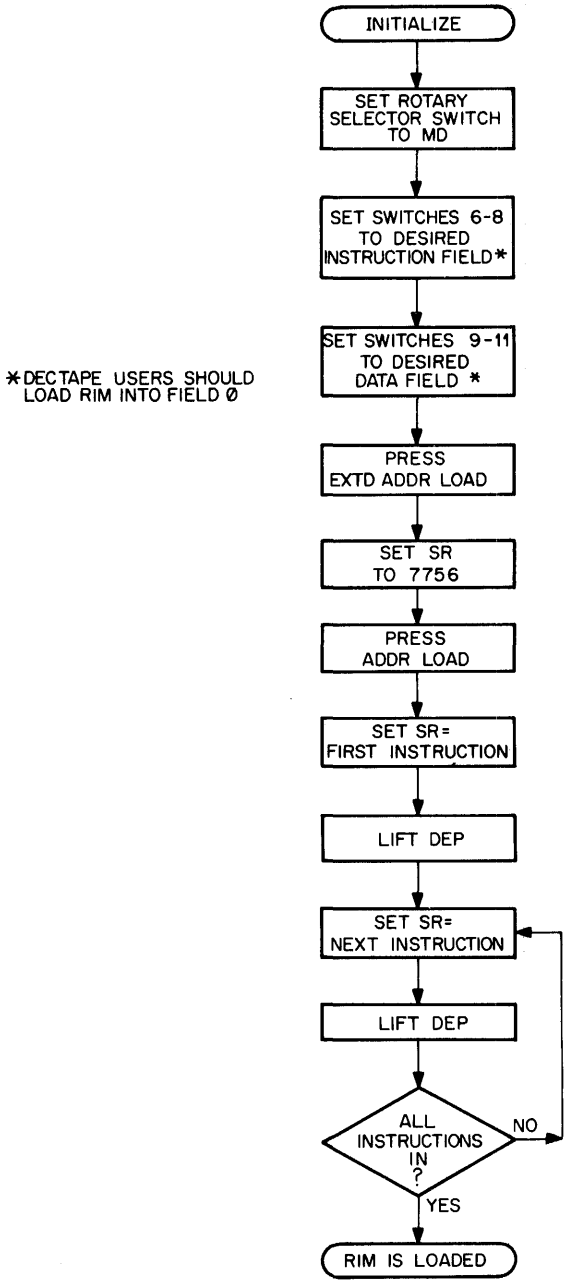


Figure 4-8 Loading the RIM Loader

To ensure that the load has been successful, follow the steps shown in Figure 4-9.

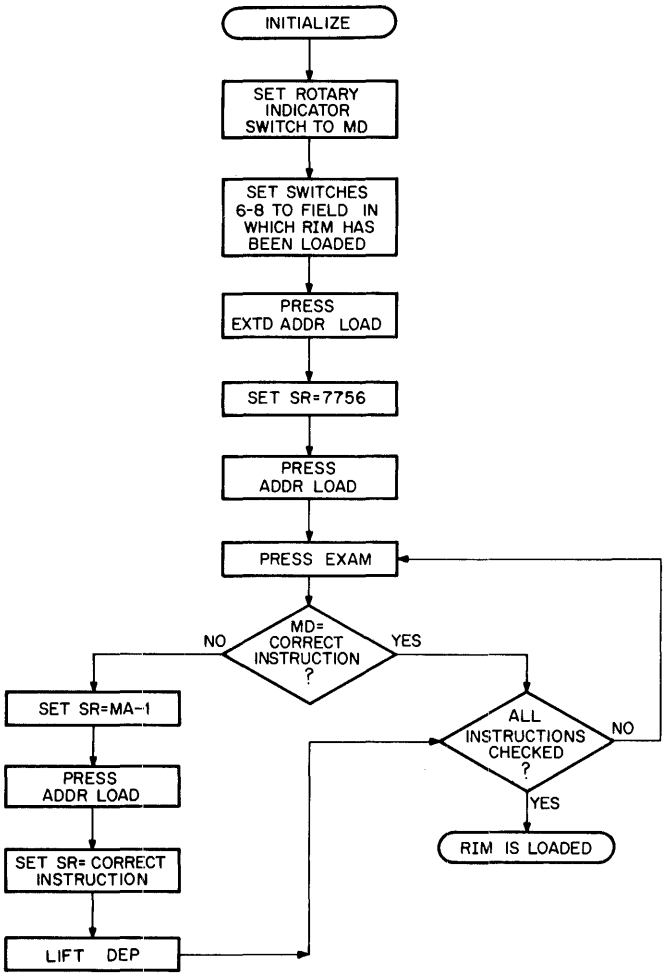


Figure 4-9 Checking the RIM Loader

4.3.2 The Microprocessor Host Loader

The Microprocessor Host Loader (MHL) is a utility program which is loaded into core to read binary-coded data on paper tape and store it in core memory. MHL is used primarily to load system binary programs.

MHL is stored on punched paper tape in RIM-coded format; therefore, RIM must be in core before MHL can be loaded. When loading MHL, the input device (low-speed or high-speed reader) must be the same as that selected when loading RIM, and RIM and MHL must be loaded into the same field.

Once stored in core, MHL resides on the last page of core, occupying absolute locations 7625 through 7752 and 7777 of the field in which it was loaded. The programmer must be aware that if he writes a program that uses the last page of core, MHL will be destroyed when the program is run, and both RIM and MHL will require reloading before another program can be loaded. Figure 4-10 details the method of loading MHL.

The programmer is now able to load binary tapes using the method described in Figure 4-11.

4.3.3 The Microprocessor Language Editor

The Microprocessor Language Editor (MLE) provided with this system is a paper tape-oriented source text editor with which the user modifies source program tapes by submitting commands from the Teletype keyboard. Using MLE alleviates the tedious task of preparing source program tapes off-line. This program is described in detail in Chapter 5.

The Microprocessor Language Editor is provided to users of this system in the form of a binary tape which is loaded into core by means of the Microprocessor Host Loader, using either the low-speed or high-speed paper-tape reader. The unit is selected at the time MLE is loaded, as shown in Figure 4-11. MLE itself uses either the low-speed or high-speed paper tape reader/punch for I/O. Text to be modified may be entered into core using either the Teletype or the paper-tape reader. The modified source text may be punched-out using the paper-tape punch. Switch Register bits are set to indicate high-speed input and output (low-speed tape is the default).

4.3.4 The Microprocessor Language Assembler

The Microprocessor Language Assembler (MLA) offers a complete instruction set and group of pseudo-instructions for straightforward development and processing of assembly language programs for the Processor Module. For consistency of use and ease of training, the Assembler's character set, available operators, and construction of statements, symbols, and expressions conform in many ways to other standard PDP-8 Assemblers. This program is described in detail in Chapter 6.

The Assembler is provided to users of this system in the form of binary tape which is loaded into core by means of the Microprocessor Host Loader, using either the low-speed or high-speed paper-tape reader. The unit is selected at the time MLA is loaded, as shown in Figure 4-11.

The Assembler itself is oriented to the use of paper tape, and uses either the low-speed or high-speed paper-tape reader/punch for I/O. The source program to be assembled is usually prepared using the Microprocessor Language Editor (but can be generated off-line) and is read by the available paper-tape reader. The assembly itself is performed in three passes, each of which produces certain

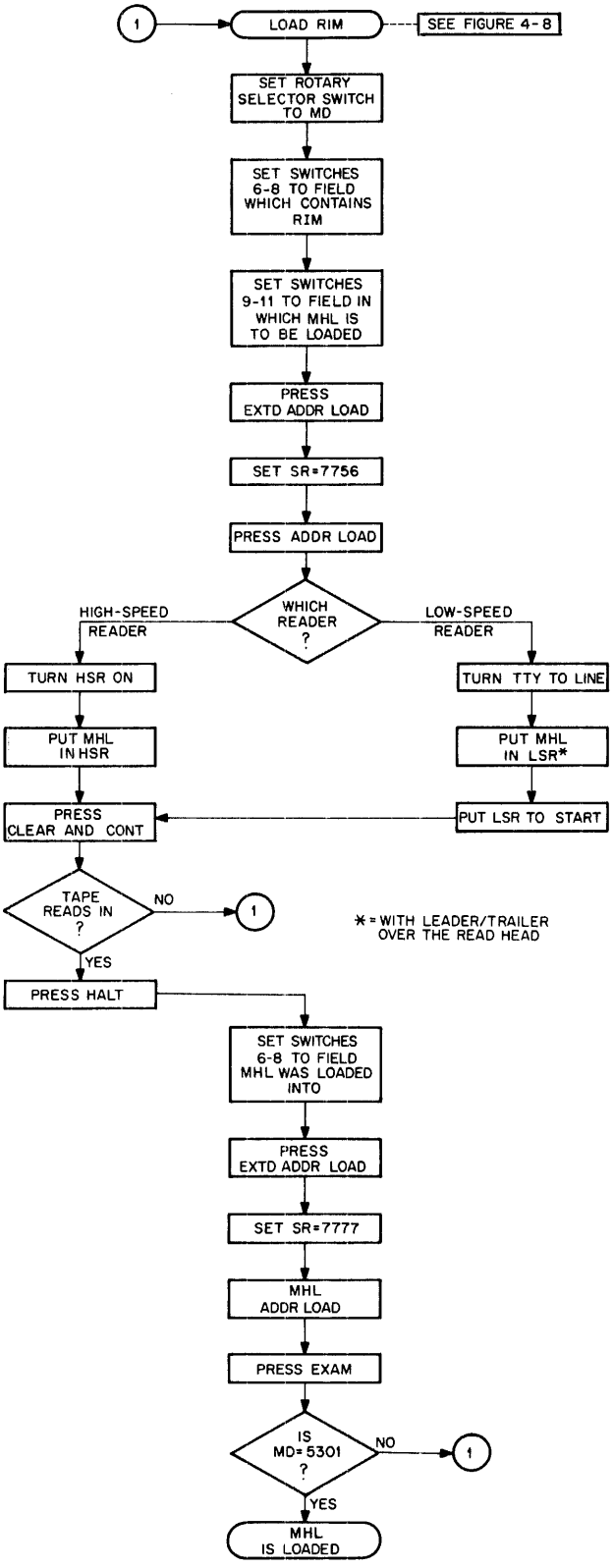


Figure 4-10 Loading the Microprocessor Host Loader

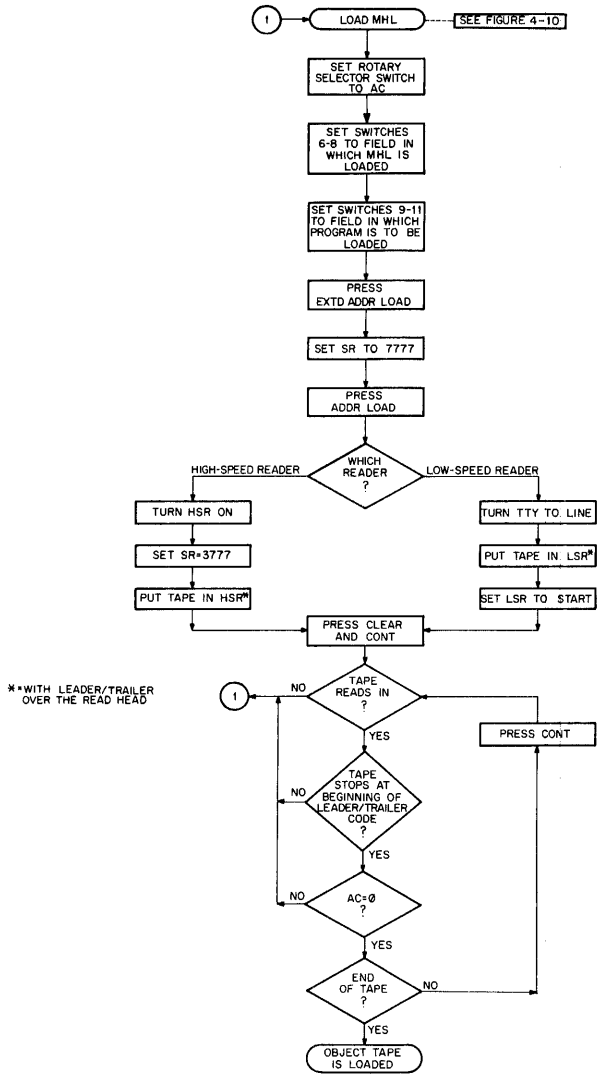


Figure 4-11 Loading a Binary Tape Using MHL

listings of errors and program text. Listings can be produced on either the Teletype, the line printer, or the paper-tape punch; selection of the appropriate unit is made at the start of pass 3 by setting the appropriate Switch Register bits. A symbol table created in pass 1 is used to punch a binary-coded output tape during pass 2. This binary tape can subsequently be loaded into the Processor Module for testing and execution.

Either the low-speed or high-speed paper-tape reader can be used for Assembler input. If both units are available, the high-speed reader is selected for use in reading the source tape and producing Assembler output. If only one unit is

available, the Assembler will dynamically determine which type of paper-tape reader is to be used. Switch Register bits are set to determine which unit is to be used for punching purposes.

4.3.5 Master Tape Duplicator/Verifier

The Master Tape Duplicator/Verifier (MTD) is a system program used to copy and check eight-channel paper tapes using the high-speed paper-tape reader and punch. Installations with a low-speed (Teletype) punch can perform the same function by simply copying and listing off-line, as outlined in the paragraph on the low-speed paper-tape reader/punch.

MTD is provided to users of this system in the form of a binary tape which is loaded into core by means of the Microprocessor Host Loader. Since only installations with a high-speed paper-tape reader punch will be using MTD, the high-speed device will normally be selected at the time the system program is loaded. Once in core, MTD uses all but the last page of memory as a buffer.

Paper tapes are duplicated and verified in three passes as described below:

Pass	Function
1	Production of the master tape
2	Duplication of the master tape
3	Verification of the duplicated tape

After MTD has been loaded into core, the user performs the following initial procedures:

1. Set the Switch Register to 0200.
2. Press the ADDR LOAD and START keys; the computer will halt.
3. Set the Switch Register to 4000.
4. Place the paper tape to be duplicated in the high-speed paper-tape reader.
5. Turn the paper-tape punch on and feed paper through the punch unit for approximately one length of a fold.
6. Press the CONTinue key.

After these procedures have been completed, pass 1 begins. The input tape is read and an output master tape is punched. Because the paper-tape reader is faster than the punch, the reader stops occasionally to allow the punch to catch up. After punching is complete, the following message will be displayed on the terminal:

MASTER CREATED

During pass 1, two types of checksums are accumulated:

1. The number of nonzero characters on the tape.
2. The sum of characters on the tape.

Both sums are computed module 4096 and are punched-out at the end of the master tape.

To begin pass 2, remove the punched master tape from the punch unit and place it in the reader. Prepare the punch unit for another copy by feeding blank tape. Set the Switch Register to 2000 and press the CONTinue key. During this pass, the master tape will be duplicated and a new set of checksums will be accumulated. After the master tape has been duplicated, the following message will be displayed on the terminal:

SET SWITCHES TO NUMBER OF COPIES TO BE MADE – PRESS CONTINUE

If the user wants to generate one master and four copies of his input tape, now he will set the Switch Register to 0003, turn the punch on, ready tape, and press the CONTinue key. After the next copy has been duplicated, MTD will feed several folds of blank tape and then output the following message on the terminal:

PRESS CONTINUE

The user presses CONTinue. MTD produces another copy, feeds several folds of blank tape, and displays the following message:

PRESS CONTINUE

The user presses the CONTinue key. MTD produces another copy and displays the following message:

DUPLICATION OK

After all copies have been produced, pass 3 can begin. During this pass, the tape copies are verified by comparing checksums. The user places the "master" tape in the reader, sets the Switch Register to 1000, and presses CONTinue. The master tape will be read and the following message displayed:

0001 VERIFY OK

Next, the user loads the first duplicate tape in the reader and presses CONTinue. After reading the tape, MTD will display the following:

0002 VERIFY OK

The next three duplicate tapes are loaded and verified in turn. The following messages are displayed – one for each tape:

0003 VERIFY OK
0004 VERIFY OK
0005 VERIFY OK

MTD uses the PDP-8 program interrupt facility to keep both the paper-tape reader and punch operating at maximum speed. A buffer is filled by the reader and emptied by the punch. *Never* remove a paper tape from the reader until all punching has stopped or valuable data may be lost from the tape.

The Duplicator does not currently check for extra blank frames in the duplicate tape. If this presents a problem for users of the program, another program can be used for secondary verification. User tapes that are too long for the space left in core should be broken into two or more shorter tapes for this operation. The alternate binary tape and documentation may be ordered from the DECUS Program Library, numbered DIGITAL-5-10-S-BIN.

4.3.6 Microprocessor ROM Programmer

The Microprocessor ROM Programmer (MRP) is used to read, write, and verify programmable read-only memory (PROM) chips for use on the MPS modules. PROMs can be copied to or from paper tape, and memory locations can be examined, modified, zeroed, or listed. This program is described in detail in Chapter 9.

MRP is provided to users of this system in the form of binary tape which is loaded into core by means of the Microprocessor Host Loader, using either the low-speed or high-speed paper-tape reader. The unit is selected at the time MRP is loaded, as shown in Figure 4-11. Switch Register bits can be set to send selective output to the line printer or to choose the high-speed or low-speed paper-tape punch as the punch unit.

4.3.7 Microprocessor Debugging Program

The Microprocessor Debugging Program (MDP) is a debugging aid which runs on the Processor Module, not on the PDP-8. It enables the user to read, modify, and rewrite binary programs in paper tape form. MDP capabilities include the ability to examine memory locations, condition flip-flops, and index registers, to set a breakpoint, and to allow a program segment to execute to that breakpoint. Binary code can be examined, tested, and modified without requiring reassembly on the PDP-8. This program is described in detail in Chapter 8.

MDP is provided to users of MPS in the form of an MPS binary tape. It is loaded into module memory by means of the Microprocessor Program Loader (MPL). Input to MDP is normally an MPS binary tape produced by the Microprocessor Language Assembler. MDP produces paper tape and Teletype printer listings as output.

4.3.8 Microprocessor Program Loader

The Microprocessor Program Loader (MPL) is a loader which is supplied as part of the Microprocessor Series hardware and is available to users of the KC341 MPS Monitor/Control Panel (MCP). It allows programs to be loaded into MPS memory from paper tape for execution on the module. A paper-tape reader must be available for use with MPL, as well as the MPS Universal Asynchronous Receiver/Transmitter (UART) interface, and read-only and random-access MPS memory. Operation of the loader is described in detail in Chapter 7.

CHAPTER 5
MICROPROCESSOR LANGUAGE EDITOR

5.1 INTRODUCTION TO THE EDITOR

The Microprocessor Language Editor (MLE) provided to users of this system is a PDP-8 based Editor oriented to paper tape usage. It is interactive and offers an extensive set of commands which can be entered from the Teletype or other terminal keyboard. Primarily it is used as an on-line tool for creating and modifying source program tapes.

The Editor facilitates both program entry and program correction. Source text is either entered directly from the keyboard or read into core using the low-speed (Teletype) or high-speed paper-tape reader. Once in core, the program text can be changed freely, expanded, deleted, or reformatted. At any point, all or some of the source text can be listed on the terminal printer or punched out using one of the paper-tape punches.

MLE is supplied in the form of a paper tape which is loaded into core using the Microprocessor Host Loader (MHL). Precise instructions for loading the Editor are supplied in Paragraph 5.9 of this chapter. The Editor occupies about 1000 locations of core and reserves all but the last page of core for source program use. In a 4K machine, this provides room for approximately 4200 decimal characters; that is about 60 lines of heavily commented text or about 340 lines of text without comments. When the core area (text buffer area) used by the source program is full, the Editor causes the Teletype bell to ring or an audible signal to be produced on another terminal. The buffer may then be enlarged, as described in the Operating Procedures, or the buffer may be dumped by punching it out onto paper tape. After punching, the Editor can be restarted and can continue with a clear text buffer area. If this occurs, it is recommended that the remainder of the source program be placed in core and punched out, so that the entire source program is on a single paper tape.

5.2 OVERVIEW OF EDITOR COMMANDS

This paragraph summarizes general syntax and error-detection characteristics of the Microprocessor Language Editor.

5.2.1 General Editor Syntax

MLE commands are entered from the keyboard in the following way:

Form [[m,]n[$j]]command<cr>

The command is a one-character function that directs the Editor to perform a particular operation; commands are preceded by zero, one, two, or three arguments. These arguments, represented by m, n, and j in the syntax, are digits or expressions that specify line numbers in the source text which are affected by the particular MLE function. The , (comma) and $ (dollar sign) symbols in this form represent argument delimiters, and <cr> is the command terminator – usually a carriage return. The user typically types the carriage return, and MLE inserts an automatic line feed character. Arguments enclosed in brackets are normally optional, but specific usage depends on the syntax of particular commands.

Table 5-1 provides examples of different forms of Editor commands.

5.2.2 Errors in Specifying Commands

A question mark (?) followed by a carriage return/line feed will be displayed on the terminal printer if the user does any of the following:

1. Specifies a nonexistent command; for example, the following is an error:

   H
   ?

   because H is not an MLE command.

2. Requests nonexistent information; for example, if the user requests a listing by typing L, and the text buffer is empty, the following will occur:

   L
   ?

Similarly, if certain lines not in the buffer are requested or if negative line numbers are supplied, a question mark will be displayed.

3. Includes too few arguments; for example, if the user wants to move the first 17 lines of text in the buffer to precede line 100, and types the following:

```
17$100M
?
```

the question mark will be printed, because a MOVE command requires two arguments before $, and only one is supplied. The correct format is:

```
1,17$100M
```

4. Specifies line numbers in incorrect order; in form:

```
m,ncommand
```

m must be less than n; therefore the following specification results in an error condition:

```
7,5L
?
```

Whenever a question mark is displayed in this way, the command specification in which the error occurred is ignored, and the user is free to re-enter the command.

There is one kind of command "error" which does not cause the question mark to be displayed and the command to be ignored. This is the case in which one or more arguments may be supplied for a command which requires no arguments; the following illustrates this incorrect usage:

```
1,15A
```

Because the APPEND command takes no arguments, the 1,15 specification will simply be ignored, and the text will be appended as usual.

5.2.3 Line Numbering

All lines in the text buffer area are assigned implicit decimal line numbers starting with 1. This implicit numbering scheme causes line numbers to be continually updated by the Editor to account for line insertions, moves, and deletions. This implies that the line numbers on the following original lines of text may be changed during the editing process.

Implicit Line Number	Text
1	AAA
2	BBB
3	DDD
4	EEE

For example, if the following two commands are performed to delete line 4 and insert a new line between 2 and 3

```
4D
3I
CCC
```

Table 5-1
Editor Command Options

Type of Command	Format	Example	Meaning
No Argument	A	A	Append incoming text to buffer.
One Argument	nI	84I	Insert incoming text before line number 84.
Two Arguments	m,nL	1,100L	List text buffer lines 1 through 100.
Three Arguments	m,n$jM	12,20$96M	Move lines 12 through 20 to before line number 96.

the following will result:

Implicit Line Number	Text
1	AAA
2	BBB
3	CCC
4	DDD

If deletions are being performed, it is wise to delete from the bottom of the text to the top, since deletions cause all text after lines being deleted to be adjusted. If the command shown had been performed in a different order, that is:

```
3I
CCC
4D
```

the following would have occurred:

Implicit Line Number	Text
1	AAA
2	BBB
3	CCC
4	EEE

The line containing CCC was inserted as line 3; the previous line 3, containing DDD, was therefore renumbered line 4 and the line containing EEE was renumbered line 5. After deletion of the new line 4, line 5 was renumbered line 4.

5.3 EDITOR MODES OF OPERATION

To distinguish between editing commands and actual text to be entered into the buffer, the Editor operates in either COMMAND mode or TEXT mode. In COMMAND mode, all input typed on the keyboard is interpreted as commands to the Editor to perform some operation on one or more lines of text stored in the buffer. In TEXT mode, all typed input is interpreted as text to replace, be inserted into, or be appended to the contents of the text buffer.

Immediately after being loaded into core memory and started, the Editor is in COMMAND mode waiting for a command. The user can freely enter any of the Editor commands described in this chapter. The Editor moves automatically into TEXT mode when an APPEND (A), INSERT (I), or CHANGE (C) command is supplied. In this mode, new text lines or text corrections and insertions are typed and appended or inserted as specified. To return from TEXT to COMMAND mode, the user types either of the following characters on the terminal keyboard:

1. CTRL/L: Type L while holding down the CTRL key
2. CTRL/G: Type G while holding down the CTRL key

The Editor indicates the successful transition from TEXT back to COMMAND mode by ringing the bell on the Teletype or producing an audible signal on another terminal.

5.4 SPECIAL CHARACTERS AND FUNCTIONS

Editor commands entered from the terminal often involve using certain special keys for such purposes as error correction, mode transition, paper tape control, and text buffer analysis. This paragraph summarizes the functions of these keys with particular reference to the Teletype keyboard. Specific keys may differ slightly on the keyboards of other terminals, and differences in function will be noted where these occur.

5.4.1 RETURN: Terminating a Line

In both COMMAND and TEXT modes, typing the RETURN key signals the Editor to process the information just typed. In COMMAND mode, it allows the Editor to execute the command just entered. A command will not be executed until it is terminated by the RETURN key (with the exception of = and :, explained later). In TEXT mode, RETURN causes the line of text which it follows to be entered into the text buffer. A typed line is not actually part of the buffer until terminated by the RETURN key.

5.4.2 CTRL/U: Erasing a Line

The erase character (CTRL/U combination) is used for error recovery in both COMMAND and TEXT modes. It is generated by holding down the CTRL key while typing a U and is not echoed on the Teletype. When used in TEXT mode, CTRL/U cancels everything to the left of itself back to the beginning of the line; the Editor performs a carriage return/line feed (<cr><lf>). The user then continues typing on the next line. When used in COMMAND mode,

CTRL/U cancels the entire command; MLE prints a ? and performs a <cr><lf>. The erase character cannot cancel past a <cr><lf> in either COMMAND or TEXT mode. For example, in COMMAND mode, the CTRL/U character after the A cancels the append command.

```
A?
```

In TEXT mode, the CTRL/U is pressed after "THIS" and results in a carriage return. The line containing "THIS" will not be entered in the text buffer.

```
THIS
HERE IS A TEXT MODE EXAMPLE
```

5.4.3 RUBOUT: Erasing A Character

RUBOUT is used for error recovery in both COMMAND and TEXT modes with one exception. When executing a READ command (explained later) from the paper-tape reader, RUBOUTs are ignored completely and are not entered in the buffer. It is necessary for the READ command to disable the RUBOUT function because all tab characters on paper tape are, for timing purposes, followed by RUBOUTs; recognition of these characters would cause the tabs to be ignored. RUBOUTs are not stored in the text buffer but are inserted by the Editor following all tab characters on the output tape.

At any other time, typing the RUBOUT key in TEXT mode echoes a backslash (\) and deletes the last typed character. Repeated RUBOUTs delete from right to left up to but not including the <cr><lf>, which separates the current line from the previous one. For example:

```
THE QUUICK\\\\ICK BROWN FOX
```

will be entered in the buffer as:

```
THE QUICK BROWN FOX
```

When used in COMMAND mode, RUBOUT is equivalent to CTRL/U and cancels the entire command; the Editor then prints a ?, performs a <cr><lf>, and waits for the user to type another command.

5.4.4 CTRL/L: Entering A Form Feed

The form feed character signals the Editor to return to COMMAND mode. A character of this kind is generated by typing L while holding down the CTRL key. This combination is typed while in TEXT mode to indicate that the desired text has been entered and that the Editor should now return to COMMAND mode. The Editor rings the Teletype bell or produces an audible signal on another terminal in response to a CTRL/L to indicate that it is back in COMMAND mode. If the Editor is already in COMMAND mode when CTRL/L is typed, no bell or signal will sound. CTRL/G is equivalent to CTRL/L (except in the case of a SEARCH command, as explained later).

5.4.5 Dot (.): Identifying the Current Line

The Editor keeps track of the implicit decimal number of the line on which it is currently operating. At any given time, the dot, which is produced by typing the period key, represents this number and may be used as an argument in a command. For example:

```
.L
```

means list the current line, and

```
.-1,.+1L
```

means list the line preceding the current line, the current line, and the line following it; then update the current line counter to the decimal number of the last line printed. The current line counter, represented by the dot, is generally updated as follows:

1. After a READ or APPEND command, dot is equal to the number of the last line in the buffer.

2. After an INSERT or CHANGE command, dot is equal to the number of the last line entered.

3. After a LIST or SEARCH command, dot is equal to the number of the last line listed.

4. After a DELETE command, dot is equal to the number of the line immediately after the deletion.

5. After a KILL command, dot is equal to zero.

6. After a GET command, dot is equal to the number of the line printed by the GET.

7. After a MOVE command, dot is not updated and remains whatever it was before the command.

5.4.6 Slash (/): Identifying the Last Line

The slash (/) symbol has a value equal to the decimal number of the last line in the text buffer. It may also be used as an argument in a command. For example:

```
10,/L
```

means list from line 10 to the end of the buffer.

5.4.7 LINE FEED: Identifying the Next Line

Commands and lines of text are terminated by the RETURN key which generates a carriage return/line feed combination. LINE FEED characters are completely ignored when input is on paper tape. During output, the Editor automatically punches a LINE FEED following each carriage return.

Typing the LINE FEED while in COMMAND mode is equivalent to typing:

```
.+1L
```

and will cause the Editor to print the line following the current one and to increment the value of the current line counter (.) by one.

5.4.8 ALT MODE: Incrementing the Current Line

Typing the ALT MODE key while in COMMAND mode will cause the line following the current line to be printed and the current line counter (.) to be incremented by one. If the current line is also the last line in the buffer, typing either ALT MODE or LINE FEED will cause a ? to be typed by the Editor to indicate that there is no next line. (Some Teletypes and most other terminals have an escape key (ESC) in place of the ALT MODE; the function is identical for both ESCape and ALT MODE.)

5.4.9 Right Angle Bracket (>): Identifying the Next Line

Typing the right angle bracket (>) while in COMMAND mode is equivalent to typing:

```
.+1L
```

and will cause the Editor to echo > and then print the line following the current line. The value of the current line counter is incremented by one so that it refers to the last line printed.

5.4.10 Left Angle Bracket (<): Identifying the Previous Line

Typing the left angle bracket (<) while in COMMAND mode is equivalent to typing:

```
.-1L
```

and will cause the Editor to echo < and then print the line preceding the current line. The value of the current line counter is decremented by one so that it refers to the last line printed.

5.4.11 Equal Sign (=): Requesting a Value

The equal sign is used in conjunction with the line indicators dot (.) or slash (/). When typed in COMMAND mode it causes the Editor to print the decimal value of the argument preceding it. In this way the number of the current line may be found (.=xxx), or the total number of lines in the buffer (/=xxx) or the number of some particular line (/-8=xxx) may be determined without counting from the beginning.

5.4.12 Colon (:): Requesting a Value

Colon is a lower-case character with exactly the same function as the equal sign (=).

5.4.13 Blank Tape and Leader/Trailer Tape: Processing Paper Tape

Both blank tape and leader/trailer (octal code 200) tape are completely ignored on an input tape, as are line feed characters and RUBOUTs. Line feeds and RUBOUTs are automatically replaced wherever necessary on output, but blank tape and leader/trailer are not. The production and processing of paper tape at the terminal is, of course, specific to the Teletype.

5.4.14 CTRL/I: Tabbing Editor Output

The Editor simulates tab stops at eight-space intervals across the Teletype paper. The user can tabulate by typing I while holding down the CTRL key. A tabulation consists of from one to eight spaces, depending on the number needed to bring the carriage to the next tab stop. This feature facilitates the production of neat columns on output copy.

The tab function is used in conjunction with two Switch Register bits set to allow the user to produce and control tabulations in the text buffer during input and output operations (see Paragraph 5.5). On input (under a READ command), the Editor can replace a group of two or more spaces with a tabulation if the user chooses to set bit 0 on.

On output, it will produce either a tab character followed by a RUBOUT (for timing purposes) or enough spaces to reach a tab stop, depending on the setting of bit 1. The Editor cannot output tab characters unless tabulations have been entered in the buffer either from the keyboard or by setting bit 0 on input.

NOTE

Location 0002 contains the negative (two's complement) of the number of spaces used to simulate tab stops. To change the tabulation, simply change the constant in location 0002 after loading the Editor.

5.5 SWITCH REGISTER OPTIONS

The Editor uses five Switch Register bits in conjunction with input and output commands to control the reading and punching of paper tape. Switch Register bits may be set for a variety of reasons including the following:

1. To select the low-speed (Teletype) or high-speed paper-tape reader or paper-tape punch
2. To suppress output operations
3. To select certain interpretations for tabulation

The selection of the paper tape unit is probably the most critical of these functions. Naturally, if a PDP-8 configuration has a terminal device other than the Teletype, it is necessary to select the high-speed paper-tape reader and punch. If both high- and low-speed devices are supported, the decision might be more complex. Setting Switch Register options allows the user to select one unit for reading and the other for punching.

It is often desirable to be able to interrupt a command before it finishes. For example, if the user mistakenly supplied a LIST command instead of a PUNCH, he may not want to wait for the terminal to list a large amount of text. Setting bit 2 on the console Switch Register allows the user to interrupt any output command and to return immediately to COMMAND mode. Table 5-2 lists options for all relevant Switch Register bits.

5.6 INPUT COMMANDS

Input commands allow source text to be entered into the text buffer area, either from one of the paper-tape readers or from the terminal keyboard. Available input commands are listed in Table 5-3.

Table 5-2
Switch Register Options

Bit	Setting	Meaning
0	0	Read the input tape exactly as is.
	1	Read the input tape and keep track of spaces. Each time two or more successive spaces are found, substitute in the buffer a tabulation for that whole group of spaces; this option affects only the READ command.
1	0	On punching (or listing) text from the buffer, interpret tabulations as an appropriate number of spaces.
	1	Interpret tabulations as a tab character followed by a rubout (Teletype codes 211 and 377).
2	0	Normal operation; all output commands completed as specified.
	1	Suppress list, punch, or search operation. If at any time during execution of an output command this bit is set to 1, output will cease and the Editor will return immediately to COMMAND mode; if this occurs while a line is being searched, any modifications to the line made during that search will be disregarded; the current line counter (.) will be equal to the number of the line being printed or punched at that time. Until the bit is set to 0, any further output command will be ignored.
10	0	Low-speed output; all punching will be performed on the Teletype punch.
	1	High-speed output; all punching will be performed on the high-speed punch.

Table 5-2 (Cont)
Switch Register Options

Bit	Setting	Meaning
11	0	Low-speed input; the READ command expects the source tape to be in the Teletype reader. Do not use the APPEND command to read tapes.
	1	High-speed input; the source tape will be read from the high-speed reader.

Table 5-3
Input Commands

Command	Meaning
R	Read a page of text and append it to the text buffer using the paper tape unit defined by Switch Register bit 11.
A	Append text entered from the terminal to the text buffer.
I	Insert text entered from the terminal before line 1 of the text buffer.
nI	Insert text entered from the terminal before line n of the text buffer.

For all input commands, the Editor is assumed to be in TEXT mode until a form feed character (CTRL/L) is encountered. If CTRL/L is typed or if a full buffer condition occurs, the Editor returns to COMMAND mode.

NOTE

In these commands, the Editor ignores ASCII codes 340 through 376. These codes include the codes for the lower-case alphabet (ASCII 341–372).

5.6.1 R: Reading Paper Tape

The READ command is issued as follows:

Form R

It is used to read a page of text from the paper-tape reader. Depending on the position of Switch Register bit 11, reading will be performed on the high-speed (one) or low-speed (zero) reader. MLE will read the input tape until a form feed character (CTRL/L key combination) is detected or until the Editor senses a text buffer full condition. All incoming text except the form feed is appended to the end of the text buffer. Information already in the buffer remains there.

In the case of input from the high-speed reader, the end of the tape will be interpreted as a form feed if an actual form feed character does not appear on the tape; the Editor will return to COMMAND mode. In the case of input from the low-speed reader, a form feed must be entered from the keyboard to return the Editor to COMMAND mode if an actual form feed character does not appear on the tape. If this is not done, the READ command remains in effect, and all subsequent commands will be interpreted erroneously as text and appended to the text just read from tape.

Any RUBOUT encountered during a READ command will be ignored, as described in the discussion of the special Teletype keys.

The appropriate paper-tape reader unit must be turned on and positioned to read at the time the READ command is issued. For the low-speed reader, do the following:

1. Set the paper-tape reader switch to STOP or FREE.
2. Release the plastic cover of the reader unit and place the tape over the read station with the small sprocket holes over the sprocket wheel. Close the cover.
3. Push the paper-tape reader switch to START and release.

For the high-speed reader, do the following:

1. Turn the reader unit on.
2. Turn the control knob to raise the tape retaining lever.
3. Place a fan-folded tape in the right-hand bin.
4. Place several folds of leader in the left-hand bin and position the tape so that the sprocket wheel engages the feed holes.

5. Turn the control knob to lower the tape retaining lever.

6. Press the FEED switch briefly to ensure that the tape is properly positioned.

5.6.2 A: Appending Terminal Text

The APPEND command is issued as follows:

Form A

It signals the Editor that the text which is entered next from the terminal keyboard is to be appended to the text already in the buffer. If the buffer is empty at the time the command is issued, a new file is created. This effectively generates a symbolic program on-line by accepting program text from the keyboard. On receiving the APPEND command, MLE enters TEXT mode to accept as much text as the user enters (until the buffer area is full). To return to COMMAND mode, a form feed (CTRL/L key combination) is typed.

A RUBOUT character encountered during execution of an APPEND command (i.e., while program text is being entered) will delete the last typed character. Repeated RUBOUTs will delete from right to left up to but not beyond the beginning of the current line.

5.6.3 I: Inserting Text in the Buffer

The INSERT command causes text to be read from the terminal keyboard and inserted into the buffer in the specified position. It is issued as follows:

Form [n] I

If a command of the form nI is entered, text from the keyboard will be inserted in the buffer just before the line implicitly numbered n. If a simple I command is typed, text will be inserted at the very beginning of the text buffer, just before line 1.

The Editor enters TEXT mode to accept input, and the first line typed becomes the new line n. Both the line count and the numbers of all lines following the insertion are increased by the number of lines inserted; the value of the current line counter (.) is equal to the number of the last line inserted using the I command. To re-enter COMMAND mode, the form feed (CTRL/L combination) must be typed terminating TEXT mode. If CTRL/L is not typed, all subsequent commands will be interpreted erroneously as text and entered in the program immediately after the intended insertion. The following example illustrates the use of the INSERT command. The text buffer is assumed to contain the following:

Implicit Line Number	Text
1	AAA
2	BBB
3	CCC

The following command is given

```
3I
ABA
BAA
BAB
```

To insert text before line 3 and cause the following:

Implicit Line Number	Text
1	AAA
2	BBB
3	ABA
4	BAA
5	BAB
6	CCC

Next, the following command causes two lines of text to be inserted before the old line 1:

```
I
AA0
A0A
```

Implicit Line Number	Text
1	AA0
2	A0A
3	AAA
4	BBB
5	ABA
6	BAA
7	BAB
8	CCC

5.7 OUTPUT COMMANDS

Output commands are available for both listing and punching purposes. Both kinds of commands provide for the output of part or all of the contents of the text buffer. LIST commands facilitate examining the text by producing output on the terminal keyboard. PUNCH commands output leader and trailer tape, form feeds, corrected text, or duplication of pages of an input tape on the paper-tape punch. Neither LIST nor PUNCH commands affect the contents of the buffer in any way. All listing operations can be interrupted by setting Switch Register bit 2 on.

5.7.1 L: Listing on the Terminal Printer

The LIST command causes part or all of the contents of the text buffer to be listed on the terminal. A LIST command is constructed as follows:

Form [[m,]n]L

where m and n are optional arguments as defined in Table 5-4, which summarizes different forms of the LIST command:

Table 5-4
LIST Commands

Command	Meaning
L	List the entire page; this causes the Editor to list the entire contents of the text buffer on the terminal.
nL	List line n; this line will be printed followed by a carriage return and a line feed.
m,nL	List lines m through n inclusive (m must be less than n); lines m through n will be printed on the terminal.

The Editor remains in COMMAND mode after a LIST command and the value of the current line counter is updated to be equal to the number of the last line printed. Some examples of the LIST command might be helpful. The contents of the text buffer at the time the commands are issued as follows:

Implicit Line Number	Text
1	AAA
2	BBB
3	CCC
4	DDD

The following illustrates interspersed command input and text output as in actual terminal interactions:

```
2L
BBB
1,3L
AAA
BBB
CCC
L
AAA
BBB
CCC
DDD
```

5.7.2 P: Punching Out Paper Tape

The PUNCH command causes part or all of the contents of the text buffer to be punched-out using either the low-speed or high-speed paper-tape punch. The device selection depends on the setting of Switch Register bit 10 at the time the PUNCH is issued. If bit 10 is set on, the high-speed device is used; otherwise the Teletype punch is selected.

The PUNCH command is constructed as follows:

Form [[m,]n]P

where m and n are optional arguments as defined in Table 5-5, which summarizes different forms of the PUNCH command. This table also illustrates utility commands used for punching purposes.

Table 5-5
PUNCH Commands

Command	Meaning
P	Punch the entire contents of the text buffer using the punch unit defined by Switch Register bit 10.
nP	Punch line n only.
m,nP	Punch lines m through n inclusive (where m must be less than n).
F	Punch four blanks, a form feed character, and approximately two inches of leader/trailer tape.
T	Punch four inches of leader/trailer tape.

Table 5-5 (Cont)
PUNCH Commands

Command	Meaning
N	Punch the entire contents of the text buffer, issue a FORM FEED command, erase the contents of the text buffer, and read a new page of text into the text buffer using the appropriate paper-tape reader unit (perform P, F, K, and R).
nN	Perform P, F, K, and R n times in sequence.

Because the appropriate paper-tape punch unit must be readied before tape is punched, the Editor will cause the computer to be temporarily halted when a PUNCH is supplied. After the unit has been readied, the user should press the CONTinue key on the console to cause punching to begin.

To position tape in the low-speed punch, do the following:

1. Turn the Teletype control knob to LOCAL.
2. Turn the punch unit on.
3. Press the HERE IS key on the Teletype to produce several inches of reader tape.
4. Turn the punch off.
5. Turn the Teletype control knob to LINE.
6. Turn the punch unit on.

For the high-speed punch, do the following:

1. Turn the punch unit on.
2. Press the FEED switch briefly to ensure that the tape is properly positioned; this switch will advance the tape with only its feed holes punched.

The Editor remains in COMMAND mode after a punching operation, and the value of the current line counter is updated to be equal to the number of the last line punched. PUNCH commands do not cause a form feed character to be output following the text. An explicit form feed must be supplied using the F command.

MLE is designed to minimize the possibility of illegal or meaningless characters being punched into a source tape; therefore the illegal codes 340–376 and 140–177 and most illegal control characters will not be punched. This provides a means of correcting a tape containing illegal characters by simply reading this tape using the Editor and by subsequently punching it out.

5.7.3 F: Punching a Form Feed

The FORM FEED command is issued as follows:

Form F

and is used to punch the following on a paper tape, using either the high-speed or low-speed unit.

1. Four blanks.
2. A form feed character.
3. Approximately two inches of blank tape.

The Editor does not cause the computer to halt when a FORM FEED command is encountered. To avoid the insertion of extraneous characters on the paper tape when the low-speed paper-tape punch has been selected, the user should follow this sequence when issuing a FORM FEED command:

1. Turn the punch off.
2. Type F followed by RETURN.
3. Turn the punch on.

5.7.4 T: Punching a Paper Tape Trailer

The TRAILER command is issued as follows:

Form T

and is used to punch a paper-tape trailer consisting of approximately four inches of blank tape on either the high-speed or low-speed unit.

The Editor does not cause the computer to halt when a TRAILER command is encountered. To avoid inserting extraneous characters on the paper tape when the low-speed paper-tape punch has been selected, the user should follow this sequence when issuing a TRAILER command:

1. Turn the punch off.
2. Type T followed by RETURN.
3. Turn the punch on.

5.7.5 N: Combining P, F, K, and R Commands

The NEXT command is a utility command which is used to read in the next page of text from paper tape one or more times by combining the functions of four distinct MLE commands. It is issued as follows:

Form nN

and performs the functions of the four commands in Table 5-6 in sequence.

Table 5-6
NEXT Command Functions

Command	Function
PUNCH	Punch the entire contents of the text buffer using the punch unit defined by Switch Register bit 10.
FORM FEED	Punch four blanks, a form feed character, and approximately two inches of blank tape.
KILL	Erase the contents of the text buffer.
READ	Read a page of text into the text buffer using the reader unit defined by Switch Register bit 11.

If a simple N command is given, this sequence will be performed only once. If nN is supplied, the sequence will be performed n times.

Because the appropriate paper-tape punch unit must be readied before tape is punched, MLE will cause the computer to be temporarily halted when a NEXT is supplied. After the unit has been readied, as just described, the user should press the CONTinue key on the console. If an nN command is given, the halt will occur only before the first cycle. If n is greater than the number of pages of input tape, the commands will proceed in the specified sequence until the end of the input tape is read. The Editor will then return to COMMAND mode if the unit being used is the high-speed punch. If the Teletype unit has been selected, the user must type CTRL/L to return to COMMAND mode when the tape runs out.

5.8 EDITING COMMANDS

Editing commands allow source text to be deleted, changed, moved, and expanded in the text buffer. In addition to standard replacement and deletion features, the Editor described in this chapter facilitates such advanced editing features as moving a block of text from one part of the buffer to another or searching the text buffer for specific characters or for lines containing tags.

5.8.1 C: Changing Lines in the Text Buffer

The CHANGE command allows the user to replace one or more lines in the buffer with text entered on the terminal keyboard. It is issued as follows:

Form [m,]nC

where m is optional and n is a required argument, supplied as shown in Table 5-7.

Table 5-7
CHANGE Commands

Command	Meaning
nC	Delete line n and replace it with the line(s) that follow.
m,nC	Delete lines m through n and replace them with the lines that follow (m must be less than n).

When the Editor receives a CHANGE command, it deletes the specified lines and then enters TEXT mode to accept text typed by the user to replace the deleted line(s).

Once in TEXT mode, RUBOUTs can be used to erase characters in the inserted text. It is not necessary to replace the changed text with the same number of lines that were deleted. Since lines are automatically renumbered, the user may enter any number of new lines to replace the old. The

line count will automatically be updated. In addition, after a CHANGE has been performed, the value of the current line counter (.) is equal to the number of the last line of the inserted text.

An example of changing text in the buffer might be helpful. If the contents of the buffer are as follows:

Implicit Line Number	Text
1	AAA
2	BBB
3	CCC
4	DDD

typing the command:

```
2C
B11
B22
B33
```

will result in the following:

Implicit Line Number	Text
1	AAA
2	B11
3	B22
4	B33
5	CCC
6	DDD

The next command:

```
4,6C
BBB
```

causes the following:

Implicit Line Number	Text
1	AAA
2	B11
3	B22
4	BBB

To return to COMMAND mode after replacing text with the CHANGE command, the user types a form feed (CTRL/L) to terminate input from the keyboard.

5.8.2 D: Deleting Lines of Text

The DELETE command causes the deletion of one or more lines of text in the buffer. It is issued as follows:

Form [m,]nD

where m is optional, and n is a required argument, supplied as shown in Table 5-8.

Table 5-8
DELETE Commands

Command	Meaning
nD	Delete line n.
m,nD	Delete lines m through n (m must be less than n).

When the Editor has performed the specified deletion, it automatically renumbers all succeeding lines reducing their implicit numbers by the number of lines deleted. When a command of the form

m,nD

is performed, the line following n becomes the new line m, and the rest of the lines are renumbered accordingly.

Following is an example of deletion of text in the buffer:

Implicit Line Number	Text
1	AAA
2	BBB
3	CCC
4	DDD

The command:

2,3D

changes the text buffer to the following:

Implicit Line Number	Text
1	AAA
2	DDD

5.8.3 G: Getting a Tagged Line

The GET command is used to locate the next line in the text buffer with a tag associated with it. It is issued as follows:

```
Form      [n]G
```

If a simple G command is supplied, the Editor begins the search for the next tagged line with the line following the current line (the value of the current line counter). If the optional argument precedes the G, the search begins at line n, testing it and each succeeding line. The line which first passes the test will be printed on the terminal by the Editor.

A tagged line is defined as one which does not begin with a tab, slash, or space character. Usually such lines begin with tags or labels, but lines not indented will also pass the test. For example:

```
          LAB              This is the current line.
          INP2
/THIS IS A COMMENT
HERE,     LMA              This line will be printed by GET.
          RAR
          JFZ     HERE
*20#377                    This line will be printed next if another GET is entered.
```

If the GET command succeeds in finding a tagged line, the current line counter is updated. However, if the GET reaches the end of the buffer without finding a tagged line, the current line counter (i.e., the value of the line counter before the GET was issued) is preserved. A question mark (?) is typed on the terminal to indicate that tests were not successful. The Editor remains in COMMAND mode after all GET operations.

The following is the current text buffer:

Implicit Line Number	Text
1	LAB
2	INP2
3	/OLD VERSION
4	VER1, LMA
5	RAR
6	VER2, JFZ VER1
7	HLT

Sample GET commands, with the initial line counter at line 5, are given below:

```
1G
VER1,     LMA
G
VER2,     JFZ       VER1
G
?
```

5.8.4 K: Killing the Text Buffer

The KILL command causes the entire page in the text buffer to be erased. It is issued as follows:

```
Form      K
```

The values of special characters / (last line in buffer) and . (current line counter) are set to zero. The Editor remains in COMMAND mode after the buffer has been erased.

5.8.5 M: Moving Text in the Buffer

The MOVE command is used to move a line or lines from one location in the text buffer to another. It is issued as follows:

```
Form      m,n$jM
```

where m, n, and j are all required arguments. The MOVE command causes text buffer lines m through n inclusive to be moved to the position just before line j. The user must ensure that m is less than n. After the MOVE is complete, lines are renumbered but the value of the current line counter (.) is not changed.

The following example:

```
1,10$20M
```

causes lines 1 through 10 to be moved to just before line 20. If the user wishes to move a single line to a new

location, three arguments must nevertheless be supplied; this is accomplished by specifying the same value for both m and n, as in the following:

```
15,15$25M
```

This moves line 15 to before line 25. To insert text at the beginning of the buffer, simply specify 1 as the j argument:

```
20,30$1M
```

To move text to the end of the current text in the buffer, the user supplies a special j specification of /+1 (i.e., end-of-buffer + 1), as in the following:

```
1,10$/+1M
```

This moves lines 1 through 10 to the end of the current contents of the text buffer.

MLE remains in COMMAND mode after performing a MOVE command. Moving lines does not affect the size of the buffer in any way since lines are merely rearranged, not added.

The following example illustrates the use of several versions of the MOVE command. The initial contents of the buffer follows:

Implicit Line Number	Text
1	AAA
2	BBB
3	CCC
4	DDD
5	EEE
6	FFF

First, move line 5 to the beginning of the buffer:

```
5,5$1M
```

The buffer now looks like this:

Implicit Line Number	Text
1	EEE
2	AAA
3	BBB
4	CCC
5	DDD
6	FFF

Next, move lines 1, 2, and 3 to the end of the buffer:

```
1,3$/+1M
```

Implicit Line Number	Text
1	CCC
2	DDD
3	FFF
4	EEE
5	AAA
6	BBB

Finally move lines 3 and 4 to before line 2:

```
3,4$2M
```

Implicit Line Number	Text
1	CCC
2	FFF
3	EEE
4	DDD
5	AAA
6	BBB

5.8.6 S: Searching the Text Buffer

The SEARCH command is used to examine all or part of the text buffer for a specified character. It is issued as follows:

Form [[m,]n]S

where m and n are optional arguments, supplied as shown in Table 5-9.

Table 5-9
SEARCH Commands

Command	Meaning
S	Search the entire buffer for all occurrences of a particular character.
nS	Search line n for an occurrence of a particular character.
m,nS	Search lines m through n for an occurrence of a particular character.

The SEARCH command is very different from the MLE commands already discussed in this manual. It is far more interactive and therefore facilitates far more complex editing operations.

The SEARCH command string itself does not specify the character for which the Editor is to search. That character must be entered after the command, terminated by a RETURN key, has been typed. No search operations will begin until a character is typed; MLE will simply halt waiting for input. The user is expected to type a single character as the object of the search, but the typed character will not be echoed. Instead, the Editor will respond either by typing a question mark (?) to indicate that the desired character cannot be found in the specified line or lines, or by supplying part of the first line encountered which contains the specified character, from the first character in the line to the position where the specified character occurred.

An example may be helpful. Assume that line 10 contains AB**N&&:

```
10S
AB*
```

In this example, the user requested a search of line 10 for the asterisk character (*) entered in the place where A appears. The * is not echoed, but the Editor locates that character in line 10 and types out line 10 from the beginning to the position in which * occurs. At this point, the user may choose one of many options which affect the selected line or the text in the rest of the buffer. These options are listed in Table 5-10.

The options shown in Table 5-10 apply particularly to the nS form of the SEARCH command. In almost all cases, however, they are applicable to the other command formats as well. If the following form:

```
m,nS
```

is used, indicating a search in lines m through n inclusive, one major difference occurs. If RETURN is typed in response to the Editor's display of the search character line, the entire unprinted portion of the line is deleted and the line is terminated. However, the search will continue on the next line.

By typing CTRL/G to change search characters, all editing of a single line may be performed in one pass. Typing CTRL/G twice will cause the search to terminate since the search character will now be BELL, which is not stored in the buffer.

Table 5-10
SEARCH Options

Option	Action
CTRL/U	Delete the entire printed portion of the line with the CTRL/U (erase) character. This preserves the unprinted portion of the line. A carriage return/ line feed character is automatically generated.
RETURN	Delete the entire unprinted portion of the line, and terminate the search. MLE resumes in COMMAND mode.
RUBOUT (\)	Delete one character from right to left for each RUBOUT typed; only printed characters are affected.
character(s)	Insert typed character(s) after the last printed character.
LINE FEED	Insert a carriage return/line feed character between the printed and unprinted portions of the line.
CTRL/L	Continue the search to the next occurrence of the specified character; when the next line portion has been printed, all options are available again.
CTRL/G character	Change the search character to character and continue the search to the first occurrence of this character.

Specifying a simple S command causes the entire buffer to be searched for occurrences of a single search character. It should be remembered, however, that as with CHANGE, every SEARCH command uses additional buffer space for storage of the new line. This is necessary, since the program can have no prior knowledge of whether the size of the line will be less than, greater than, or equal to that of the old line, and it must therefore assume that it will be greater. The entire text buffer is searched and a new image of this text is created in core; it is guaranteed to occupy the same space as before, or somewhat less, since all deleted spaces have been removed. The only prerequisite to condensing the text image is that there be enough core space left to contain another image of the edited text. The options available in a simple S specification are exactly the same as those for the m,nS version of the command.

5.9 EDITOR OPERATING PROCEDURES

This paragraph summarizes operating procedures for loading, using, and restarting the Editor. These include:

1. Loading the Editor into core.
2. Generating a source program off-line.
3. Loading a tape using the Editor.
4. Restarting the Editor.
5. Editing a tape.
6. Punching a tape.

5.9.1 Loading the Editor into Core

MLE is loaded into core using the Microprocessor Host Loader (MHL). This loading procedure is illustrated in Figure 5-1. This flowchart also illustrates the selection of Switch Register bits to specify the appropriate paper-tape reader/punch and demonstrates actions required to generate a program on-line. After the Editor is loaded, it resides in core in locations 0200–1624.

5.9.2 Generating a Symbolic Program Off-Line

Figure 5-2 illustrates the generation of a symbolic program off-line using the Teletype low-speed punch. This procedure is generally much slower than using MLE, but in the case of creating extremely short programs it may prove advantageous. Leader/trailer tape made up of octal 200 code (rather than the blank tape produced by the HERE IS key) may be generated off-line by pressing the SHIFT, CTRL, REPT and P keys in order and holding all down simultaneously.

5.9.3 Loading a Symbolic Tape Using the Editor

Figure 5-3 illustrates how to load a symbolic tape using either the low-speed or high-speed paper-tape reader. MLE will continue to read a tape until a form feed code is encountered (see the section on FORM FEED). As soon as it recognizes the form feed character, MLE enters COMMAND mode and rings the Teletype bell or produces an audible signal on another terminal to indicate that it is ready to accept a command.

NOTE

When using the Teletype reader, if the form feed code is encountered before the symbolic tape has been completely read in (as indicated by the bell or signal), turn off the paper-tape reader. Otherwise, characters on tape will be interpreted as commands to the Editor. The section of tape read in up to the form feed code should then be edited before proceeding with the remainder of the tape.

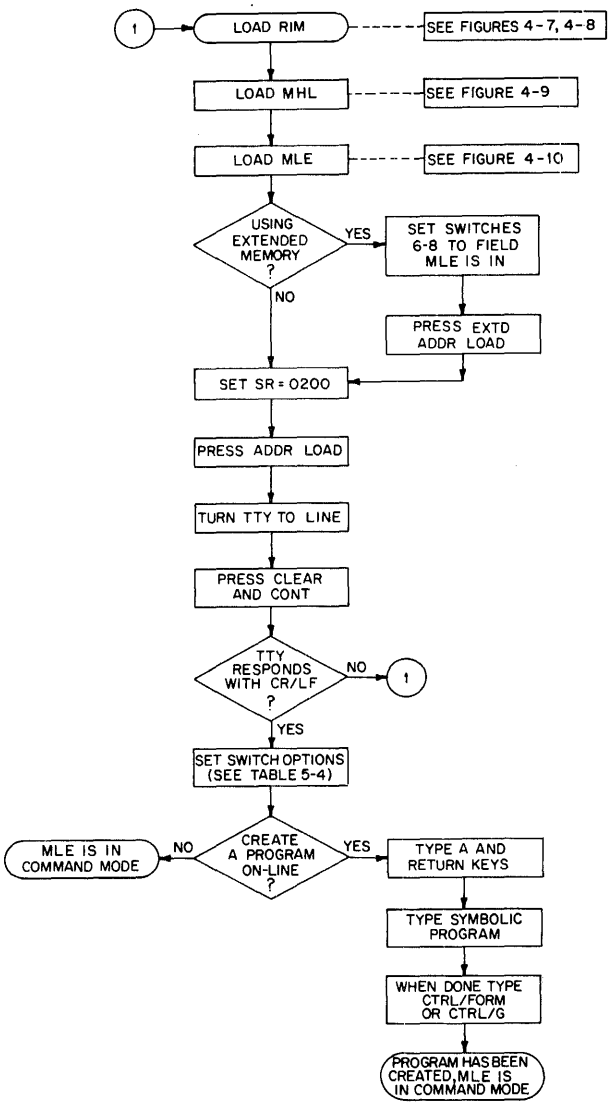


Figure 5-1 Loading the Editor into Core

5.9.4 Restarting the Editor

If the user halts the computer for any reason during the editing process, MLE may be restarted. The user has the option of either clearing the text buffer or restarting so that the text in the buffer is maintained.

1. To clear the buffer, set 0176 in the Switch Register; press ADDR LOAD, CLEAR, and CONT.

2. To restart without clearing the buffer, set 0177 in the Switch Register; press ADDR LOAD, CLEAR, and CONT.

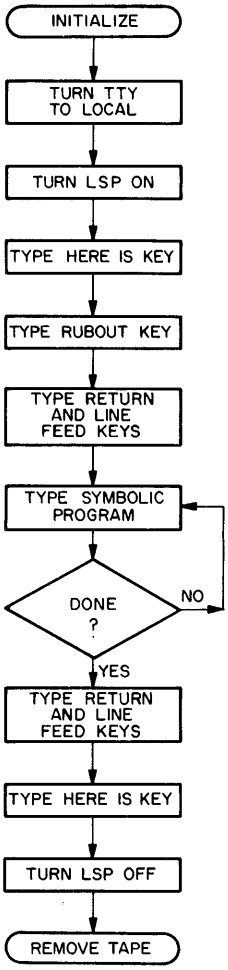


Figure 5-2 Generating a Symbolic Program Off-Line

3. Set 0200 in the Switch Register.

4. Press ADDR LOAD, CLEAR, and CONT.

This has the effect of restarting the Editor in COMMAND mode.

5.9.5 Editing the Source Program

Actual editing procedures depend, of course, on the particular programs being created or modified. A general approach is illustrated in the example presented in Paragraph 5.10. For input, editing, and output commands to the Editor, refer to the specific paragraphs just discussed. Also observe the following operating notes and precautions:

1. Terminate each command to the Editor by typing the RETURN key. This directs MLE to execute the command.

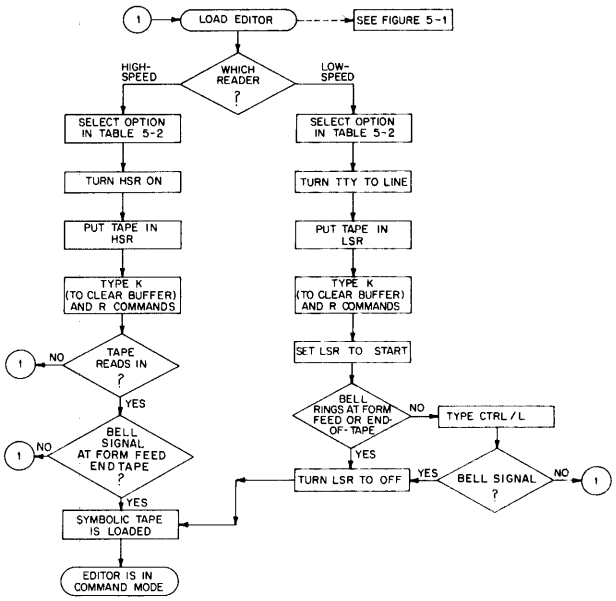


Figure 5-3 Loading a Symbolic Tape Using the Editor

2. After a command to insert, change, or append text to a symbolic program has been executed, MLE remains in TEXT mode until the operator types CTRL/L on the terminal. This generates the form feed code, which tells the Editor to return to COMMAND mode.

3. The Editor senses a buffer full condition (buffer capacity is approximately 60 lines of commented text or 340 lines of uncommented text) when, after completing input of a text line, it finds that characters have been packed in the last 128 locations in the text buffer. When this condition occurs, MLE rings the Teletype bell or produces an audible signal on another terminal five times and exits to COMMAND mode. The user then has a choice of deleting text and continuing editing as usual, or attempting to input more than 200 additional characters. After each line, the buffer full alarm will precede a return to COMMAND mode. When no more characters can be packed, the Editor will again output the alarm five times and will exit from the input routine. Any further attempts to input text will be answered in the same manner until deletions have made room for text input. Although characters are received through the input device, they probably will not be appended as text.

If the Editor runs out of buffer space while searching a line, the unsearched portion of the line may be lost or the text line counter may be incorrectly set during the buffer full exit so that the Editor recognizes one more line of text in the buffer than actually exists. Occurrence of the latter will cause an error return after or during any output operation involving the last line (for example, an N operation will be terminated as soon as the text buffer is punched). After the error return, the line counter will contain the correct value.

Users should note that all such problems may be avoided by logically segmenting a program on paper tape into "pages" of 50 to 60 lines. This is accomplished by punching groups of 50 lines followed by a form feed character.

4. The Editor may be stopped at any time by pressing the HALT key on the console; to continue, press the CONTinue key.

5.9.6 Punching the Corrected Symbolic Tape

The procedure for punching out the corrected symbolic tape depends to some extent on the user's requirements. The general sequence is given below and in Figure 5-4.

1. Enter output commands to punch blank tape for leader/trailer purposes (T), form feeds (F), the appropriate lines of text (m,nP), or the entire text buffer (P).

2. Following the PUNCH command, the computer will halt giving the user the opportunity to check Switch Register bits and to turn on the appropriate punch if he has not done so already. Punching is initiated by pressing the CONTinue key on the console.

NOTE

If the low-speed punch is used, it should be turned off during the typing of commands; otherwise these codes will be punched on the symbolic tape.

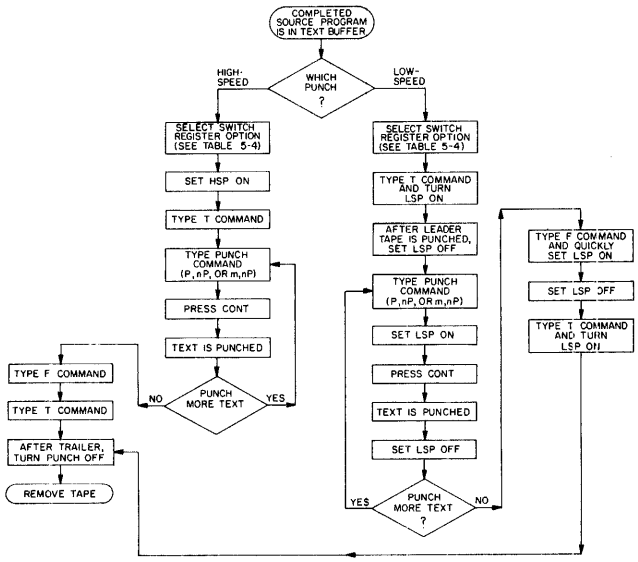


Figure 5-4 Generating a Symbolic Tape Using the Editor

3. Punching the symbolic program does not delete it from memory. The page remains in the text buffer in core until the KILL command is given to erase it. If the user wants to read another tape into the buffer he must first delete the entire page of text (K). Remember that the recommended page length, as delimited by the form feed, is approximately 60 lines of heavily commented text. However, MLE can accept more text if necessary.

5.10 EDITING EXAMPLE

This paragraph illustrates the reading, editing, and punching of a short assembly language program. MLE is loaded and started as described in the previous section. The paper tape containing the program to be edited is then read in and listed.

```
R
L
        *100#0          /CLEAR ACC
        XR%             /LOAD DATA FROM MEMORY
        JTS     NEG     /JUMP TO NEG IF DATA NEGATIVE
ROTAT,  RAR             /ROTATE DATA RIGHT
```

The user wishes to append text to this incomplete program and does so by typing:

```
A
        LMA                 /STORE RESULT
NEGT,   RAR/ROTATE DATA RIGHT
        OR%     200         /SET THE LEFTMOST BIT TO 1
        LAM                 /STORE RESULT
        HLT
        $
```

To obtain a fresh listing of the text, the user types L again:

```
L
        *100#0
        XR%                 /CLEAR ACC
        ADM                 /LOAD DATA FROM MEMORY
        JTS     NEG         /JUMP TO NEG IF DATA NEGATIVE
ROTAT,  RAR                 /ROTATE DATA RIGHT
        LMA                 /STORE RESULT
NEGT,   RAR/ROTATE DATA RIGHT
        OR%     200         /SET THE LEFTMOST BIT TO 1
        LAM                 /STORE RESULT
        HLT
        $
```

A few errors are noticed immediately and are corrected.

```
2L
        XR%                 /CLEAR ACC
2C
        XRA                 /CLEAR ACC
6L
        LMA                 /STORE RESULT
7I
        HLT                 /DONE
7,11L
        HLT                 /DONE
NEGT,   RAR/ROTATE DATA RIGHT
        OR%     200         /SET THE LEFTMOST BIT TO 1
        LAM                 /STORE RESULT
        HLT
11D
11I
        HLT                 /DONE
10L
        LAM                 /STORE RESULT
10C
        LMA     0           /STORE RESULT
```

The user decides that the program is ready for assembly and formats the output tape in the following way:

```
P
F
T
```

This produces the text of the program followed by a form feed, followed by a trailer. The computer halts after the PUNCH command is issued and waits for the paper-tape unit to be readied. When this has been performed, the CONTinue key on the console must be pressed.

The first assembly of this program produces the following error messages on pass 1. See Chapter 6 for detailed descriptions of all statement syntax and errors.

```
AD        AT 00 000
IC  245   AT 00 010
OA        AT 01 011
US  NEG   AT 00 002
```

Addresses in AT specifications are given in octal offsets within blocks. These messages can be matched to source statements as follows:

Message	Statement	Meaning
AD AT 00 000	*100#0	Address specified in origin is out of range. Maximum block-offset is 77#377; origin is set to zero so the addresses in the rest of the program are not affected.
IC 245 AT 00 010	OR%	Illegal character (%) is not recognized by the Assembler. ASCII representation of % (245) is supplied; ORI was simply misspelled.
OA AT 00 011	LMA 0	Operand error; LMA load instruction does not take an operand and 0 is supplied.
US NEG AT 00 002		Undefined address. No NEG label can be found in the program; statement with label NEGT is probably in error.

The program can now be corrected as follows:

```
1L
        *100#0
1C
        *35#40
G
ROTAT,  RAR             /ROTATE DATA RIGHT
6G
NEGT,   RAR/ROTATE DATA RIGHT
.=
8
8C
NEG,    RAR             /ROTATE DATA RIGHT
1,12S
        OR%     200     /SET THE LEFTMOST BIT TO 1
.=
9
9C
        ORI     200     /SET THE LEFTMOST BIT TO 1
```

A few commands require explanation. The interaction:

```
1,12S
OR%
```

does not show the typing of the search character, the %. The user types % right after the carriage return following 1,12S, but this character is not echoed. To return to COMMAND mode after the line is found and the string OR% printed, the user types CTRL/L. The .= command requests, in two cases, that the decimal number of the current line be printed so that it can be changed. Then, the user verifies that all corrections have been made by listing the entire program.

```
L
        *35#40
        XRA             /CLEAR ACC
        ADM             /LOAD DATA FROM MEMORY
        JTS     NEG     /JUMP TO NEG IF DATA NEGATIVE
ROTAT,  RAR             /ROTATE DATA RIGHT
        LMA             /STORE RESULT
        HLT             /DONE
NEG,    RAR             /ROTATE DATA RIGHT
        ORI     200     /SET THE LEFTMOST BIT TO 1
        LMA             /STORE RESULT
        HLT             /DONE
        $
```

The program is then punched-out for assembly and the text buffer cleared for subsequent use.

```
P
F
T
K
```

CHAPTER 6
MICROPROCESSOR LANGUAGE ASSEMBLER

6.1 INTRODUCTION TO THE ASSEMBLER

The Microprocessor Language Assembler (MLA) is a powerful paper tape-oriented system program which is used to assemble source code on the PDP-8 into binary output which is loaded and executed on the Processor Module. Input to the Assembler is usually prepared with the aid of the PDP-8 Microprocessor Language Editor (MLE) described in Chapter 5; it can also be generated off-line. The Assembler expects all input to be in paper tape form, and reads input tapes, dynamically selecting the high-speed paper-tape reader. If a high-speed device is not part of the system configuration, the Assembler reads from the low-speed reader associated with the Teletype. At the user's option, assembled code can be output in punched paper tape form or can be listed on the terminal printer. Diagnostic messages are displayed to indicate errors in syntax, warnings, or actions taken by the Assembler.

The Assembler operates in three passes to produce binary code suitable for testing and executing on the Microprocessor Series, and outputs a variety of program, diagnostic, and symbol table listings. The PDP-8 therefore serves as the host computer for the preparation and development of executable code. In an effort to aid those who have used other PDP-8 Assemblers, such as PAL III, MLA has been designed to conform to such existing Assemblers in terms of its character set, error messages, available operators, and construction of statements, symbols, and expressions whenever possible.

6.2 OVERVIEW OF THIS CHAPTER

Information necessary for developing and assembling programs for the Microprocessor Series is presented in this chapter and structured as a series of seven major topics:

1. Character set, including use of numbers and symbols.
2. Language syntax, focusing on construction of statements and use of instructions, operators, and expressions.
3. Internal Assembler characteristics, including mapping of memory and use of tables.
4. Instruction set comprising index register, accumulator, program-counter, stack-control, input/output, and machine instructions.
5. Pseudo-instructions used to assign values to storage, to indicate end of program or tape, to define operation codes, and to perform other Assembler functions.
6. Operating instructions necessary to load the Assembler and the source program, to run and restart the Assembler, and to define desired output.
7. Error messages produced during assembly, with suggested possible reasons for occurrence.

This information, when supplemented by specific details of Editor (Chapter 5) and Loader (Chapter 7) usage, should provide the necessary background for developing source programs and preparing them for execution on the target module.

6.3 BASIC CHARACTER SET

This paragraph defines the set of characters allowed in Assembler input, defines special characters used for paper tape or listing control, and describes construction of numbers and symbols.

6.3.1 Legal Source Text Characters

The following list summarizes all characters that may be included in a source program and are accepted by the Assembler:

- Alphabetic characters A through Z
- Numeric characters 0 through 9
- Selected special (printing) characters, such as:

Character	Meaning
Space	Space
+	Plus
-	Minus
,	Label terminator
=	Is replaced by
&	Logical AND
!	Logical OR
↑	High byte-selection operator
;	Data separator
$	End of source program
.	Current value of location counter
/	Comment initiator
<	Used to enclose numeric representation of ASCII character
>	
*	Used to reset location counter
#	Block-offset operator

- Selected special (nonprinting) characters, such as:

  TAB
  Carriage return
  Form feed

- Ignored characters, often produced by the editor, such as:

  Blank tape
  RUBOUT
  Leader/trailer (octal code 200)
  Line feed

Characters other than those listed above may be included in an assembly language source program only when included after a comment character (/) or within a TEXT literal. The presence of an illegal character in Assembler code causes the following message to be displayed during Assembler pass 1:

IC xxx AT yy zzz

where IC identifies this as an illegal character message, xxx is the ASCII value of the illegal character, and yy zzz is the address at which the error was encountered, in block-offset notation. This message causes the illegal character and the rest of the current line to be ignored while assembly proceeds. If an illegal character is detected in a symbol, however, the symbol is assumed to terminate at the position where the illegal character occurred.

6.3.2 Format Control

Certain keys on the terminal may be used to affect the format of an Assembler listing by skipping to the next line or by inserting blank lines or spaces. These special characters include the following:

Character	Meaning
Form feed	Assembler outputs 12 blank lines in the output listing when form feed (CTRL/L on the terminal) is entered. It is used to begin a new page of text and has no effect on the binary output.
TAB	Assembler outputs between one and eight spaces in the output listing line if bit 10 is set in the Switch Register; total number of characters in inserted spaces and prior symbols equals eight. It is also used to line up columns of code (symbols, comments, etc.) in the listing.
Carriage return	Terminates a line in the source program and output listing; used to separate one statement from the next.

6.3.3 Construction of Numbers

The Assembler recognizes numbers in octal, hexadecimal, and decimal form. In all cases, a number must be represented by a string which begins with a digit (0–9). Therefore, hexadecimal A2 is constructed as 0A2.

6.3.4 Construction of Symbols

The user can construct symbols to represent labels in the assembly language program by combining legal letters and digits in a string. Rules for user-defined symbols follow:

1. A symbol must begin with an alphabetic character (A–Z).
2. A symbol may contain any number of letters (A–Z) and digits (0–9), but only the first six characters are recognized.
3. A symbol must be terminated by a comma.
4. The space character is used to delimit a field and may not be embedded in a symbol.

Any of the following are legal symbolic labels:

```
TOTAL,
NEG,
PROD1,
HALT10,
A1B2C3D4E5,      (only A1B2C3 is recognized).
```

```
        NEG,    RAR             /ROTATE DATA RIGHT
                ORI     200     /SET THE LEFTMOST BIT TO 1
                LMA             /STORE RESULT
                HLT             /DONE
```

The following are examples of illegal labels which do not fit the definition of a symbol:

Label	Reason
1ADDUP	First character (1) not alphabetic
SET%X	Illegal character
ADD UP	Embedded space
SET*X	Legal but nonalphanumeric character
NEG	Not terminated directly by comma (spaces not allowed)

6.4 STATEMENT SYNTAX

An assembly language statement consists of a maximum of four fields and is constructed as follows:

Form [label,] [instruction] [operand] [/comment]

All four fields are optional under different circumstances. The instruction is normally a required item in every statement. In many cases an operand field must be included as well. A label is only necessary to identify the object of a branch, but if it is included, the label must be separated from the rest of the statement by a comma (,). The comment field is optional in all cases; if included, the comment must be preceded by a slash (/). Comments may be entered on a line without any other text, as in the following:

```
/FOLLOWING IS A
/COMMENT
```

In this case, the instruction field need not be included.

The order of the fields shown in this form must be preserved, but the particular placement of individual fields is not significant. Spacing is specified to give the impression of tabbing throughout this chapter, but tabbing is, of course, optional, and both of the following examples are interpreted identically by the Assembler. First, with tabbing to columns 1, 9, 17, and 25 for labels, instructions, operands, and comments respectively:

Next, with free-form specification of fields:

```
NEG,RAR/ROTATE DATA RIGHT
ORI 200      /SET THE LEFTMOST BIT TO 1
LMA/STORE RESULT
HLT/DONE
```

The use of standard tabbing does improve the readability of a source listing and is recommended for that reason. But a certain delimiting of fields is necessary whether or not tabbing is performed. Fields must be separated by:

1. A label from an instruction by a comma (followed by as many spaces as desired),
2. An instruction from an operand by at least one space or tab, or
3. An instruction or an operand from a comment by a slash (preceded and followed by as many spaces as desired).

To facilitate a clear and easy-to-read program listing, the following coding practices are recommended. Use of these practices is also recommended to improve the ease of sharing programs among different programmers.

1. If a title or introductory program comment is included, begin the line with a slash and regard the entire line as a comment.

2. Begin all statement labels at the left margin and tab once to the statement's instruction field.

3. Tab once from the margin before typing an unlabeled instruction or pseudo-instruction to align this field with labeled instructions.

4. If an operand is included in the statement, tab twice from the margin before typing it.

5. If a comment is included, tab once from the operand field, twice from the instruction field (if an operand is not required), or three times from the margin (if the comment is effectively continued from the previous line) to line up all comment fields. It is legal to include a full-line comment and to specify a slash as the first character of the line; this is useful in delineating parts of the program.

The four components of a statement are described in greater detail in the paragraphs which follow.

6.4.1 Construction of a Label

A statement label is used to tag an assembly language statement and thereby identify its location so other statements can branch to it or reference it during execution. A label must be a legal symbol, as just defined, and must be immediately followed by a comma, as in the following examples:

```
END1,   HLT
NEG,    LMA
STORE,  ORI     200
```

If a label is included, it must be the first field in a statement preceding the instruction, operand, and comment fields.

All statement labels referenced in an assembly language program must appear in that program. For example, in the following routine:

```
        1#200
        XRA
        ADM
        JTS     NEG
        RAR
        LMA
        HLT
NEGT,   RAR
        ORI     200
        LMA
        HLT
        $
```

The JTS jump instruction references label NEG which does not appear in the program (in actual practice, this may indicate a simple misspelling, as suggested). At the end of Assembler pass 1, the symbol NEG will be undefined and the following message will be printed

```
US NEG AT 01 202
```

where US identifies this as an undefined symbol message, NEG is the symbol used but not defined, and 01 202 is the current location at the time the undefined symbol was first specified (the JTS instruction).

A different error message will be output if the same symbol is used as a label more than once in a single program. In this case, the following message will be printed

```
DT xxxx AT yy zzz
```

where DT identifies this as a duplicate tag message, xxxx is the duplicate symbol, and yy zzz is the location of the second occurrence of the label in block-offset notation. The symbol will not be redefined.

6.4.2 Construction of an Instruction

The instruction field in a statement must be occupied by one of the following:

- Mnemonic MLA instruction
- MLA pseudo-instruction
- User-defined instruction

Assembler instructions and pseudo-instructions are implemented as reserved words and are described in detail in subsequent paragraphs. These words may not be selected as user-defined symbols or used in any other part of a statement except in the comment field, as follows:

```
ENDL,    HLT        /HLT INDICATES END PROGRAM
```

User-defined instructions can be used only after they have been explicitly defined in the user program. Some instructions and pseudo-instructions require the inclusion of operands. If an operand is specified as the object of an instruction, at least one space must separate the two fields. If an operand is not included, the comment field (if specified) must be preceded by a slash. If an instruction is labeled, a comma must separate the label and the instruction.

6.4.3 Construction of an Operand

The inclusion of an operand is required after many Assembler instructions and pseudo-instructions. After an instruction, the operand is usually an octal, hexadecimal, or decimal address, a symbol representing the data to be manipulated, or the address to be referenced when the instruction is executed. An operand is usually included after a pseudo-instruction as the argument of that pseudo-instruction. At least one space must separate the instruction field from the operand field of a statement. A slash must separate the operand from a comment (if included).

6.4.4 Construction of a Comment

Comments can be included in an assembly language program to annotate the functions of particular statements or to document complicated logic for future ease of debugging or recoding. In processing input statements, the Assembler ignores everything from the slash, used to denote the beginning of a comment, to the next carriage return. This means that the comment field must be the last or the only field in a statement. Some examples follow:

```
/THIS IS A SAMPLE PROGRAM
/
          *1#200
          XRA              /CLEAR ACCUMULATOR
          ADM              /LOAD DATA FROM MEMORY
          JTS       NEG    /JUMP TO NEG IF DATA NEGATIVE
          RAR              /ROTATE DATA RIGHT
          LMA              /STORE RESULT
NEG,      HLT              /DONE
```

MLA allows lines to be inserted in the text of a program as shown in the third line of the previous example.

6.5 THE LOCATION COUNTER

The location counter is a special Assembler pointer which is constantly updated during program execution to keep track of the current address. The user ordinarily sets the location counter at the beginning of the program with the origin (*) pseudo-instruction to indicate the address at which execution is to begin. Subsequently, the location counter can be referenced as an operand to set a value or to specify a jump to a location relative to the counter. In the following code

```
*1#200
INP1              /GET STATUS
NDI       200     /MASK BIT 7
JFZ       .-3     /LOOP UNTIL BIT IS SET
```

*1#200 sets the counter initially at block 1 offset 200, and JFZ .-3 indicates a conditional jump to the current location minus 3.

Because MLA is a symbolic Assembler capable of setting the location counter and performing all other operations based on symbolic assignments like

```
*A#B
```

the extensive use of the . to set the location counter is not recommended. Symbolic addresses and values should be used to avoid rewriting the entire program if the program is moved in core and specific locations have been represented throughout the source text. Relative addressing with . should be used only if available symbol table space is very tight.

6.6 EXPRESSIONS AND OPERATORS

This paragraph describes legal operators in the MLA assembly language and discusses the construction and evaluation of expressions.

There are five different kinds of operators available to MLA users:

- Replacement operator =
- Arithmetic operators + and -
- Logical operators & and !
- High byte-selection operator ↑
- Block-offset operator #

6.6.1 Expression Evaluation

Expressions are evaluated by the MLA from left to right without precedence in signed 23-bit arithmetic. Parentheses are not legal characters and cannot be used to impose precedence on an expression; the components must simply be ordered appropriately.

6.6.2 Replacement and Arithmetic Operators

Replacement and arithmetic operators are used in arithmetic expressions to indicate two's complement addition (+), subtraction (-), and replacement (=). Following are two examples of arithmetic expressions:

```
D=A-B+3
C=.+2
```

Arithmetic is carried out on signed 23-bit numbers. Only the lower 14 bits are used as an address in a jump or call instruction. If the high-order bits are set, an error message of the following kind will result

```
AD AT yy zzz
```

where AD identifies this as an address out-of-range message and yy zzz is the value of the location counter when the address was specified in block-offset notation.

6.6.3 Logical Operators

Two logical operators have been implemented for use with the Assembler. These are &, indicating a logical AND operation, and !, indicating a logical OR operation. Logical operators are used in logical expressions of the following forms:

```
A&B
X!Y!Z
A&B!X&Z
X+Y&Z-A!B
```

Logical and arithmetic operations can be mixed as just shown in the fourth example.

6.6.4 High Byte-Selection Operator

The high byte-selection operator ↑ (up arrow) is a post-operator used to indicate selection of the high byte of the entire expression (from the beginning) which the ↑ follows. For example,

```
LHI     A+B↑
```

indicates a load immediate of the high byte of expression A+B to register H. The example below, on the other hand, indicates a load immediate of the low byte of value A to register L.

```
LLI     A
```

By selecting the high byte, this operator performs an effective signed divide by 256. For example, if the following assignments have been made

```
A = 0400
B = 0377
```

the expression

```
A+B+10↑+1
```

results in 0003.

6.6.5 Block-Offset Operator

The block-offset operator # is used to indicate an address in terms of its block number and an offset within that block. For example, an origin (*) pseudo-instruction of the form

```
*2#200
```

sets the location counter to location 200 within block 2 (octal 1200). The example

```
JMP     35#377
```

sets the location counter to block 35 offset 377 (octal 16777).

The conversion from block-offset to octal notation proceeds as in the following. The bit pattern of 35#377 can be represented by:

```
011  101  11   111  111
 3    5   3    7    7
```

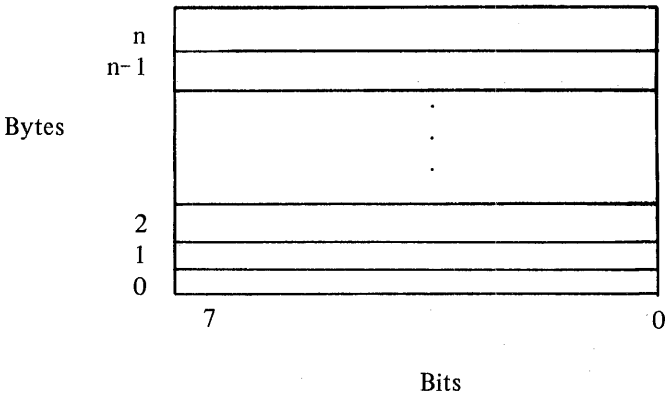
The effect of the # operator is to shift block number 35 to the right causing the following displacements and conversion:

```
01   110  111  111  111
 1     6    7    7    7
```

Appendix F serves as a conversion table for block-offset to octal notation.

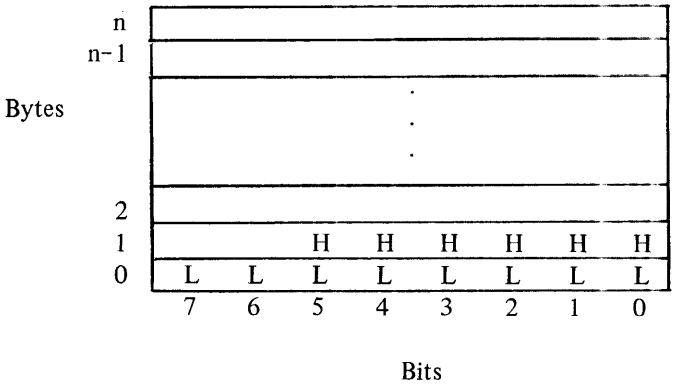
6.7 THE MEMORY MAP

The memory map of the M7341 module is relevant to the user's understanding of the instruction set described in Chapter 3. The map consists of a string of 8-bit bytes as shown:

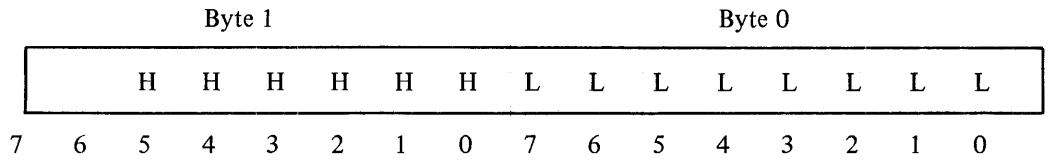


Bytes go from low (0) to high (n) in increments of 1. Within each byte, bits are numbered from right (0) to left (7). Bit 7 is the most significant bit.

Addresses in memory are 14 bits long and consist of one 8-bit byte plus the low six bits of the next byte, as shown:



Bit 5 of byte 1 above is therefore the most significant bit of the 14-bit address stored in bytes 0 and 1, as shown below:



6.8 ASSEMBLER SYMBOL TABLES

Symbols that appear in assembly language statements are stored in one of the following three symbol tables:

1. Pseudo-instruction symbol tables
2. Instruction symbol table
3. User symbol table

When searching for a symbol defined in the instruction field (or OP code field) of a statement, the search precedence is:

- Pseudo-instruction symbol table
- Instruction symbol table

When searching for a symbol defined in the operand field, the precedence is:

- User symbol table
- Instruction symbol table

If an operand of a three-byte ("type 2") instruction (e.g., CAL or JMP) is found in the instruction symbol table, this operand is assumed to be a two-byte address. The following warning message will be given

AW xxxx AT yy zzz

where AW identifies this as an address warning message, xxxx is the address (label) found in the instruction symbol table, and yy zzz is the current location counter at the time the label was processed, in block-offset notation. This message applies only to three-byte instruction entries and it is only a warning – the code is generated from the instruction symbol table.

The instruction symbol table is often called the OP Code Table, and the pseudo-instruction symbol table the Pseudo-OP Code Table.

6.9 MLA INSTRUCTION SET

The MPS instruction set is described fully in Chapter 3 in terms of components, number of bytes and time states, types of machine cycles executed, and examples of program usage. The following list summarizes classes of instructions:

- Index register instructions
- Accumulator (arithmetic/logical) instructions
- Program counter and stack control instructions
- Input/output instructions
- Machine instructions

Legal instructions in each of these categories are described in detail in Chapter 3 and are summarized in Appendix B.

6.10 PSEUDO-INSTRUCTIONS

The MLA pseudo-instructions documented in this section have been implemented to supplement the capabilities offered by the instruction set itself. Pseudo-instructions are referenced in the pseudo-instruction symbol table (also called the pseudo-OP table).

6.10.1 $: Indicating End of Program

The $ (dollar sign) pseudo-instruction is used to indicate the end of an assembly language program as follows:

Form $

The $ is a required part of every program and causes the current pass to be terminated. A carriage return must terminate the $ or the program will not execute.

The following is a legitimate end to an assembly language program:

```
        .
        .
        .
LMA             /STORE RESULT
HLT             /DONE
$
```

6.10.2 PAUSE: Pausing During Assembly

The PAUSE pseudo-instruction causes a pause in the Assembler processing and is issued as follows:

Form PAUSE

A carriage return must be typed after the PAUSE. The Assembler stops processing the paper tape being read at the time the PAUSE is encountered, but the current pass on the tape is not terminated. Processing continues when the user presses the CONTinue switch on the PDP-8 console.

PAUSE is normally used only at the end of a physical tape when the program being processed is stored on more than one tape. When the Assembler PAUSEs, it resets the input buffer pointer and waits for the operator to resume. He is expected to position the next tape segment of the program in the reader and, when the tape has been readied, to press the CONTinue switch.

6.10.3 *: Specifying an Origin

The * (asterisk) pseudo-instruction is used to specify the origin of the program and to set the initial program location counter as follows:

Form *expression

where expression is any legal Assembler expression as defined in previous paragraphs. The origin is assumed to be the value of expression and the location counter is set to this initial value. The value of the expression must be in the range block 0 offset 0 through block 77 offset 377 (octal 0 through 37777) respectively. If a value larger than the maximum value or smaller than zero is specified, the error message given in the following example is printed:

```
*177#000
AD AT 77 000
```

AD identifies this as an out-of-range address message, and 77 000 is the value of the current location counter. To avoid generating needless errors in the assembly, the out-of-range expression included in the origin statement will be truncated to 14 bits before the location counter is set. A symbolic origin can be specified, but any symbol must be defined before the origin pseudo-instruction is given. The following shows an example of such a specification:

```
START=04#200
*START
```

6.10.4 OCT, HEX, and DEC: Specifying Radix Control

Three pseudo-instructions have been implemented to allow the user to set the radix for numbers interpreted in the assembly language program. The octal base is assumed by default and must be explicitly overridden by one of the following pseudo-instructions:

Forms HEX
DEC

All numbers appearing after the new declaration are interpreted in the new base. To resume octal interpretation, the following specification must be given:

Form OCT

These pseudo-instructions set the radix as follows:

Pseudo-Instruction	Meaning
HEX	Set radix to base 16
DEC	Set radix to base 10
OCT	Set radix to base 8

6.10.5 EXPUNGE: Deleting the Instruction Symbol Table

The EXPUNGE pseudo-instruction is used to delete the entire instruction symbol table and is issued as follows:

Form EXPUNGE

EXPUNGE is used to give the assembly language programmer more core storage for his own user-defined symbols. It is recognized by the Assembler during pass 1 and is ignored during pass 2 and pass 3.

EXPUNGE deletes only the instruction symbol table and has no effect on the pseudo-instruction table. To define a new instruction symbol table, OPDEF must be invoked before any definitions are supplied. EXPUNGE must also be used before the space allocated for the user symbol table is used.

6.10.6 OPDEF: Specifying User-Defined Instructions

The OPDEF pseudo-instruction allows the assembly language programmer to define his own instructions with the following format:

Form OPDEF mnemonic;value;type

The mnemonic represents the user-defined operation and value is the value generated by that operation. The type represents the type of instruction generated and must be one of the following:

Type	Meaning
0	One-byte instruction
1	Two-byte instruction
2	Three-byte instruction

Type 0 instructions require no operands but both type 1 and type 2 instructions do require operands.

OPDEF must be issued before the space allocated for the user symbol table is used.

6.10.7 DATA: Assigning a Value to Storage

The DATA pseudo-instruction is used to assign one or more values to specific memory locations. It is included in a statement in the following way

Form label, DATA n0;n1;n2; . . .nm

where each n is a value, variable, or expression constructed as described in the paragraphs above. Entries are separated by semicolons (;). The numerical values of the expressions are assigned in sequence to memory locations beginning at the value of the current location counter.

For example, in the following example

```
*70#370
DATA  5;6;7
```

the current location counter is set at block 70, offset 370 (octal 34370) by the origin (*) pseudo-instruction. Values 5, 6, and 7 are assigned in sequence to memory locations 70 370, 70 371, and 70 372. The current location counter is then reset at 70 372. Inclusion of a label in the DATA statement is optional.

6.10.8 BLOCK: Assigning a Block of Data

The BLOCK pseudo-instruction is used to assign a block of core by placing in it a fixed value or by filling it with values with fixed increments or decrements. A BLOCK statement is constructed as follows

Form label, BLOCK size[; initial[; increment]]

where a block of the size specified is assigned values. If the following example is evaluated

```
*1#0
BLOCK  10;1;1
```

a block of size 10 (octal) is filled as follows with an initial value of 1 and a fixed increment of 1.

Address	Value
400	1
401	2
402	3
.	.
.	.
.	.
407	10
(octal)	

If a simple BLOCK 10 specification is supplied, the entire block will be filled with zeros. If BLOCK 10;4 is given, the entire block will be filled with 4's. A label can be included optionally in the BLOCK statement.

6.10.9 TEXT: Specifying a Character String

The TEXT pseudo-instruction is used to specify the inclusion of an ASCII character string in an assembly language program. It is issued as follows

Form label, TEXT▽literal▽. . .

where ▽ represents a delimiter and . . . indicates that the literal specification can be repeated. An example of a TEXT statement follows

TXOUT, TEXT /HI THERE/<215><0>

where HI THERE will be output followed by a carriage return (ASCII code 215) and a null (ASCII code 0).

Rules for the construction of literals in TEXT statements are summarized below:

1. The literal is delimited by a pair of any printing ASCII characters with the exception of a left angle bracket (<).

2. Right and left angle brackets (< and >) are used to enclose a numeric representation of an ASCII character. For example, <215> is used to represent a carriage return and might often be included after a text string to force a return before processing the next statement. ASCII codes are evaluated according to the current radix, set by OCT, DEC, or HEX.

3. There is no limit on the number of literals or ASCII representations that may appear in a TEXT statement, but the entire text string may not exceed the length of a line.

The TEXT statement may be labeled and commented, as shown in the following:

```
QUERY,   TEXT      /PRESS RETURN TO CONTINUE/<211><207>   /TAB AND RING BELL
```

In this example, a pair of slashes is used to delimit the text string itself and a space followed by a slash indicates the beginning of the comment field in the conventional way.

6.10.10 ADDR: Generating an Address

The ADDR pseudo-instruction is used to generate address constants in the following way

```
Form      label, ADDR a0;a1;a2; . . . ;am
```

where each a is an argument that generates a two-byte (low- and high-byte) address, and a's are separated by semicolons (;). The addresses are stored low-byte, high-byte, in sequence, in locations beginning at the current location counter. If a label is given, it refers to the low-byte of the first address.

The high two bits of an address are regarded as "don't care" bits by the Assembler and can be used as "flags."

6.11 ASSEMBLER OPERATING PROCEDURES

This paragraph summarizes Assembler inputs and outputs and describes procedures for loading and operating the Microprocessor Language Assembler.

6.11.1 Loading the Assembler into Core

MLA is supplied in the form of a paper tape, punched in binary-coded format. This tape is loaded into core by means of the Microprocessor Host Loader (MHL), using either the low-speed or high-speed paper-tape reader. Selection of the reading unit and other load procedures are shown in Figure 4-11.

6.11.2 Preparation of Input

Input to the Assembler consists of a source program punched in ASCII code on eight-channel paper tape. The tape can be prepared in one of two ways:

1. It can be punched by the user with an off-line Teletype (Model LT33), or
2. It can be punched by the Microprocessor Language Editor (MLE) (see Chapter 5).

In either case, the paper tape should begin with leader code which may be any of the following:

- Blank paper tape
- Code 200
- RUBOUT characters

The source program tape is read by the high-speed paper-tape reader or, if the high-speed device is not available, the low-speed reader associated with the Teletype. The input tape should be positioned in the appropriate reader after the Assembler itself is in core.

6.11.3 Starting the Assembler

The procedures outlined below should be followed in operating the Assembler.

1. Load the Assembler into core.
2. Set 0200 in the Switch Register and press ADDR LOAD.
3. Position the input tape in the paper-tape reader and turn on the appropriate reader and punch.
4. Set bits 0 and 1 of the Switch Register to indicate the pass and bits 2, 9, 10 and 11 as appropriate, to select output options (Tables 6-1 and 6-2).
5. Press CLEAR and CONTinue to begin pass 1.
6. The Assembler halts at the end of pass 1; set bits 0 and 1, position the source tape again, and press CONTinue to begin pass 2. Do the same for pass 3.

The Assembler will dynamically select the high-speed reader for input; if the high-speed unit is not available, MLA will select the low-speed reader.

Bits 0 and 1 of the Switch Register are set to indicate the current pass. The proper settings are listed in Table 6-1.

Table 6-1
Switch Register Settings

Pass	Bit 0	Bit 1
1	0	1
2	1	0
3	1	1

6.11.4 Assembler Output

Output from an assembly consists of a binary tape containing the object text punched by the Assembler and a listing of the source program and symbol table. These are produced as follows:

1. The symbol table is printed or punched by Assembler pass 1.

2. The object tape is punched by Assembler pass 2.

3. The listing and symbol table are printed or punched by Assembler pass 3.

The user has extensive control over the production of these outputs. By means of the Switch Register settings just mentioned, pass 2 (punching the binary tape) or pass 3 (listing the source text) can be entirely omitted. Switch Register bits 2, 9, 10 and 11 can also be set as shown in Table 6-2, to select or suppress certain Assembler outputs. The following can be controlled:

1. The listing of the symbol table can be produced or suppressed.

2. If produced, the symbol table can be punched on the high-speed device, printed on the line printer, or punched and listed on the Teletype.

3. The program listing can be produced or suppressed.

4. If produced, the program listing can be punched on the high-speed device, printed on the line printer, or listed on the Teletype.

5. The binary tape can be produced or suppressed.

6. If produced, the binary tape can be punched on the high-speed device or on the low-speed punch associated with the Teletype.

Combinations of bits can be set in such a way that any or all of these outputs can be produced on the desired media. Bit 2 can be set on to indicate suppression of the symbol table listing during passes 1 and 3. Bit 9 can be set during passes 1 and 3 to select the line printer for symbol table, program listing, or error message output. When bit 9 is set, bit 10 is automatically set on as well. Bit 10 is set during pass 3 to choose a particular tabbing convention for listing output. Bit 11 is used to select the appropriate device for symbol table, binary tape, and listing output. Table 6-2 summarizes all bit options.

If both bit 9 and bit 11 are set on during pass 3, bit 11 will take precedence and the program will be punched using the high-speed unit.

6.11.5 Symbol Table Format

During pass 1 the Assembler defines all user symbols and creates the symbol table. If the user chooses, this table is printed or punched at the end of pass 1 and repeated during pass 3. It is produced in alphabetical order showing both symbols and addresses at which they are referenced. If any symbols remain undefined, the US undefined symbol diagnostic is printed during the pass. Following is an example of the beginning of a symbol table listing:

```
A1      45  100
Q       45  177
WHAT    45  105
```

Table 6-2
Switch Register Options

Pass	Bit	Setting	Meaning
1	2	0	Output the symbol table.
		1	Suppress output of the symbol table.
	9	0	Do not send output to the line printer.
		1	Print the symbol table on the line printer.
	11	0	Print and punch the symbol table on the Teletype printer and the low-speed punch.
		1	Punch the symbol table on the high-speed punch (if available).
2	11	0	Punch the binary tape on the low-speed punch.
		1	Punch the binary tape on the high-speed punch (if available).
3	2	0	Output the symbol table.
		1	Suppress output of the symbol table.
	9	0	Do not send output to the line printer.
		1	Print the assembly listing on the line printer.
	10	0	Output TAB as TAB RUBOUT (codes 211 and 377).
		1	Output TAB (code 211) as eight-space TAB stops.
	11	0	Print the assembly listing on the terminal.
		1	Punch the assembly listing tape in ASCII on the high-speed punch.

6.11.6 Binary Output Format

A block of data punched on paper tape in absolute binary format has the following format:

FRAME	1	001	Start frame
	2	000	Null Frame
	3	xxx	Byte count (low eight bits)
	4	xxx	Byte count (high eight bits)
	5	yyy	Load address (low eight bits)
	6	yyy	Load address (high eight bits)
		.	
		.	Data placed here
		.	
	n	zzz	Last frame contains a block checksum

The binary output tape may contain one or more blocks of data. Each block has a positive integer byte count (frames 3 and 4) greater than six. The byte count is derived by counting the total number of bytes in the block excluding the checksum. The end-of-data block is signaled by a block with a byte count of exactly six. The loader will halt after loading tape in this format.

The maximum size of a block generated by the Assembler is 64 decimal (100 octal). Blocks are not padded out to an even length.

If a program contains an origin resetting as in the following

```
*1#200
INP2
  .
  .
  .
LMA
*2#300
  .
  .
  .
```

blocks will not be output for locations skipped between origin settings, and a new block will be started for each new origin setting.

A statement may generate none or several bytes of code, depending on its function and number of required operands. Blocks and offsets are in the range block 0 offset 0 through block 77 offset 377. Following is an example of part of an output listing:

```
                                                *45#100
45  100     004     A1,                         ADI Q       /WHAT?
            100
45  102     370                                 LMA
45  103     006                                 LAI A1%10
****IC      245     AT      45                  103
            103
45  105     104     WHAT,                       JMP A1
            100
            045
45  110     301                                 LAB A1
****OA              AT      45                  110
45  111     104                                 JMP A1+WHAT
            205
****AD              AT 45   111
            112                                 $
```

6.11.7 Output Listing Format

All output listings are produced in the format shown below:

Address		Code	Source
block number	offset	generated code	statement
		[generated code]	
		[generated code]	

6.12 ASSEMBLER DIAGNOSTIC MESSAGES

Errors or warnings encountered during assembly are output on the terminal or line printer; selection of the device depends on the setting of Switch Register bit 9.

Pass	Errors
1	Messages output on terminal or line printer, noting illegal values and addresses where errors occurred.
3	Listing of program output on terminal or line printer, with error messages following statements to which they apply.

The total number of errors encountered is always output on the terminal at the end of each pass regardless of the setting of bit 9. Following is an example of pass 1 output:

```
IC      245     AT  45  103
OA              AT  45  110
AD              AT  45  111

A1              AT  45  100
US      A110    AT  45  103
US      Q       AT  45  100
WHAT            AT  45  105

005     ERRORS
```

An example of pass 3 output is included in the previous section.

6.12.1 Error Types

Three types of error messages have been implemented for Assembler use and are described below. The syntax of an error message follows:

error code [symbol / ASCII representation] address

When an illegal symbol or unidentified address of some kind is found in an Assembler statement, a type 1 error message of the following kind is produced

```
US  Q   AT  45  100
```

where US indicates an undefined symbol message, Q is the address symbol which caused the error, and 45 100 is the location at which the error occurred, in block-offset notation. The listing of the statement which caused this error follows:

```
45  100   004     A1,     ADI Q
```

In type 2 error messages, the illegal symbol cannot be printed because the character that caused the error is not recognized by the Assembler. Thus the following might be produced

```
IC  245  AT  45  103
```

where IC indicates an illegal character message, 245 is the ASCII representation for % – the illegal character encountered but not recognized by the Assembler -- and 45 102 is the address at which the character was found. Following is the statement which produced this error:

```
45  102   006         LAI A1%10       /ADD A1 AND 10
          103
```

Type 3 error messages report on errors in which a symbol has not caused the error. These are errors in which an address is out of range or an operand is missing. For example, the statement:

```
45  110   301         LAB A1
```

causes an error because the LAB instruction does not take an operand. The following error message is produced

```
OA  AT  45  110
```

where OA indicates an operand error message and 45 110 is the address at which the error occurred.

6.12.2 Summary of Diagnostics

Table 6-3 summarizes all error messages that may be produced by the Assembler. These messages may be printed out during pass 1 and again during pass 3 to correspond to the statement listing produced during that pass. The type column identifies the message as a type 1, type 2, or type 3 diagnostic in accordance with the syntax specifications just given. In all of the messages, the address specification

```
AT  yy  zzz
```

indicates that the described error was encountered at block yy offset zzz.

Table 6-3
Assembler Diagnostics

Type	Message	Meaning
1	AW xxxx AT yy zzz	Address warning message; instruction xxxx found in the operand field as in JMP JMP.
1	DT xxxx AT yy zzz	Duplicate tag message; label xxxx (symbol ending with a comma) is repeated. Previous value of the symbol is retained, and the label is not redefined.
1	ID xxxx AT yy zzz	Illegal definition message; instruction or pseudo-instruction xxxx used illegally (EXPUNGE not given) as in A=LAB.
1	RD xxxx AT yy zzz	Redefinition by parameter assignment warning message; symbol xxxx is redefined (see DT for illegal definition of label fields).
1	US xxxx AT yy zzz	Undefined symbol message; xxxx is a symbol used in the program but not defined, as in US Q AT 45 100.
1	UO xxxx AT yy zzz	Undefined operation code message; xxxx is a symbol used in the instruction field of a statement but is not a legal instruction, as in A1, A2.
2	IC xxx AT yy zzz	Illegal character message; xxx is the ASCII representation of the illegal character encountered, as in IC 245 AT 45 102. The portion of the line following the illegal character is ignored.
3	AD AT yy zzz	Address out of range message. The illegal address was specified in an origin (*), a JMP, or in some other instruction, as in AD AT 45 111 for statement JMP A1+WHAT; when the error occurs in an origin, the current location counter is not set to the out-of-range value but to that value truncated to 14 bits.
3	IN AT yy zzz	Illegal numeric constant message; a specified number was unacceptable to the Assembler (e.g., 123K).
3	OA AT yy zzz	Operand error message; operand missing from a statement that requires one (e.g., JMP), or operand included in a statement that does not require one (e.g., LAB A1).
3	OV AT yy zzz	Input buffer overflow message; will happen only in the unlikely event that a simple input statement exceeds 126 characters in length.
3	ST AT yy zzz	Symbol table overflow message; will happen only if a large number of symbols defined (number of symbols allowed is approximately 200). Assembler halts and must be restarted.
3	PE AT yy zzz	Pseudo-instruction parameter error message; illegal parameter specified, as in OPDEF JUMP;104;3, where 3 is an illegal assignment for the instruction type.
2	PO xxx AT yy zzz	Pushdown list overflow message; xxx is the list location. This is a fatal error.
2	PU xxx AT yy zzz	Pushdown list underflow message; xxx is the list location. This is a fatal error.

CHAPTER 7
MICROPROCESSOR PROGRAM LOADER

This chapter summarizes the operation of the Microprocessor Program Loader (MPL) provided to MPS Monitor/Control Panel (MCP) users. Binary program tapes, normally produced by the Assembler described in Chapter 6, can be loaded into MPS memory by means of the Microprocessor Program Loader. The Loader is intended to be used extensively on the MCP for debugging and generation purposes.

7.1 OPERATING ENVIRONMENT

The Loader requires the following hardware to be available:

1. The Microprocessor Series Monitor/Control Panel (MCP)

2. A paper-tape reader (the low-speed device associated with the Teletype).

The paper-tape reader must be interfaced to the Processor Module.

7.2 LOADING A BINARY TAPE

Input to the Loader consists of a paper tape in MPS binary format usually produced as output from an MLA assembly on the PDP-8. To load the binary program into the module, follow the procedure outlined below:

1. Set the starting address on the MCP Switch Register to block 77 offset 0 (octal 37400).

2. Position the binary tape in the paper-tape reader and ready the device.

3. Press the ADDR LOAD and STRT keys on the MCP.

4. Turn the paper-tape reader on. Low-speed tape will be read until the process is stopped manually by pressing the STOP switch on the reader. High-speed tape will stop automatically.

A binary program can have blocks of almost any length (the maximum is a block of 2^{14} bytes. If binary output is generated in the normal way by the Assembler, 64-byte (decimal) blocks are output.

If the MPL load is successful, the Loader will halt at block 77 offset 141. If a checksum error occurs during the loading procedure, the load will not succeed and MPL will halt at block 77 offset 124 (octal 37524).

7.3 RESTARTING THE LOADER

It is possible to restart the loader after saving the contents of all but two of the module's registers. If the user presses ADDR LOAD and STRT at block 77 offset 200, the following registers will be saved:

Register	Block and Displacement	Octal Equivalent
A	76 340	37340
B	76 341	37341
C	76 342	37342
H	76 343	37343
L	76 344	37344

If the user presses ADDR LOAD and STRT at block 77 offset 202, the following registers will be saved:

Register	Block and Displacement	Octal Equivalent
A	76 340	37340
B	76 341	37341
C	76 342	37342
D	76 343	37343
E	76 344	37344

MPL will halt at block 77 offset 217.

7.4 MCP MEMORY

The Loader reserves two areas of Monitor/Control Panel (MCP) memory for its own use:

1. Random-Access Memory (RAM) (32 decimal words)

2. Read-Only Memory (ROM) (256 decimal words)

The RAM begins at block 76 and offsets 340 through 377 (octal 37340 through 37377). The ROM begins at block 77 and offsets 0 through 377 (octal 37400 through 37777). These addresses are used by the MCP and should not be accessed by the user.

CHAPTER 8
MICROPROCESSOR DEBUGGING PROGRAM

8.1 INTRODUCTION TO MDP

The Microprocessor Debugging Program (MDP) is a software tool which runs on the Processor Module and facilitates analysis and alteration of binary programs. These programs are normally produced by the Assembler (see Chapter 6).

MDP provides the following capabilities:

1. Reads and punches paper tape,
2. Opens specified memory locations for modification and allows the previous, current, and next locations to be opened, displayed, and closed,
3. Dumps the contents of program addresses, status flip-flops, and index registers on the Teletype printer,
4. Allows a program segment to execute for test purposes under MDP control,
5. Specifies a breakpoint location for program execution, and
6. Loads specified locations in memory with a constant value.

The major advantage of using a debugging package such as MDP is that the binary code itself can be examined and modified, allowing the program to be tested and corrected without requiring reassembly. This is an especially useful capability in the environment in which Microprocessor Series programs are typically developed.

8.2 OPERATING ENVIRONMENT

MDP is supplied in the form of a binary paper tape. To use the program, read the tape into memory using the standard Microprocessor Program Loader (MPL). After loading the tape set the starting address for MDP on the Monitor/Control Panel, raise the HLT switch, and press STRT. If the Teletype control knob is turned to LINE, MDP will respond by typing the MDP prompting character (*) on the Teletype printer. The starting address for MDP is block 0 offset 100 (octal 0100). All bits are zero except low bit 6.

The minimum memory requirement for running MDP is 1K of Random Access Memory (RAM) on a Microprocessor Series module. A Teletype must be interfaced with the M7341 module for MDP command input from the keyboard and the display of memory locations on the printer.

The low-speed paper-tape punch associated with the Teletype is used in conjunction with the R, P, T, and E commands to read the binary program tape into memory and to punch out a corrected version.

Input to MDP is usually a binary paper tape produced by the Assembler described in Chapter 6. The listing produced during pass 3 of the assembly is necessary for determining addresses to be examined and modified.

8.3 BASIC CHARACTER SET

The following list summarizes all ASCII characters that may be included in MDP command input and are recognized by the debugging program:

- Alphabetic characters B, D, E, G, L, P, R, S, T, X
- Numeric characters 0 through 7
- Selected special (printing) characters.

Character	Meaning
space	Space
#	Block-offset operator used to specify an address
;	Address separator
/	Used to open and display a location
.	Used to close and then reopen and display the current location
↑	Used to close current location and then open and display the previous location
RUBOUT	Used to delete digits back to a separator; echos backslash (\) followed by deleted digit
↑C	CTRL/C; used to abort current command

- Selected special (nonprinting) characters.

Character	Meaning
carriage return	Used to close current location or terminate a command
line feed	Used to close current location and then open and display the next location

If any character other than those just described is encountered by MDP, a question mark (?) is typed, the contents of the line containing the illegal character is ignored, and the command is aborted. The user can retype the command without typing a carriage return first.

8.4 ADDRESS SPECIFICATION

The format in which addresses are specified in MDP commands is the same format as that used in the binary program tape input to MDP. An address is a 14-bit field, described as follows:

hh lll
hh#lll

The two forms are interchangeable and represent the high six bits (hh) followed by the low eight bits (lll) of the address. For example, in address

35#200

35 represents the high bits or block, and 200 represents the low bits or offset within block 35. A detailed discussion of address specification is provided in Chapter 6 in descriptions of the Assembler block-offset operator # and the format for binary output.

Addresses are specified by the user in a great many MDP commands and the format may be either of those just shown. When output by MDP, as in the D command, an address specification is always of the form:

hh lll

If the user types too many digits when specifying an address, the results of such an error are unpredictable. It is recommended that the command be aborted by typing CTRL/C – the location can then be examined and modified if necessary.

Although leading zeros are never required in user specifications or addresses, MDP does supply the full complement of digits in its display, as follows:

```
*D 1#0;1#3
01 000/ 000
01 001/ 001
01 002/ 007
01 003/ 000
```

8.5 OVERVIEW OF MDP COMMANDS

After MDP has been started, an asterisk (*) output by the debugging program indicates that it is at monitor level and ready to accept a command. The user responds to this prompting character by entering a one-character command from the keyboard. If the specified command does not require parameters of any kind, MDP performs the operation at once without waiting for the user to end the command with a termination character (e.g., carriage return). Commands performed in this way include T, E, S, and X. MDP itself outputs necessary carriage return/line feed characters and types a new prompting character (*) after performing the specified operation. This capability implies that the user must be extremely careful to type the

correct characters. A paper tape read is performed as soon as the R command is typed, but MDP halts after reading tape and then waits to be restarted.

All other MDP commands require that parameters be included in the command line. After the user types one of the commands P, D, G, B, or L, MDP inserts a space after the one-character command and waits for necessary parameters to be typed by the user. The user must indicate that the command line is complete by typing a carriage return to terminate the command. MDP automatically inserts a line feed, performs the desired operation, and indicates a return of control to MDP monitor level by displaying an asterisk (*) on the Teletype printer. In the syntactic models shown in subsequent paragraphs, a carriage return/line feed combination, in which the user must supply the carriage return, is represented by <cr>. An explicit line feed character is represented by <lf>. Terminators output solely by MDP (as in S or T, for example) are not shown.

8.6 ERRORS IN SPECIFYING COMMANDS

A question mark (?) will be displayed on the Teletype printer if the user does any of the following:

1. Uses a character not in the basic MDP character set; for example, the following is an error:

   *D 1#0,

   because comma (,) is not a legal MDP character.

2. Specifies a nonexistent command, for example:

   *A

   A is not a valid MDP command.

3. Specifies an address with alphabetic or special characters or with illegal numeric characters; either of the following causes an error:

   *46#20A
   *1 379

After displaying the ? character, MDP ignores the current line and aborts the command. If the user omits parameters from a command line and types a carriage return to terminate this line, MDP will wait and will not return to monitor level until the parameters are typed or CTRL/C is issued.

8.7 SPECIAL FUNCTION KEYS

The following two paragraphs detail the operation of two special functions used to correct errors and to abort MDP activity.

8.7.1 RUBOUT: Deleting a Digit

The RUBOUT key on the Teletype keyboard is used for error correction when entering MDP parameters. Each RUBOUT causes the deletion of one digit, from right to left, beginning with the digit just to the left of the first RUBOUT and ending with the digit just to the right of the first separator encountered in the scan. Separators include the following:

Character	Meaning
space	Space output by MDP following a command or used as a block-offset operator
#	Block-offset operator used to separate the high and low bits of an address
;	Semicolon used to separate starting and ending address specifications
carriage return	Terminator typed after a command or used to close the current location in an examination command
line feed	Inserted by MDP after carriage return following a command or used to open the next location in an examination command
/	Open location or may be used as ;

RUBOUT echoes a backslash followed by the digit it deletes back to the most recent separator. Thus in the following command

*77#177\7\7\1277

the address originally specified

*77#177

is corrected and respecified as 77#277. The sequence \7\7\1 simply represents the digits 177 rubbed out, and 277 represents the new offset. RUBOUT of a digit causes a backslash, followed by the deleted digit, to be echoed on

the Teletype printer. If digits typed beyond the most recent separator must be deleted, the user must abort (Paragraph 8.7.2) and retype the entire command. An attempt to rub out digits beyond the separator causes zeros to be typed for these digits as in the following:

```
*72#123\3\2\1\0\0\0
```

In this command line, the user successfully rubbed out the digits 123; an attempt to delete the separator and block 72, however, failed, resulting in the printing of \0\0\0.

8.7.2 Control C: Aborting MDP Operation

A control C character can be issued at any time to return control to MDP monitor level. This function is useful to correct the entry of an invalid command or to terminate long input or output operations. It is recommended that this character be typed to abort a command when the user has made more than one or two errors when entering this command. Retyping the command is often a more straightforward and reliable method of correction than rubbing out and retyping multiple digits in an invalid line. To enter control C, type C while holding down the CTRL key. When CTRL/C is typed, the command being typed or executed is aborted, and the character is echoed as:

```
↑C
```

Control then returns to MDP, a new prompting character is output, and a new command is expected. The following is an example of the use of CTRL/C when terminating and address dump:

```
*D 45#100;47#377
45 100/ 000
45 101/ 000
45 102/ 377
45 ↑C
*
```

8.8 INPUT/OUTPUT COMMANDS

The user can read and punch binary paper tapes by means of the MDP commands listed in Table 8-1.

Table 8-1
Input/Output Commands

Command	Meaning
R	Read paper tape from low-speed paper-tape reader.
P	Punch paper tape for address range specified on low-speed punch.
T	Punch leader or trailer tape.
E	Punch end block and trailer.

These are described in more detail in Paragraph 8.8.1.

8.8.1 R: Reading Paper Tape

The R command is used to read the contents of a binary paper tape into memory. It is issued in the following way:

```
Form      R
```

This command does not require the user to type a carriage return. As soon as the character R is typed, the paper tape loaded in the low-speed tape reader associated with the Teletype is read into the memory addresses specified on the binary tape. This implies that the paper tape to be read must be properly positioned in the reader at the time the command

```
*R
```

is given.

The following sequence of steps should be used when loading paper tape into the reader:

1. Set the paper-tape reader switch to STOP or FREE.
2. Release the plastic cover of the reader unit and place the program tape over the read station with the small sprocket holes over the sprocket wheel. Close the cover.
3. Type R
4. Push the paper-tape reader switch to START and release.

After the entire program tape has been read into memory and an end block has been encountered, MDP halts. It can be restarted by setting the starting address (0 100) again, raising the HLT switch, and pressing STRT. To determine whether or not the read has been successful, the user should examine the accumulator (register A) after MDP has been restarted. If the contents of A are zero, reading has been successfully performed.

8.8.2 P: Punching Paper Tape

The P command facilitates the following operations:

- Punching selected program or other memory locations on paper tape.
- Punching an entire program tape by dumping the contents of part or all of memory.

All punching is performed on the low-speed paper-tape punch associated with the Teletype. The command is supplied as follows:

Form	P addr1;addr2<cr>
Where	addr1 is the starting memory location in block-offset notation addr2 is the ending memory location in block-offset notation
Example	*P 45#100;47#377

In this example, memory locations from block 45, offset 100 through block 47, and offset 377 are punched out on paper tape using the low-speed paper-tape punch. This command does not automatically punch leader tape and an end block so it should be used in conjunction with the T and E commands.

The addr2 parameter is not optional. If only one address is to be punched, the user must nevertheless supply starting and ending range specifications. In this case, both are identical, as in the following:

*P 11#300;11#300

The paper-tape punch must be readied at the time the command is issued. To position tape in the low-speed punch, do the following:

1. Turn the Teletype punch unit off.
2. Type the P command on the Teletype keyboard but do not type a carriage return.
3. Turn the punch unit on.
4. Type a carriage return to initiate punching.

It is important to follow this sequence to avoid punching command input on the program tape output by the punch.

It is a relatively easy matter to use MDP as a tool in generating paper tapes for use on the Microprocessor Series. Read a binary tape into memory and punch it out again using the following commands. This example assumes that program locations include blocks 15 through 17 and that all appropriate actions are taken to avoid punching unwanted characters on the output tape.

*R	Read paper tape into memory
*T	Punch header tape
*P 15#0;17#377	Punch blocks 15 through 17
*E	Punch end block and trailer tape
*	Return to MDP

8.8.3 T: Punching Leader and Trailer Tape

The T command uses the low-speed Teletype punch to produce either leader or trailer tape. Both leader and trailer tape have exactly the same format and consist of approximately four inches of tape punched with octal code 200. The command is issued as follows:

Form T

This command does not require that the user type a carriage return. As soon as the character T is typed, header or trailer tape is produced. MDP then inserts an automatic carriage return/line feed to return control to MDP monitor level.

If the punch is turned on at the time T is typed, the command character, as well as the carriage return/line feed inserted by MDP, will be output on the tape but ignored when the program is loaded. If the user wishes to exclude these extraneous characters from the program tape, he/she should follow certain procedures when producing header or trailer tape:

1. Turn the punch off.
2. Type the T command after the prompting character:

   *T

3. Turn the punch on immediately after typing the T command.

4. Turn the punch off after header or trailer tape has been produced.

Control returns automatically to MDP. Because the punch is not turned on until after T begins operation, a small amount of trailer tape might be lost.

8.8.4 E: Punching an End Block on Tape

The E command punches the end block, followed by approximately four inches of octal code 200 trailer tape, using the low-speed paper-tape punch. It is issued as follows:

Form E

This command does not require that the user type a carriage return. As soon as the character E is typed, end block and trailer tape are produced. MDP then inserts an automatic carriage return/line feed to return control to MDP monitor level.

An end block punched by MDP has the same format as that produced by the MLA Assembler. In this format, each block of data has a byte count of greater than six. The end block contains no data and therefore has a byte count of exactly six. The sequence of steps shown for the T command could be followed to prevent the E character from being punched out on paper tape. However, it is a more serious matter to lose part of the end block than to lose part of the leader/trailer tape. Therefore, it is usually preferable to leave the punch on while typing E and to rely on these command characters being ignored when the program tape is loaded.

Note that E implies automatic execution of the T command so trailer tape need not be explicitly requested.

8.9 LOCATION-EXAMINATION COMMANDS

MDP commands have been implemented to facilitate the examination and modification of memory locations. All commands in this category primarily consist of special Teletype keyboard characters as shown in Table 8-2.

8.9.1 /: Opening a Memory Location

The / command allows the user to specify that a particular memory location is to be opened and the contents of this location displayed. These contents can subsequently be changed. The command is issued in the following way:

Table 8-2
Location-Examination Commands

Command	Meaning
/	Opens specified location for modification
carriage return	Closes current location
line feed	Closes current location and opens next location
.	Closes current location and reopens it
↑	Closes current location and opens previous location

Form addr/

Where addr is the location to be examined in block-offset notation

Example *45#100/ 001

In response to the prompting character, the user types the address to be examined and follows it with a slash (/) character. MDP automatically inserts a space after the slash and prints out the contents of the examined location in three-digit octal form. The user can then modify the contents of the location by typing the new value to replace the value displayed as follows:

*45#100/ 001 111

The space between the old and new values is also output by MDP.

To terminate the command line, to return control to MDP, or to examine another location, the user types carriage return, line feed, period, or up-arrow. The different characteristics of these Teletype keys are presented in the following paragraphs.

8.9.2 Carriage Return: Closing an Open Location

In addition to its typical function as a statement terminator (for example, in B and P commands), the RETURN key can be used to close an open location which is being examined. A carriage return is typed at the end of the following command

```
*12#141/ 000 111<cr>
```

to indicate that the specified change in contents is to be made, and the location at block 12, offset 141 is to be closed. After the RETURN key has been pressed, control returns to MDP and the prompting asterisk is displayed. No further locations are opened until explicitly directed by another command.

8.9.3 Line Feed: Opening the Next Location

The line feed character can be typed instead of the carriage return to perform three distinct actions:

1. Close the location being examined.
2. Open the next location and display its contents.
3. Allow modification of the displayed location.

Use of the line feed in terminating the command

```
*27#0/ 377<lf>
```

causes the location at block 27 offset 0 to be closed and the location at block 27 offset 1 to be opened automatically. The full interaction looks like

```
*27#0/ 377<lf>
27 001/ 001
```

where the user types only the initial 27#0/ specification.

The long form of this function requires that the user issue two separate examination commands:

```
*27#0/ 377<cr>
*27#1/ 001
```

8.9.4 .: Reopening the Current Location

The period (.) is used to perform the following functions:

1. Close the location being examined.
2. Reopen the same location and display its contents.
3. Allow modification of the displayed location.

Use of the period is valuable when correcting an incorrectly altered location or when verifying that a change has been made. For example, in the following

```
*45#10/ 000 770\0\7\326\61\171.
45 010/ 271
```

the use of RUBOUT characters, echoing deleted characters, has made the modification of location 45#10 difficult to read. The period is used to verify that the desired correction has been made. Note that rubbing out 770 has indicated that 770 was truncated to 370, since MPS addresses can include offsets of only eight bits.

8.9.5 ↑: Opening the Previous Location

Use of the up-arrow (↑) character complements the use of the line feed. While line feed allows the user to view the next location, up-arrow causes the previous location to be opened. The following functions are performed:

1. Close the location being examined.
2. Open the previous location and display its contents.
3. Allow modification of the displayed location.

Use of ↑ in the following commands

```
*22#0/ 001↑
21 377/ 377 000↑
21 376/ 001
```

allows the user to view the contents of the location before 22#0, 21#377 (377) and to modify that location. ↑ is used again to view the contents of location 21#376 (001).

8.10 DISPLAY COMMANDS

A variety of commands have been implemented to allow a range of addresses or certain key locations to be displayed on the Teletype printer. Some commands allow modification of the contents of these locations; others do not. Table 8-3 lists the commands in this category.

Table 8-3
Display Commands

Command	Meaning
D	Dump the contents of a range of specified addresses on the Teletype printer.
S	Display the contents of the condition flip-flops and allow modification.
X	Display the contents of the accumulator and allow modification.

These are more fully described in subsequent paragraphs.

8.10.1 D: Dumping Address Contents

The D (dump) command allows the user to obtain a listing on the Teletype printer of all or some of the binary program in memory. It is issued as follows:

Form D addr1;addr2<cr>

Where addr1 is the starting memory location in block-offset notation
addr2 is the ending memory location in block-offset notation

Example *D 1#0;1#377

The user terminates the command by typing a carriage return; MDP inserts a line feed and proceeds to type out the desired listing in the following format:

```
*D addr1;addr2<cr>
addr1/ contents
addra/ contents
addrb/ contents
addrc/ contents
addrd/ contents
   .
   .
   .
addr2/ contents
*
```

An example is included below:

```
*D 1#0;1#377
01 000/ 000
01 001/ 000
01 002/ 001
01 003/ 002
01 004/ 077
   .
   .

01 377/000
*
```

If the user decides that he need not view the entire dump, the listing can be terminated by typing CTRL/C on the Teletype keyboard.

The addr2 parameter is not optional. If only one address is to be dumped, the user must nevertheless supply starting and ending range specifications. In this case both are identical, as in the following:

*D 36#0;36#0

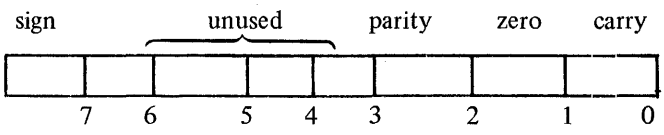
8.10.2 S: Displaying Status Flip-Flops

The S (status) command allows the user to examine and modify the contents of the status register which contains the following condition flip-flops:

Bit	Meaning
7	Sign
2	Parity
1	Zero
0	Carry

The status register has an organization as shown below:

STATUS REGISTER



The unused bits are always considered to be zeroed. Sign, parity, zero, and carry are normally set to one or reset to zero depending on the results of instruction execution. MDP allows the user to set these bits explicitly by means of the S command.

If the sign bit is set to one and all other bits are zero, the status register has contents of 200, as shown below:

```
10 000 000
 2  0   0
```

If the parity, zero, and carry bits are set to one and the sign bit is zero, the status register will have contents of 007, as follows:

```
00 000 111
 0  0   7
```

If only the zero bit is set, the following will be the case:

```
00 000 010
 0  0   2
```

The S command is entered in the following way:

```
Form      S
```

This command does not require that the user type a carriage return. As soon as the character S is typed, MDP itself inserts a carriage return/line feed and produces status bits in the following form on the Teletype printer:

```
addr/ contents
```

An example follows:

```
*S
20 105/ 200
```

The address returned by MDP is the address in which the status register is found. In the example just shown, the sign bit is set and all other bits are zero. To modify status bits, simply type new contents as in the following:

```
*S
20 105/ 200 005
```

Here the space between the old and new contents is output by MDP. The user changes the contents so that the parity and carry bits are set.

8.10.3 X: Displaying an Index Register

The X (index) command is used to examine and modify the index registers. The accumulator (register A) is accessed in the following way:

```
Form      X
```

This command does not require that the user type a carriage return. As soon as the character X is typed, MDP itself inserts a carriage return/line feed and displays the contents of the accumulator in the following form on the Teletype printer:

```
addr/ contents
```

An example follows:

```
20 111/ 000
```

The address returned by MDP is the address of the accumulator storage location on the user's machine and is not necessarily the one just shown. The contents of this address can be modified by typing in a new value, as follows:

```
*X
20 111/ 000 377
```

To examine and modify subsequent registers (B, C, H, L), the user terminates this and succeeding lines with an explicit line feed. For example:

```
*X
20 111/ 000 377 <lf>
20 112/ 001 <lf>
20 113/ 000 <lf>
20 114/ 007 <lf>
20 115/ 377 000 <cr>
```

<lf> indicates a line feed entered by the user, <cr> a carriage return. This sequence allows the user to examine all index register storage locations.

8.11 CONTROL COMMANDS

Commands used to set breakpoints, begin test execution, and clear memory locations can be categorized as control commands. Table 8-4 summarizes the functions of these commands.

Table 8-4
Control Commands

Command	Meaning
G	Start execution of binary program at specified address.
B	Set breakpoint at specified address in binary program.
L	Set range of addresses to specified constant.

These commands are described in greater detail in the paragraphs that follow.

8.11.1 G: Executing the Program

The G (go) command is used to execute a binary program or part of the program. Often it is used in conjunction with the B (breakpoint) command to test part or all of the binary program read into memory. G is issued in the following way:

Form G addr<cr>

Where addr is the first address to be executed in block-offset notation

Example *G 10#121

The user types a carriage return to terminate the G command, and MDP inserts an automatic line feed. The status bits and registers saved when MDP was loaded are restored, and the program segment beginning at addr is executed. For test purposes, the user can set initial conditions before beginning program execution by modifying the status bits and index register storage locations as previously indicated.

8.11.2 B: Setting a Breakpoint

The B command provides one of the most useful features available through MDP. It is used to specify a location to be used as a breakpoint in the binary program currently in memory. When the program encounters this location during execution, it returns control to MDP. The B command is issued as follows:

Form B addr<cr>

Where addr is the address to be used as a breakpoint in block-offset notation

Example *B 37#0

The user types a carriage return to terminate the B command, and MDP inserts an automatic line feed.

The B command modifies the location specified and the two following locations; therefore, care must be used when placing breakpoints due to the variable length instructions. It specifies the address to be treated as a breakpoint location when the program is executed. Thus B is used in conjunction with G to test segments of a program. When the specified address is executed, the following actions occur:

1. The binary program stops.
2. Registers A, B, C, H, L, and the status bits are saved.
3. Control returns to MDP and the following is printed on the Teletype

   B
   *

   to indicate that the breakpoint has been reached.

The saved registers and status bits can now be examined and modified if necessary.

The breakpoint does not normally remain in the binary program; it is removed under any of the following circumstances:

1. The specified address is executed.
2. MDP is restarted.
3. A special version of the B command is entered:

The explicit B reset command has the following form:

Form B R

This command does not require that the user conclude with a carriage return. As soon as the B R combination is typed, MDP itself inserts a carriage return/line feed. The space between B and R is also output by MDP. After B R is typed, MDP returns to monitor level, removes the breakpoint from the binary program, and outputs a new prompting character.

8.11.3 L: Loading Memory with a Constant

The L command is used to load a segment of memory with a specified constant. It is issued in the following way:

Form L addr1;addr2;[constant]<cr>

Where addrl is the starting memory location in block-offset notation
addr2 is the ending memory location in block-offset notation
constant is optional and represents the value to be inserted in the memory locations

Examples *L 76#340;76#377;77<cr>
*L 76#343;76#362;<cr>

The user terminates the command with a carriage return and MDP inserts an automatic line feed. The addr2 parameter is not optional. If only one address is to be cleared, the user must nevertheless supply starting and ending range specifications. In this case, both are identical, as in the following:

*L 1#0;1#0;

L is often used to clear memory locations; if constant is omitted from the command, zero is the default, and the memory range specified is zeroed. The semicolon following addr2 must be supplied, even if constant is omitted from the command. After the operation defined in L is performed, MDP returns to monitor level and outputs a new prompting character.

CHAPTER 9
MICROPROCESSOR ROM PROGRAMMER

9.1 INTRODUCTION TO MPR

The Microprocessor Read-Only Memory (ROM) Programmer (MRP) is a PDP-8 system program used to read and write programmable read-only memory (PROM) circuits for use on the M7345 PROM Module. The write function performed by MRP can also be called programming a PROM. A special hardware assembly is required to implement the functions supported by MRP, since PROMs, the I/O medium used by this program, are not a standard device supported by a PDP-8 interface. The particular hardware environment in which MRP functions, is described in detail in the next paragraph.

MRP provides the following capabilities:

1. Reads and punches paper tape using the high-speed or low-speed unit
2. Reads, writes, and verifies PROMs
3. Opens specified memory locations for modification and allows the previous, current, and next locations to be opened, displayed, and closed
4. Dumps the contents of memory locations on the Teletype or line printer
5. Loads specified locations in memory with a constant value

MRP permits examination and modification of PROM locations read into memory, and provides a facility for obtaining both Teletype and line printer listings of programs stored in paper tape or PROM form.

9.2 HARDWARE ENVIRONMENT

This paragraph describes the characteristics and use of the special hardware used by MRP.

9.2.1 MR873 Hardware Assembly

Using the MRP on the PDP-8 requires a special hardware assembly to be placed on the machine's I/O bus. Only those PDP-8 models with OMNIBUS construction can support this assembly; thus MRP can run on PDP-8/E, PDP-8/F, and PDP-8/M models, but not normally on the PDP-8/I, PDP-8/L, or PDP-8/S.* The special hardware used by MRP consists of one basic unit, the MR873, which holds PROM circuits and is attached to the I/O bus as shown in Figure 9-1.

Any other devices supported by a particular PDP-8 configuration can also be conveniently added to the bus. Often a high-speed paper-tape reader/punch and a line printer are supported.

The MR873 consists of the following:

1. Main rack-mountable or bench top MR873 hardware assembly
2. One M1703 input interface module which plugs directly into the PDP-8 OMNIBUS
3. One M1705 output interface module which plugs directly into the PDP-8 OMNIBUS
4. Two Y168 zero-insertion force socket modules

When these components are unpacked at the user's installation they should be inspected for any obvious signs of damage that may have occurred during shipping.

*It is possible to use a special hardware interface to facilitate MRP use on the PDP-8/I, PDP-8/S, and PDP-8/L. Users of these CPU's should contact Logic Products Applications Engineering at Digital for information.

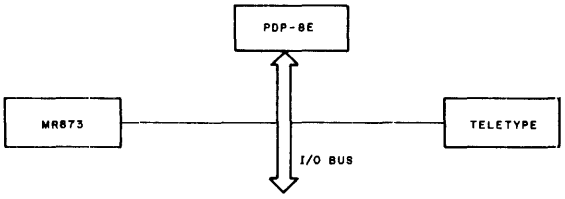


Figure 9-1 PDP-8 I/O Bus

Protruding from the rear of the MR873 unit are three flat gray cables and one standard power cord. When the BC08R-6 cables, used to connect the MR873 to the OMNIBUS, are unrolled the user will find a sticker attached to each cable at the plug end. This sticker identifies each cable and identifies its terminus as follows:

1. From J1 – MR873 to J1 M1703
2. From J2 – MR873 to J1 M1705
3. From J3 – MR873 to J2 M1705

The user should connect these cables in the way described, ensuring that the letters on the plug housing match the letters on the top of the board-mounted jack. Then the M1703 and M1705 modules can be installed in any convenient position on the OMNIBUS of the PDP-8/E, PDP-8/F, or PDP-8/M.

Next, the user should install the two Y168 modules that are designed to hold the PROMs. Plug one module into the slot labeled CHECK, VERIFY, FETCH, and the other into the slot labeled WRITE. After the line cord has been connected to any nearby 115 Vac 60-cycle outlet, the MR873 is ready for operation under MRP.

9.2.2 PROM Assembly and Manipulation

The Y168 modules are designed to hold PROM packages and to facilitate easy movement of these packages from one module socket to another. There are three sockets mounted on the front panel of the MR873 assembly labeled from left to right:

1. SIMULATE (RTM ONLY)
2. CHECK, FETCH, VERIFY
3. WRITE

These sockets have the following functions:

1. The socket labeled SIMULATE is not relevant to MRP and should be disregarded by the user.
2. The socket labeled CHECK, FETCH, VERIFY can be used only to read PROM data during check, fetch, or verify MRP operations.
3. The socket labeled WRITE can be used only to load PROM data during write MRP operations.

Only 1702A ultraviolet-erasable PROMs can be used with the MR873 and MRP. PROMs of this kind can be written in approximately two minutes.

WARNING

When PROMs are being written by the MR873, high voltage pulses (60 Vdc) are generated; the user should therefore be careful not to handle the PROM or MR873 assembly during this process. The voltage level generated can be dangerous and is present on both the PROM and the etch of the Y168 module into which the PROM is plugged while being written.

If it is ever necessary to remove or touch a PROM before writing has been completed, the user should stop the process either by pressing the HALT switch on the PDP-8 console or by typing CTRL/C on the Teletype keyboard.

Before a PROM is written, it should be completely erased by exposing it to ultraviolet light for five to ten minutes. The user should be sure to protect his eyes from the ultraviolet light.

PROMs must be inserted in the socket in the proper way, as shown in Figure 9-2.

If the user inserts the PROM into the module incorrectly, the chip may be destroyed. With the locking lever in the raised position, insert the PROM in the socket with the dot on the PROM in the position shown in Figure 9-3. Then lock it in by pushing the lever all the way down.

The following example illustrates a sequence in which a segment of the PDP-8 memory data buffer is loaded into a PROM and then verified. It is assumed that PDP-8 memory contains the desired data.

1. Install a clear PROM in the socket labeled CHECK, FETCH, VERIFY, and check that the PROM is clear by issuing a C (check) command from MRP.

2. Move the PROM to the socket labeled WRITE and load it with data by issuing a W (write) MRP command.

3. Move the PROM back to the socket labeled CHECK, FETCH, VERIFY and verify that the PROM was loaded correctly by issuing a V (verify) command from MRP.

Figure 9-2 MR873 ROM Programmer

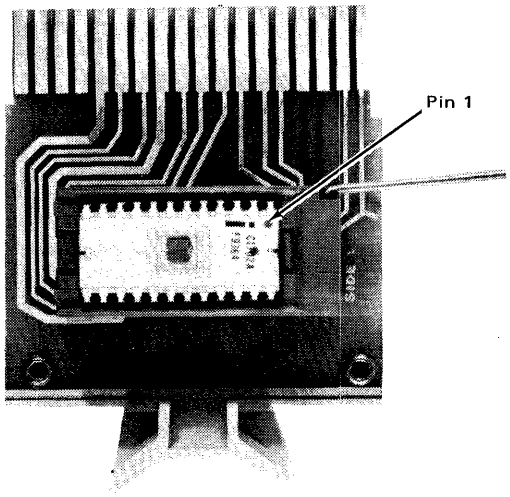


Figure 9-3 Y168 Socket Module (with PROM inserted)

Table 9-1 summarizes the correct location of the PROM during various MRP PROM I/O commands.

Table 9-1
Socket Positions for PROM Commands

Command	Socket Labeled WRITE	Socket Labeled CHECK FETCH VERIFY
WRITE	X	
CHECK		X
FETCH		X
VERIFY		X

To fetch data from a PROM, the user positions the PROM in the socket labeled CHECK, FETCH, VERIFY, and issues an F (fetch) command from MRP. Before performing this or any PROM operation, ensure that the PROM is securely locked in place with the socket lever.

9.3 OPERATING ENVIRONMENT

MRP is provided to users of this system in the form of a binary paper tape which is loaded into core by means of the Microprocessor Host Loader (MHL) using either the low-speed or high-speed paper-tape reader. Selection of the reading unit and other load procedures performed at this time are illustrated in Figure 4-11. To start the program, set the starting address 0200_8 in the Switch Register and press the ADDR LOAD and START keys on the PDP-8 console. If the Teletype control knob is turned to LINE, MRP will respond by typing the prompting character (*) on the Teletype printer.

The minimum memory and peripheral device requirements are the same as those described in Chapter 4. Input to MRP usually consists of a binary paper tape produced by the MLA Assembler and/or a previously programmed PROM. Output can be a punched binary tape, a programmed PROM, and/or a listing on the Teletype or line printer.

9.4 SWITCH REGISTER OPTIONS

Alternate output devices may be selected for use by MRP by setting the appropriate PDP-8 Switch Register bits before output is directed to these devices. Table 9-2 summarizes selection of the printing and punching units.

Bit 9 is set to indicate that the line printer is the primary output device. Error messages are always displayed on the terminal; if bit 9 is set they will also appear on the line printer, but they will never only appear on the line printer.

Table 9-2
Switch Register Options

Bit	Setting	Meaning
9	0	Print output on the Teletype printer.
	1	Print output on the line printer.
11	0	Punch tape on the low-speed paper-tape punch associated with the Teletype.
	1	Punch tape on the high-speed punch (if available).

Although output will be printed on the specified device if the setting of bit 9 is established before the command that produces output is typed, it is possible for output to be sent to both devices. If bit 9 is set 0 when the initial character of the MRP command is typed, the user can set the bit on after MRP outputs the space following that character. Then, when output is produced, it will appear on both the Teletype printer and the line printer. If the following command is typed

*D 21#0;21#27<cr>

and bit 9 is set after the MRP space and before <cr>, locations 21#0 through 21#27 will be dumped on both output devices.

If bit 9 is set but a line printer is not part of the PDP-8 configuration, the system will wait 150 milliseconds for the line printer to be attached and will then assume that the Teletype is the primary output device.

9.5 BASIC CHARACTER SET

The following list summarizes all ASCII characters that may be included in MRP command input and are recognized by the blasting program:

- Alphabetic characters C, D, E, F, L, P, Q, R, T, V, W
- Numeric characters 0 through 7
- Selected special (printing) characters, as follows:

Character	Meaning
space	Space
#	Block-offset operator used to specify an address
;	Address Separator
/	Used to open and display a location
.	Used to close and then reopen and display the current location
↑	Used to close current location and then open and display the previous location
RUBOUT	Used to delete digits back to a separator; echoes backslash (\) followed by deleted character
↑C	CTRL/C; used to abort current command

- Selected special (nonprinting) characters, as follows:

Character	Meaning
carriage return	Used to close current location or terminate a command
line feed	Used to close current location and then open and display the next location

If any character other than those just described is encountered by MRP, a question mark (?) is typed, the contents of the line containing the illegal character is ignored, and the command is aborted. The user can retype the command without typing a carriage return first.

9.6 ADDRESS SPECIFICATION

The format in which addresses are specified in MRP commands is the same format as that used in the Microprocessor Debugging Program (MDP) (Chapter 8) or in the binary program tape input to MRP. An address is a 14-bit field, described as follows:

```
hh lll
hh#lll
```

The two forms are interchangeable and represent the high six bits (hh) followed by the low eight bits (lll) of the address. For example, in address

```
23#0
```

23 represents the high bits or block, and 0 represents the low bits or offset within block 23. A detailed discussion of address specification is provided in Chapter 6 in descriptions of the Assembler block-offset operator # and the format for binary output.

Addresses are specified by the user in a great many MRP commands and the format may be either of those just shown. When output by MRP, as in the D command, an address specification is always of the form:

```
hh lll
```

If the user types too many digits when specifying an address, the results of such an error are unpredictable. It is recommended that the command be aborted by typing CTRL/C. The location can then be examined and modified if necessary.

Although leading zeros are never required in user specifications or addresses, MRP does supply the full complement of digits in its display as follows:

```
*D 1#0;1#3
01 000/ 000
01 001/ 001
01 002/ 007
01 003/ 000
```

9.7 OVERVIEW OF MRP COMMANDS

After MRP has been started, an asterisk (*) output by the program indicates that it is at monitor level and ready to accept a command. The user responds to this prompting character by entering a one-character command from the keyboard. If the specified command does not require parameters of any kind, MRP performs the operation at once without waiting for the user to end the command with a termination character (e.g., carriage return). Commands performed in this way include R, Q, T, E, and C. MRP itself outputs necessary carriage return/line feed characters and types a new prompting character (*) after performing the specified operation. This capability implies that the user must be extremely careful to type the correct characters. If an incorrect character is typed, the user can type CTRL/C to cancel the incorrect character. RUBOUT does not erase command characters – only digits in addresses.

All other MRP commands require that parameters be included in the command line. After the user types one of the commands P, F, W, V, D, or L, MRP inserts a space after the one-character command and waits for necessary parameters to be typed by the user. The user must indicate that the command line is complete by typing a carriage return to terminate the command. MRP automatically inserts a line feed, performs the desired operation, and indicates a return of control to MRP monitor level by displaying an asterisk on the Teletype printer. In the syntactic models shown in subsequent paragraphs, a carriage return/line feed combination, in which the user must supply the carriage return, is represented by <cr>. An explicit line feed character is represented by <lf>. Terminators output solely by MRP (as in R or T, for example) are not shown.

9.8 MRP ERRORS

There are two kinds of errors that are recognized by MRP. Command errors cause a question mark (?) followed by a carriage return/line feed to be displayed on the Teletype printer (and also on the line printer, if bit 9 is set). Execution errors cause a question mark followed by a message and a carriage return/line feed to be displayed.

Command errors occur when the user specifies a nonexistent command, as in the following:

```
*Z
```

A question mark and error message will be displayed when invalid characters are included in the command:

```
*F A
? ILLEGAL CHARACTER
```

Execution errors occur when an invalid or out-of-range address specification is included in a command, when a checksum error occurs during a read, or when an illegal address separator is specified. Error and warning messages are discussed in detail in the paragraphs on specific MRP commands which produce these messages.

9.9 SPECIAL FUNCTION KEYS

The following two paragraphs detail the operation of two special functions used to correct errors and to abort MRP activity.

9.9.1 RUBOUT: Deleting a Digit

The RUBOUT key on the Teletype keyboard is used for error correction in entering MRP parameters. Each RUBOUT causes the deletion of one digit, from right to left, beginning with the digit just to the left of the first RUBOUT and ending with the digit just to the right of the first separator encountered in the scan. Separators include the following:

Character	Meaning
space	Space output by MRP following a command, or used as a block-offset operator
#	Block-offset operator used to separate high and low bits of an address
;	Semicolon used to separate starting and ending address specifications
carriage return	Terminator typed after a command, or used to close the current location in an examination command
line feed	Inserted by MRP after carriage return following a command or used to open the next location in an examination command

RUBOUT echoes a backslash followed by the digit it deletes back to the most recent separator. Thus in the following command

```
*77#177\7\7\1277
```

the address originally specified

```
*77#177
```

is corrected and respecified as 77#277. The sequence \7\7\1 represents the digits 177 rubbed out and 277 represents the new offset. RUBOUT of a digit causes a backslash, followed by the deleted digit, to be echoed on the Teletype printer. If digits typed beyond the most recent separator must be deleted, the user must abort (Paragraph 9.9.2) and retype the entire command. An attempt to rub out digits beyond the separator causes zeros to be typed for these digits, as in the following.

```
*72#123\3\2\1\0\0\0
```

In this command line, the user successfully rubbed out the digits 123; an attempt to delete the separator and block 72, however, failed, resulting in the printing of \0\0\0.

9.9.2 Control C: Aborting MRP Operation

A control C character can be issued at any time to return control to MRP monitor level. This function is useful to correct the entry of an invalid command, or to terminate long input or output operations. It is recommended that this character be typed to abort a command when the user has made more than one or two errors when entering this command. Retyping the command is often a more straightforward and reliable method of correction than rubbing out and retyping multiple characters in a command line. To enter control C, type C while holding down the CTRL key. When CTRL/C is typed, the command being typed or executed is aborted, and the character is echoed as:

```
↑C
```

Control returns to MRP, a new prompting character is output, and a new command is expected. Following is an example of the use of CTRL/C in terminating an address dump:

```
*D 45#100;47#377
45 100/ 000
45 101/ 000
45 102/ 377
45 ↑C
*
```

There is one case in which CTRL/C cannot be used to terminate an I/O operation. If the low-speed paper-tape reader is in the process of reading a program tape, CTRL/C will not terminate the input operation. The computer must be halted and the program restarted to return control to MRP monitor level. Set 0200 in the Switch Register and press the ADDR LOAD and START keys to restart.

9.10 PAPER TAPE I/O COMMANDS

The user can read and punch binary paper tapes by means of the MRP commands listed in Table 9-3.

Table 9-3
Paper Tape I/O Commands

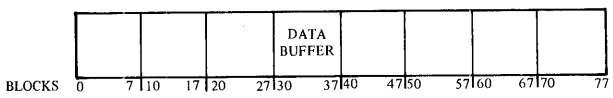
Command	Meaning
R	Read paper tape (up to capacity of data buffer) from high-speed or low-speed paper tape reader.
Q	Clear data buffer and continue to read paper tape from reader.
P	Punch paper tape from address range specified on high-speed or low-speed punch.
T	Punch leader or trailer tape.
E	Punch end block and trailer.

These are described in more detail in the following paragraphs.

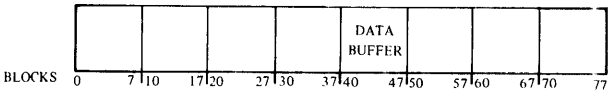
9.10.1 R: Reading Paper Tape

The R command is used to read a segment of binary paper tape into the data buffer. The capacity of this buffer at any time is only eight blocks or PROMs. Thus 2048 decimal or 4000 octal words can be loaded with a single read or queue (Paragraph 9.10.2) command. Any address in this data buffer can then be examined or modified with other MRP commands. When the next paper tape input command is processed, the data buffer is cleared before new data is loaded. A constant value or the contents of a PROM can be read into part of the buffer, however, overlaying or supplementing the current contents at any time.

The data buffer can be considered to be an eight-block window on the 64-block address space of the processor. The following illustrates this concept:



At this point, the data buffer consists of blocks 30 through 37. Only addresses in this eight-block space can be accessed with normal examination, modification, or display commands. As new data is queued from paper tape, however, the placement of the data buffer window will change, as follows:



Now only addresses in the range 40 through 47 can be examined and modified.

Paper tape can be read from either the high-speed or low-speed paper-tape reader. If the high-speed device is available, it will be selected automatically for use. The low-speed reader associated with the Teletype will be used if the high-speed device is not available.

The basic read command is issued as follows:

Form R

This command does not require the user to type a carriage return. As soon as the character R is typed, the paper tape loaded in the appropriate paper-tape reader is read into the data buffer. Note that MRP does not halt after the read is issued; thus the paper tape must be properly positioned in the reader at the time the command

*R

is given.

If the data buffer must be cleared of data previously read into it, the following interaction will take place:

```
*R
CLEARING THE DATA BUFFER
THE NEW DATA BUFFER RANGE IS FROM
     BLOCK 20 TO BLOCK 27
```

As the tape is read into blocks 20 through 27, the following messages will be output:

```
STARTING AT 20 000
LOADING BLOCK 20
LOADING BLOCK 21
LOADING BLOCK 22
LOADING BLOCK 23
LOADING BLOCK 24
LOADING BLOCK 25
LOADING BLOCK 26
LOADING BLOCK 27
STOPPED AT 27 377
```

If the tape end block is encountered before the end of the current data buffer, the following messages might be displayed:

```
*R
STARTING AT 20 000
LOADING BLOCK 20
LOADING BLOCK 21
STOPPED at 21 305
END-OF-DATA BLOCK WAS SEEN
```

If there is nothing in the segment of tape being read, the following will be displayed:

```
*R
STARTING AT 20 000
STOPPED AT 20 000
END-OF-DATA BLOCK WAS SEEN
```

If this occurs, nothing will be loaded into the data buffer.

If an error occurs while performing the read, the following message might be output:

```
*R
STARTING AT 20 000
LOADING BLOCK 20
? CHECKSUM ERROR
```

If the high-speed paper-tape reader is used, the following sequence of steps should be followed to position tape in the reader:

1. Turn the control knob to raise the tape retaining lever.
2. Place a fan-folded tape in the right-hand bin.
3. Place several folds of leader in the left-hand bin and position the tape so that the sprocket wheel engages the feed holes.
4. Turn the control knob to lower the tape retaining lever.
5. Press the FEED switch briefly to ensure that the tape is properly positioned.
6. Issue R command.

The sequence of steps below should be followed if the low-speed reader is selected:

1. Set the paper-tape reader switch to STOP or FREE.
2. Release the plastic cover of the reader unit and place the program tape over the read station with the small sprocket holes over the sprocket wheel. Close the cover.
3. Issue R command.
4. Push the paper-tape reader switch to START and release.

9.10.2 Q: Reading Additional Paper Tape

The Q (queue) command is used to clear the data buffer if it currently contains data and to read the next segment of paper tape. It is issued as follows:

Form Q

This command does not require that the user type a carriage return. As soon as the character Q is typed, the next segment of the paper tape loaded in the paper-tape reader used by the previous read command is read.

Q is used in conjunction with a paper tape whose addresses span more than eight consecutive blocks. It can also be used to reset the data buffer at any time. The R command is issued to read the first eight blocks of the new tape positioned in the high-speed or low-speed paper-tape reader. Q is used to clear the buffer of the data just read and to load the next address section(s).

If the data buffer must be cleared of data previously read into it, the following interaction might take place:

```
*Q
CLEARING THE DATA BUFFER
THE NEW DATA BUFFER IS FROM
      BLOCK 30 TO BLOCK 37
```

As the tape is read into blocks 30 through 37, the following messages will be output:

```
STARTING AT 30 000
LOADING BLOCK 30
LOADING BLOCK 31
LOADING BLOCK 32
LOADING BLOCK 33
LOADING BLOCK 34
LOADING BLOCK 35
LOADING BLOCK 36
LOADING BLOCK 37
STOPPED AT 37 377
```

If the tape end block is encountered before the end of the current data buffer, the following messages might be displayed:

```
*Q
CLEARING THE DATA BUFFER
THE NEW DATA BUFFER IS FROM
      BLOCK 30 to BLOCK 37
STARTING AT 30 000
LOADING BLOCK 30
LOADING BLOCK 31
LOADING BLOCK 32
STOPPED AT 32 377
END-OF-DATA BLOCK WAS SEEN
```

If an error occurs while performing the queue, the following message might be output:

```
*Q
CLEARING THE DATA BUFFER
THE NEW DATA BUFFER IS FROM
      BLOCK 30 TO BLOCK 37
STARTING AT 30 000
LOADING BLOCK 30
? CHECKSUM ERROR
```

If an R command has not preceded the Q or if the read encountered a checksum error or an end-of-data block, the following warning message will be displayed:

```
*Q
% ILLEGAL USE OF Q
% READ COMMAND ASSUMED
*R
```

After Q has been typed, enabling of the low-speed paper-tape reader can be performed at any time; MRP will wait until the reader is ready. The high-speed reader must be readied at the time the command is issued.

9.10.3 P: Punching Paper Tape

The P command facilitates the following operations:

1. Punching selected locations from paper tape, PROM, or the data buffer on paper tape.
2. Duplicating a paper tape by punching segments of the binary program stored in the data buffer.
3. Duplicating a PROM by writing out the contents of the data buffer.
4. Backing up a PROM on paper tape, or paper tape on one or more PROMs.

Punching is performed on either the high-speed or low-speed paper-tape punch, depending on the setting of Switch Register bit 11. If the bit is on, the high-speed punch is selected; otherwise, the low-speed device associated with the Teletype is used.

The P command is issued as follows:

Form	P addr1;addr2<cr>
Where	addr1 is the starting memory location in block-offset notation addr2 is the ending memory location in block-offset notation
Example	*P 45#100;47#377

In this example, memory locations from block 45, offset 100 through block 47, offset 377 are punched out on paper tape using the appropriate paper-tape punch. This command does not automatically punch leader tape and an end block so it should be used in conjunction with the T and E commands.

The addr2 parameter is not optional. If only one address is to be punched, the user must nevertheless supply starting and ending range specifications. In this case, both are identical, as in the following:

```
*P 11#300;11#300
```

There are several error messages which may be produced because of errors in address specifications. If an alphabetic or special character is supplied in an address of if an invalid address separator is typed, the following message will be displayed:

```
*P 30#0;37#37A
? ILLEGAL CHARACTER
```

If the starting block is greater than the ending block in the address specification, the following message will be displayed:

```
*P 7#0;0#377
? ADDRESS SPECIFICATION ERROR:
      BLOCK 1 > BLOCK 2
```

If the starting offset is greater than the ending offset in the address specification, the following message will be displayed:

```
*P 20#377;20#0
?LOW BYTES OF THE ADDRESS
      SPECIFICATION ARE
      REVERSED
```

This message only occurs when the starting and ending block specifications are the same.

The paper-tape punch must be readied at the time the command is issued. No special action need be taken to position paper tape for punching on the high-speed device. To ready tape for the low-speed punch, do the following:

1. Turn the Teletype punch unit off.
2. Type the P command on the Teletype keyboard but do not type a carriage return.
3. Turn the punch unit on.
4. Type carriage return to initiate punching.

It is important to follow this sequence in order to avoid punching command input on the program tape output by the punch.

It is a relatively easy matter to use MRP as a tool to facilitate high-speed or low-speed on-line tape duplication. Read a binary tape into memory and punch it out again using the following commands. This example assumes that program locations include blocks 15 through 17 and that all appropriate actions are taken to avoid punching unwanted characters on the output tape. Remember that the tapes will not be exact copies since MRP inserts a few special control characters.

```
*R              Read paper tape into memory
*T              Punch header tape
*P 15#0;17#377  Punch blocks 15 through 17
*E              Punch end block and trailer tape
*               Return to MRP
```

9.10.4 T: Punching Leader and Trailer Tape

The T command uses the high-speed or low-speed Teletype punch to produce either leader or trailer tape. Both leader and trailer tape have exactly the same format and consist of approximately four inches of tape punched with octal code 200. Selection of the punching unit depends on the setting of Switch Register bit 11. The command is issued as follows:

Form T

This command does not require the user to type a carriage return. As soon as the character T is typed, header or trailer tape is produced. MRP then inserts an automatic carriage return/line feed to return control to MRP monitor level.

If the punch is turned on at the time T is typed, the command character, as well as the carriage return/line feed inserted by MRP, will be output on the tape but ignored when the program is loaded. If the user wishes to exclude these extraneous characters from the program tape, he should follow certain procedures when producing header or trailer tape.

1. Turn the punch off.

2. Type the T command after the prompting character:

   *T

3. Turn the punch on immediately after typing the T command.

4. Turn the punch off after header or trailer tape has been produced.

Control returns automatically to MRP. Because the punch is not turned on until after T begins operation, a small amount of trailer tape might be lost.

9.10.5 E: Punching an End Block on Tape

The E command punches the end block, followed by approximately four inches of octal code 200 trailer tape, using the high-speed or low-speed paper-tape punch. Selection of the punching unit depends on the setting of Switch Register bit 11. It is issued as follows:

Form E

This command does not require the user to type a carriage return. As soon as the character E is typed, end block and trailer tape are produced. MRP then inserts an automatic carriage return/line feed to return control to MRP monitor level.

An end block punched by MRP has the same format as that produced by the MLA Assembler. In this format, each block of data has a byte count of greater than six. The end block contains no data and therefore has a byte count of exactly six. The sequence of steps shown for the T command could be followed to prevent the E character from being punched out on paper tape. However, it is a far more serious matter to lose part of the end block than to lose part ot the leader/trailer tape. It is therefore preferable to leave the punch on while typing E and to rely on these command characters being ignored when the program tape is loaded.

Note that E implies automatic execution of the T command, so trailer tape need not be explicitly requested.

9.11 PROM I/O COMMANDS

A variety of input/output commands have been implemented to allow the MRP user to read, check, write, and verify PROMs. These commands are listed in Table 9-4.

Table 9-4
PROM I/O Commands

Command	Meaning
F	Read or fetch contents of PROM in read socket, copying it into data buffer.
C	Check contents of PROM in read socket to ensure that all locations are clear.
W	Write specified address range onto PROM in write socket.
V	Verify that contents of PROM in write socket correspond to specified addresses in data buffer.

These are described in greater detail in the following paragraphs.

9.11.1 F: Reading a PROM

The F (fetch) command is used to read the contents of a PROM into the data buffer. The PROM must be in the read socket on the MR873 assembly. This command is issued as follows:

Form F addr1;addr2<cr>

Where addr1 is the starting memory location in block-offset notation
addr2 is the ending memory location in block-offset notation

Example *F 30#0;30#377

In this command, addr1 and addr2 reference memory locations in the data buffer in which the contents of the PROM will be loaded.

There are several error messages which may be produced because of errors in address specifications. If an alphabetic or special character is supplied in an address or if an invalid address separator is typed, the following message will be displayed:

```
*F 30#0;30#37%
? ILLEGAL CHARACTER
```

If the starting offset is greater than the ending offset in the address specification, the following message will be displayed:

```
*F 20#377;20#0
?LOW BYTES OF THE ADDRESS
      SPECIFICATION ARE
      REVERSED
```

Because the capacity of a PROM is only 256 decimal or 400 octal words, the address range cannot exceed one block and cannot cross a block boundary. All of the following are therefore invalid:

```
*F 1#0;7#377
*F 30#377;31#123
*F 47#200;50#177
```

If the block numbers of the starting and ending range specifications are not the same, the following will occur:

```
*F 1#0;7#377
?HIGH BYTES OF THE ADDRESS
      SPECIFICATION MUST BE
      THE SAME
```

Careful placement of PROM data is essential when copying from PROM to paper tape. The following illustrates concatenation of three PROMS onto paper tape. This example assumes that a new PROM is inserted in the read socket for each fetch.

*T	Punch header tape
*F 10#0;10#377	Fetch data from PROM and copy to data buffer
*F 11#0;11#377	Fetch data from PROM and copy to data buffer
*F 12#0;12#377	Fetch data from PROM and copy to data buffer
*L 13#0;17#377	Clear remaining locations in data buffer
*P 10#0;12#377	Punch blocks 10 through 12
*E	Punch end block and trailer tape
*	Return to MRP

The user types a carriage return to conclude the F command. MRP inserts an automatic line feed, performs the fetch, and returns to monitor level.

The fetch command is a very powerful one, since it can be used to redefine the current window on the data buffer. If the data buffer is defined as extending from 10#0 through 17#377 and the following command is issued

```
*F 37#16;37#377
```

MRP will display the following message:

```
CLEARING THE DATA BUFFER
THE NEW DATA BUFFER RANGE IS FROM
      BLOCK 30 TO BLOCK 37
```

The user has the option of electing not to redefine the data buffer at this time. If an error has been made or if valuable information is still in the current data buffer, the user can simply type CTRL/C at any time while the message is being displayed. After the message has been typed completely, the data buffer will be redefined as extending from 30#0 through 37#377.

9.11.2 C: Checking a PROM

The C (check) command examines every location of a PROM to ensure that the entire PROM is clear before the user attempts to write on it. The PROM must be in the read socket in the MR873 assembly at the time the command is given. The user types the following:

```
Form      C
```

It is not necessary to terminate this command with a carriage return. As soon as the character C is typed, MRP begins to examine PROM locations.

Each location is checked to ensure that it is clear. If the entire PROM is clear, the following message will be displayed:

```
*C PROM IS CLEAR
```

If any locations in the PROM have invalid contents, MRP will display both address and contents in a formatted list. Following is an example:

```
*C
ADRS PROM
003 001
127 010
322 072
376 077
377 177
*
```

Because a PROM consists of one complete block (400 octal words) of data, it is not necessary for MRP to supply the

block numbers of addresses with invalid contents. The three octal digits displayed beneath the ADRS label represent offsets within the block.

MRP automatically outputs a carriage return/line feed combination at the end of each line printed; it returns automatically to the monitor when all relevant addresses and contents have been printed.

9.11.3 W: Writing a PROM

The W (write) command is used to load (or program) a PROM with the contents of specified addresses in the data buffer. The PROM to be used for output must have been checked for clear contents and must be in the write socket on the MR873. The W command is issued as follows:

Form	W addr1;addr2<cr>
Where	addr1 is the starting memory location in block-offset notation addr2 is the ending memory location in block-offset notation
Example	*W 1#0;1#377

The user types a carriage return to conclude this command. MRP inserts an automatic line feed and displays the following message:

```
WAIT FOR BELL
```

The PROM will now be programmed. By watching the lights on the PDP-8 console panel, the user can determine when the PROM has been loaded. During the loading process, the MQ register displays the binary representation of all characters loaded. When the panel lights stabilize, the PROM has been loaded. At this time, MRP causes the Teletype bell to ring or the audible signal on another terminal to be produced. A carriage return/line feed is output and MRP returns to monitor level.

There are several error messages that may be produced because of errors in address specifications. If an alphabetic or special character is supplied in an address or if an invalid address separator is typed, the following message will be displayed:

```
*W
? ILLEGAL CHARACTER
```

If the starting offset is greater than the ending offset in the address specification, the following message will be displayed:

```
*W 20#377;20#0
?LOW BYTES OF THE ADDRESS
     SPECIFICATION ARE
     REVERSED
```

Because the capacity of a PROM is only one block, the address range cannot exceed one block and cannot cross a block boundary. The following are therefore invalid:

```
*W 20#0;27#377
*W 50#377;51#10
```

If the block numbers of the starting and ending range specifications are not the same, the following will occur:

```
*W 20#0;27#377
?HIGH BYTES OF THE ADDRESS
     SPECIFICATION MUST BE
     THE SAME
```

MRP can be used to duplicate PROMs in a straightforward on-line way. Read a PROM into the data buffer and write it out again using the following commands. This example assumes that program locations comprise block 36.

```
*C PROM IS CLEAR          Check that PROM is clear

*F 36#0;36#377            Fetch data from PROM
                          and copy it into data
                          buffer

*W 36#0;36#377            Write PROM from data
                          buffer

WAIT FOR BELL

*V 36#0;36#377            Verify contents of PROM
PROM VERIFIED OK
*                         Return to MRP
```

MRP can also be used to copy PROM-to-tape or tape-to-PROM. Remember, however, that the capacity of paper tape is much larger than that of a PROM, so care must be used when specifying addresses to be copied.

9.11.4 V: Verifying a PROM

The V (verify) command compares specified addresses in the data buffer with addresses of the PROM in the read socket on the MR873 assembly. It is issued in the following way:

Form	V addr1;addr2<cr>
Where	addr1 is the starting memory location in block-offset notation addr2 is the ending memory location in block-offset notation
Example	*V 74#100;74#277

The user types a carriage return to conclude this command. MRP inserts an automatic line feed, performs the verification operation, and returns to monitor level.

There are several error messages that may be produced because of errors in address specifications. If an alphabetic or special character is supplied in an address of if an invalid address separator is typed, the following message will be displayed:

```
*V 30#0;30#37A
? ILLEGAL CHARACTER
```

If the starting offset is greater than the ending offset in the address specification, the following message will be displayed:

```
*V 20#377;20#0
?LOW BYTES OF THE ADDRESS SPECIFICATION
      ARE REVERSED
```

This message only occurs when the starting and ending block specifications are the same.

Because the capacity of a PROM is only one block, the address range cannot exceed one block and cannot cross a block boundary. The following are therefore invalid:

```
*V 70#0;75#377
*V 60#377;61#2
```

If the block numbers of the starting and ending range specifications are not the same, the following will occur:

```
*V 20#0;27#377
?HIGH BYTES OF THE ADDRESS SPECIFICATION
      MUST BE THE SAME
```

If the specifications are outside the range of the current data buffer, the message shown below will be displayed. The current data buffer is assumed to include blocks 20 through 27.

```
*V 70#0;70#377
?HIGH BYTE OF THE ADDRESS SPECIFICATION
      OUTSIDE THE RANGE OF THE CURRENT
      DATA BUFFER
```

The contents of each PROM location is compared to the contents of the corresponding data buffer address. If the address contents are all the same, the following message will be displayed:

```
PROM VERIFIED OK
```

If any locations do not correspond, MRP will display both addresses and contents in a formatted list. Following is an example:

```
*V 36#0;36#377
ADRS BUF PROM
305 001 210
375 007 010
*
```

Because a PROM consists of one complete block of data, it is not necessary for MRP to supply the block numbers of addresses that cannot be verified. The three octal digits displayed beneath the ADRS label represent offsets within the block. The digits that appear beneath the BUF label are the contents of the specified offset in the data buffer; the digits beneath PROM are the contents of the PROM at that offset.

MRP automatically outputs a carriage return/line feed combination at the end of each line printed. It returns automatically to the montior level when all relevant addresses and contents have been printed.

9.12 LOCATION-EXAMINATION COMMANDS

MRP commands have been implemented to facilitate the examination and modification of memory locations. All commands in this category consist primarily of special Teletype keyboard characters as shown in Table 9-5.

Table 9-5
Location-Examination Commands

Command	Meaning
/	Opens specified location for modification
carriage return	Closes current location
line feed	Closes current location and opens next location
.	Closes current location and reopens it
↑	Closes current location and opens previous location

9.12.1 /: Opening a Memory Location

The / command allows the user to specify that a particular data buffer location is to be opened and the contents of this location displayed. These contents can subsequently be changed. The command is issued in the following way:

Form	addr/
Where	addr is the location to be examined in block-offset notation
Example	*45#100/ 001

In response to the prompting character, the user types the address to be examined and follows it with a slash (/) character. MRP automatically inserts a space after the slash and prints out the contents of the examined location in three-digit octal form. The user can then modify the contents of the location by typing the new value to replace the value displayed, as follows:

*45#100/ 001 111

The space between the old and new values is also output by MRP.

To terminate the command line, returning control to MRP or examining another location, carriage return, line feed, period, or up-arrow can be typed. The different characteristics of these Teletype keys are presented in the following paragraphs.

9.12.2 Carriage Return: Closing an Open Location

In addition to its typical function as a statement terminator (for example, in F and P commands), the RETURN key can be used to close an open location that is being examined. A carriage return is typed at the end of the following command

*12#141/ 000 111<cr>

to indicate that the specified change in contents is to be made, and the location at block 12, offset 141 is to be closed. After the RETURN key is pressed, control returns to MRP and the prompting asterisk is displayed. No further locations are opened until explicitly directed by another command.

9.12.3 Line Feed: Opening the Next Location

The line feed character instead of the carriage return can be typed to perform three distinct actions:

1. Close the location being examined.
2. Open the next location and display its contents.
3. Allow modification of the displayed location.

Use of the line feed in terminating the following command

*27#0/ 377<lf>

causes the location at block 27 offset 0 to be closed and the location at block 27 offset 1 to be opened automatically. The full interaction looks like

*27#0/ 377<lf>
27 001/ 001

where the user types only the initial 27#0/ specification.

The long form of this function requires that the user issue two separate examination commands, as follows:

```
*27#0/ 377<cr>
*27#1/ 001
```

9.12.4 .: Reopening the Current Location

The period (.) is used to perform the following functions:

1. Close the location being examined.
2. Reopen the same location and display its contents.
3. Allow modification of the displayed location.

Use of the period is valuable when correcting an incorrectly altered location or when verifying that a change has been made. For example, in the following:

```
*45#10/ 000 770\0\7\326\671.
45 010/ 271
```

the use of RUBOUT characters, echoing deleted characters, has made the modification of location 45#10 difficult to read. The period is used to verify that the desired correction has been made. Note that rubbing out 770 has indicated that 770 was truncated to 370, since MPS addresses can include offsets of only eight bits.

9.12.5 ↑: Opening the Previous Location

Use of the up-arrow (↑) character complements the use of line feed. While line feed allows the user to view the next location, up-arrow causes the previous location to be opened. The following functions are performed:

1. Close the location being examined.
2. Open the previous location and display its contents.
3. Allow modification of the displayed location.

Use of the ↑ in the following commands

```
*22#0/ 001↑
21 377/ 177 000↑
21 376/ 001
```

allows the user to view the contents of the location before 22#0, 21#377 (177) and to modify that location; ↑ is used again to view the contents of location 21#376 (001).

9.13 DISPLAY COMMAND

The D command has been implemented to allow the MRP user to obtain listings of part or all of the data buffer on the Teletype or the line printer.

9.13.1 D: Dumping Address Contents

The D (dump) command allows the user to obtain a listing on the Teletype or line printer of a range of memory addresses in the data buffer. It is issued as follows:

Form	D addr1;addr2<cr>
Where	addr1 is the starting memory location in block-offset notation addr2 is the ending memory location in block-offset notation
Example	*D 1#0;7#377

There are several error messages that may be produced because of errors in address specifications. If an alphabetic or special character is supplied in an address or if an invalid address separator is typed, the following message will be displayed:

```
*D 30#0;37#37N
? ILLEGAL CHARACTER
```

If the starting block is greater than the ending block in the address specification, the following message will be displayed:

```
*D 7#0;0#377
? ADDRESS SPECIFICATION ERROR:
      BLOCK 1 > BLOCK 2
```

If the starting offset is greater than the ending offset in the address specification, the following message will be displayed:

```
*D 20#377;20#0
?LOW BYTES OF THE ADDRESS SPECIFICATION
      ARE REVERSED
```

This message occurs only when the starting and ending block specifications are the same.

A dump command can access only the current data buffer. If the buffer is assumed to include blocks 0 through 7, the following are illegal specifications:

```
*D 1#0;36#377
*D 27#377;30#0
*D 70#0;70#377
```

The following interaction will take place:

```
*D 1#0;36#377
?HIGH BYTE OF ADDRESS SPECIFICATION
      IS OUTSIDE THE RANGE OF THE
      CURRENT DATA BUFFER
```

The user terminates the D command by typing a carriage return; MRP inserts a line feed and proceeds to type out the desired listing in the following format:

```
*D addr1; addr2<cr>
addr1/ contents
addra/ contents
addrb/ contents
addrc/ contents
addrd/ contents
      .
      .
      .
addr2/ contents
*
```

An example is included below:

```
*D 1#0;7#377
01 000/ 000
01 001/ 001
01 002/ 007
01 003/ 000
01 004/ 070
      .
      .
      .
07 377/ 000
*
```

If the user decides that he need not view the entire dump, or if the Teletype or line printer requires maintenance of any kind, the listing can be terminated by typing CTRL/C on the Teletype keyboard.

The addr2 parameter is not optional. If only one address is to be dumped the user must nevertheless supply starting and ending range specifications. In this case, both are identical, as in the following:

```
*D 36#0;36#0
```

9.14 CONTROL COMMAND

One MRP command has been implemented to allow the user to clear memory locations or to fill specified addresses with a constant.

9.14.1 L: Loading Memory with a Constant

The L command is used to load a segment of memory with a specified constant. It is issued in the following way:

Form	L addr1 ;addr2;[constant] <cr>
Where	addr1 is the starting memory location in block-offset notation addr2 is the ending memory location in block-offset notation constant is optional and represents the value to be inserted in the memory location
Examples	*L 76#340;76#352;7 *L 20#343;27#377;

The user terminates the command with a carriage return, and MRP inserts an automatic line feed. The addr2 parameter is not optional. If only one address is to be cleared the user must nevertheless supply starting and ending range specifications. In this case, both are identical, as in the following:

```
*L 70#0;70#0;
```

L is often used to clear memory locations; if constant is omitted from the command, zero is the default, and the memory range specified is zeroed. The semicolon following addr2 must be supplied, even if constant is omitted from the command.

There are several error messages that may be produced because of errors in address specifications. If an alphabetic or special character is supplied in an address or if an invalid address separator is typed, the following message will be displayed:

```
*L 30#0;37#37Q
? ILLEGAL CHARACTER
```

If the starting block is greater than the ending block in the address specification, the following message will be displayed:

```
*L 7#0;0#377;
? ADDRESS SPECIFICATION ERROR:
      BLOCK 1 > BLOCK 2
```

If the starting offset is greater than the ending offset in the address specification, the following message will be displayed:

```
*L 20#377;20#0;
?LOW BYTES OF THE ADDRESS SPECIFICATIONS
      ARE REVERSED
```

This message only occurs when the starting and ending block specifications are the same.

The load command can be used to redefine the current window on the data buffer. If the buffer is defined as extending from 20#0 through 27#377 and the following command is issued:

```
*L 72#0;73#377;
```

MRP will display the following message:

```
CLEARING THE DATA BUFFER
THE NEW DATA BUFFER IS FROM
      BLOCK 70 TO BLOCK 77
```

The user has the option of electing not to redefine the data buffer at this time. If an error has been made or if valuable information is still in the current data buffer, the user can simply type CTRL/C at any time while the message is being displayed. After the message has been typed completely, the data buffer will be redefined as extending from 70#0 through 77#377.

Although the data buffer can be redefined by specifying an address range in the new data buffer, a load cannot actually cross the boundary of a data buffer. Therefore, if the current data buffer extends from 20#0 through 27#377, the following is legal:

```
*L 70#0;77#377;
```

but the following example is illegal and results in the message displayed:

```
*L 26#0;32#377;
?HIGH BYTE OF ADDRESS SPECIFICATION
      IS OUTSIDE THE RANGE OF THE
      CURRENT DATA BUFFER
```

CHAPTER 10
SAMPLE PROGRAMS

This chapter contains a series of sample programs that might be useful as a reference when the user begins to construct programs based on the syntax described in Chapter 6. This sample code is heavily commented and is included in the form of assembly listings. Symbol table listings are included when indicated by the setting of the PDP-8 Switch Register.

10.1 LOADING REGISTER IN RAM

The following example illustrates suppression of symbol table output during assembly.

```
                        *77#200
                        RAM = 76#340
                    /OPERATION = LOAD ADDRESS OR JUMP TO 77#200
77 200   335     SAVEHL, LDH                 /PUT REGISTER H IN REGISTER D
77 201   346             LEL                 /PUT REGISTER L IN REGISTER E

                    /OPERATION = LOAD ADDRESS OR JUMP TO 77#202
77 202   066     SAVREG, LLI     RAM↑        /SAVE REGISTERS A–E
         340
77 204   056             LHI     RAM         /IN LOCATION 76#340 (37340)
         076
77 206   370             LMA                 /STORE A
77 207   060             INL                 /B
77 210   371             LMB
77 211   060             INL                 /C
77 212   372             LMC
77 213   060             INL                 /D
77 214   373             LMD
77 215   060             INL                 /AND E
77 216   374             LME
77 217   000             HLT                 /HALT
                        $
000 ERRORS
```

10.2 READING A BLOCK OF DATA

```
                                /READ IN A BLOCK OF DATA OF 10 (OCTAL)

                                        OPDEF     INPUT;INP;0  /DEFINE OPCODE INPUT
                                        OPDEF     CALL;106;2   /DEFINE OPCODE CALL
                                        START = 04#200
                                        *START
04 200    056                           LHI       BLK↑         /LOAD ADDRESS BLOCK TO H AND L REGS
          004
04 202    066                           LLI       BLK
          230
04 204    016                           LBI       10           /SET COUNT TO 10
          010
04 206    106           AGAIN,          CALL      READIN       /READ A BYTE OF DATA
          220
          004
04 211    370                           LMA                    /STORE IT IN BLOCK
04 212    060                           INL                    /INCREMENT POINTER IN BLOCK
04 213    011                           DCB                    /DECREMENT COUNTER
04 214    110                           JFZ       AGAIN        /IF NOT DONE, GO BACK FOR NEXT
          206
          004
04 217    000                           HLT

04 220    103           READIN,         INP1                   /INPUT STATUS
04 221    044                           NDI       40           /MASK "DA" (IGNORE ERRORS)
          040
04 223    150                           JTZ       READIN       /WAIT FOR "DA"
          220
          004
04 226    101                           INPUT                  /GET A BYTE OF DATA
04 227    007                           RET                    /RETURN

04 230    000           BLK,            BLOCK     10
          000
          000
          000
          000
          000
          000
          000
                                        $
AGAIN     04 206
BLK       04 230
READIN    04 220
START     04 200

000 ERRORS
```

10.3 CONVERSION/PRINT SUBROUTINES

The following subroutines are included as examples of MLA code. They must be assembled with other segments of a program.

```
                        /BINARY TO DECIMAL CONVERSION AND PRINT
                        /B AND C REGISTERS ARE USED FOR WORKING VARIABLES
                        /E REGISTER WILL HOLD FINAL DIGIT TO BE PRINTED
                        /CAL DCM TO PRINT INTEGER PORTION OF NUMBER
                        /CAL FRA TO PRINT FRACTIONAL PART
                        /CAL EITHER WITH DATA IN AC
                              *22#042
22 042   066    DCM,    LLI TENS
         062
22 044   026             LCI -3          /TYPE 3 PLACES BEFORE DECIMAL POINT
         375
22 046   104             JMP DCP
         055
         022
22 051   066    FRA,    LLI TENTH
         065
22 053   026             LCI -2          /TYPE 2 PLACES AFTER DECIMAL POINT
         376
22 055   310    DCP,    LBA             /SAVE BINARY IN B
22 056   046             LEI 60          /INIT E TO CHAR (0)
         060
22 060   104             JMP SKP
         064
         022
22 063   310    SVB,    LBA             /SAVE NEW B
22 064   250    SKP,    XRA             /CLEAR AC & CARRY
22 065   201             ADB             /ADD B TO AC
22 066   207    PTR,    ADM             /TRIAL SUBTRACT OF CONVERTER
22 067   100             JFC CNS         /SKIP IF CARRY NOT SET
         076
         022
22 072   040             INE
22 073   104             JMP SVB         /ELSE, BUMP DIGIT AND LOOP
         063
         022
22 076   106    CNS,    CAL TYP         /PRINT DIGIT
         361
         021
22 101   046             LEI 60          /RESET 'E'
         060
22 103   060             INL             /ADVANCE TO NEXT CONVERTER
22 104   020             INC             /ARE WE DONE?
22 105   110             JFZ SKP         /NO
         064
         022
22 110   007             RET             /YES, RETURN TO CALLING ROUTINE
```

```
                                    *24#062
24 062    234       TENS,     DATA        234;366;377 /-100,-10,-1
          366
          377
24 065    347       TENTH,    DATA        347;375 /-.1,-.01
          375

                                    *21#361
                    /TYPE A CHARACTER FROM REGISTER 'E'
                    /
21 361    103       TYP,      INP1        /READ THE 'UART' STATUS
21 362    044                 NDI 20      /TRANSMITTER BUFFER EMPTY?
          020
21 364    150                 JTZ TYP     /NO, WAIT
          361
          021
21 367    304                 LAE         /PRINT CHARACTER IN REGISTER 'E'
21 370    121                 OUT0
21 371    007                 RET
```

APPENDIX A
SUMMARY OF EDITOR (MLE) COMMANDS

Category	Command	Example	Function	Reference
READ	R	R	Read text from paper-tape reader and append it to text buffer.	5.6.1
APPEND	A	A	Read text from terminal and append it to text buffer.	5.6.2
INSERT	I	I	Insert text from terminal before line 1 in text buffer.	5.6.3
	nI	3I	Insert text from terminal before line n in text buffer.	5.6.3
LIST	L	L	List the contents of the text buffer on the terminal.	5.7.1
	nL	100L	List line n of the text buffer on the terminal.	5.7.1
	m,nL	1,50L	List lines m through n of the text buffer on the terminal.	5.7.1
PUNCH	P	P	Punch the contents of the text buffer on the high- or low-speed paper-tape punch (selection depends on setting of Switch Register bit 10).	5.7.2
	nP	6P	Punch line n of the text buffer on the paper-tape punch.	5.7.2
	m,nP	100,120	Punch lines m through n of the text buffer on the paper-tape punch.	5.7.2
FORM FEED	F	F	Punch a form feed (four blanks, a form feed character, and approximately two inches of blank tape) on the paper-tape punch.	5.7.3

Category	Command	Example	Function	Reference
TRAILER	T	T	Punch a trailer (approximately four inches of blank tape) on the paper-tape punch.	5.7.4
NEXT	N	N	Perform the functions of P, F, K, and R respectively.	5.7.5
	nN	10N	Perform the functions of P, F, K, and R respectively, n times in sequence.	5.7.6
CHANGE	nC	100C	Delete line n of the text buffer and replace it with the text entered from the terminal.	5.8.1
	m,nC	1,10C	Delete lines m through n of the text buffer and replace them with the text entered from the terminal.	5.8.1
DELETE	nD	42D	Delete line n from the text buffer.	5.8.2
	m,nD	12,20D	Delete lines m through n from the text buffer.	5.8.2
GET	G	G	Output the first tagged (labeled) line after the current location in the text buffer on the terminal.	5.8.3
	nG	15G	Output the first tagged line after line n in the text buffer on the terminal.	5.8.3
KILL	K	K	Kill (erase) the entire contents of the text buffer.	5.8.4
MOVE	m,n$jM	1,10$20M	Move lines m through n in the text buffer to the location just before line j.	5.8.4
SEARCH	S	S	Search the entire text buffer for all occurrences of the character entered from the terminal, but not echoed, after the carriage return.	5.8.6
	nS	10S	Search line n for occurrences of the character entered from the terminal and then allow command modification.	5.8.6
	m,nS	1,10S	Search lines m through n for occurrences of the character entered from the terminal and then allow command modification.	5.8.7

APPENDIX B
SUMMARY OF ASSEMBLER (MLA) INSTRUCTIONS

Instruction	Example	Function	Reference
Lr(1)r(2)	LAB	Load register 1 with the contents of register 2.	3.3.1
LrM	LDM	Load a register with the contents of memory.	3.3.1
LMr	LMA	Load memory with the contents of a register.	3.3.1
LrI	LAI A+B	Load a register with the byte of data immediately following the instruction.	3.3.2
LMI	LMI 104	Load memory with the byte of data immediately following the instruction.	3.3.2
Inr	INL	Increment a register.	3.3.3
DCr	DCB	Decrement a register.	3.3.4
ADr	ADD	Add the contents of a register to the accumulator.	3.4.1
ACr	ACB	Add the contents of a register and the carry flip-flop to the accumulator.	3.4.1
SUr	SUB	Subtract the contents of a register from the accumulator.	3.4.1
SBr	SBD	Subtract the contents of a register and the carry flip-flop from the accumulator.	3.4.1
NDr	NDB	Logical AND the contents of register with the accumulator.	3.4.1
XRr	XRA	Exclusively OR the contents of a register with the accumulator.	3.4.1
ORr	ORB	Inclusively OR the contents of a register with the accumulator.	3.4.1
CPr	CPB	Compare the contents of a register with the accumulator and set the status flip-flops.	3.4.1
ADM	ADM	Add the contents of memory to the accumulator.	3.4.2

Instruction	Example	Function	Reference
ACM	ACM	Add the contents of memory and the carry flip-flop to the accumulator.	3.4.2
SUM	SUM	Subtract the contents of memory from the accumulator.	3.4.2
SBM	SBM	Subtract the contents of memory and the carry flip-flop from the accumulator.	3.4.2
NDM	NDM	Logical AND the contents of memory with the accumulator.	3.4.2
XRM	XRM	Exclusively OR the contents of memory with the accumulator.	3.4.2
ORM	ORM	Inclusively OR the contents of memory with the accumulator.	3.4.2
CPM	CPM	Compare the contents of memory with the accumulator and set the status flip-flops.	3.4.2
ADI	ADI 2	Add the byte of data immediately following the instruction to the accumulator.	3.4.3
ACI	ACI 104	Add the byte of data immediately following the instruction and the carry flip-flop to the accumulator.	3.4.3
SUI	SUI 1	Subtract the byte of data immediately following the instruction from the accumulator.	3.4.3
SBI	SBI 6	Subtract the byte of data immediately following the instruction and the carry flip-flop from the accumulator.	3.4.3
NDI	NDI 100	Logical AND the byte of data immediately following the instruction with the accumulator.	3.4.3
XRI	XRI 340	Exclusively OR the byte of data immediately following the instruction with the accumulator.	3.4.3
ORI	ORI 102	Inclusively OR the byte of data immediately following the instruction with the accumulator.	3.4.3
CPI	CPI 4	Compare the byte of data immediately following the instruction with the accumulator and set the status flip-flops.	3.4.3
RLC	RLC	Rotate the contents of the accumulator one bit to the left and into the carry flip-flop.	3.4.4
RRC	RRC	Rotate the contents of the accumulator one bit to the right and into the carry flip-flop.	3.4.4

Instruction	Example	Function	Reference
RAL	RAL	Rotate the contents of the accumulator one bit to the left and through the carry flip-flop.	3.4.4
RAR	RAR	Rotate the contents of the accumulator one bit to the right and through the carry flip-flop.	3.4.4
JMP	JMP CKDONE	Jump unconditionally to the address specified in the instruction.	3.5.1
JFc	JFZ NXTBLK	Jump on a false flip-flop condition to the address specified in the instruction.	3.5.1
JTc	JTS ER	Jump on a true flip-flop condition to the address specified in the instruction.	3.5.1
CAL	CAL GETBYT	Call unconditionally the subroutine specified in the instruction.	3.5.2
CFc	CFZ ALL	Call on a false flip-flop condition the subroutine specified in the instruction.	3.5.2
CTc	CTP CKIT	Call on a true flip-flop condition the subroutine specified in the instruction.	3.5.2
RET	RET	Return unconditionally from a subroutine, popping the stack up one level.	3.5.3
RFc	RFZ	Return on a false flip-flop condition from a subroutine, popping the stack up one level.	3.5.3
RTc	RTS	Return on a true flip-flop condition from a subroutine, popping the stack up one level.	3.5.3
INP	INP	Read one byte of data from the input device into the accumulator.	3.6.1
OUT	OUT	Write one byte of data from the accumulator to an output device.	3.6.2
INP0	INP0	Read data from the UART.	3.6.3
INP1	INP1	Read status from the UART.	3.6.3
OUT0	OUT0	Output data to UART.	3.6.3
HLT	HLT	Halt the Assembler.	3.7.1
RST	RST	Restart the Assembler with a call to low memory.	3.7.2
ION	ION	Enable external events.	3.7.3
IOF	IOF	Disable external events.	3.7.3

APPENDIX C
SUMMARY OF
ASSEMBLER PSEUDO-INSTRUCTIONS

Pseudo-Instruction	Function	Reference
$	Signals end of assembly language program.	6.10.1
PAUSE	Causes pause in Assembler processing until CONTinue switch is pressed.	6.10.2
*expression	Specifies initial program location counter and can be used to reset current location counter.	6.10.3
OCT	Sets radix for subsequent numbers in program to octal (base 8).	6.10.4
HEX	Sets radix for subsequent numbers in program to hexadecimal (base 16).	6.10.4
DEC	Sets radix for subsequent numbers in program to decimal (base 10).	6.10.4
EXPUNGE	Deletes instruction symbol table.	6.10.5
OPDEF mnemonic;value;type	Allows programmer to define own instructions according to value and type given.	6.10.6
label, DATA n0;n1;n2;nm	Assigns values to incremental memory locations.	6.10.7
label, BLOCK size[;initial[;increment]]	Assigns a block of memory of the size given with values of zero, an initial value, or a set of increments.	6.10.8
label, TEXT▽ literal▽	Specifies ASCII character strings and/or numeric representations of ASCII characters to be included in a program.	6.10.9
label, ADDR a0;a1;a2;. . .;am	Assigns address constants to memory locations.	6.10.10

APPENDIX D
SUMMARY OF MICROPROCESSOR DEBUGGING PROGRAM (MDP) COMMANDS

Command	Example	Function	Reference
R	R	Read paper tape from low-speed reader.	8.8.1
P addr1;addr2	P 41#0;41#377	Punch out an address range on low-speed punch.	8.8.2
T	T	Produce leader or trailer tape (octal code 200) on low-speed punch.	8.8.3
E	E	Punch end block and trailer tape on low speed punch.	8.8.4
addr/	1#0/	Open specified location for examination or modification; specific line terminators may cause additional locations to be examined:	8.9.1
		<cr> close location.	8.9.2
		<lf> close location and open next one.	8.9.3
		. close location and reopen it.	8.9.4
		↑ close location and open previous one.	8.9.5
D addr1;addr2	D 1#0;1#377	Dump specified address range on Teletype printer.	8.10.1
S	S	Display and allow modification of status register contents.	8.10.2
X	X	Display and allow modification of index register contents.	8.10.3
G addr	G 10#121	Execute program to breakpoint location.	8.11.1

Command	Example	Function	Reference
B addr	B 37#0	Set program breakpoint at specified location.	8.11.2
L addr1;addr2;[constant]	L 76#340;76#377;7	Load a segment of memory with a specified constant.	8.11.3

APPENDIX E
SUMMARY OF MICROPROCESSOR ROM PROGRAMMER (MRP) COMMANDS

Command	Example	Function	Reference
R	R	Read paper tape from low-speed or high-speed reader.	9.10.1
Q	Q	Clear data buffer and continue to read paper tape.	9.10.2
P addr1;addr2	P 45#100;47#377	Punch out an address range on low-speed or high-speed punch.	9.10.3
T	T	Produce leader or trailer tape (octal code 200) on punch.	9.10.4
E	E	Punch end block and trailer tape on punch.	9.10.5
F addr1;addr2	F 30#0;30#377	Read contents of PROM into data buffer.	9.11.1
C	C	Check that each location of PROM is clear.	9.11.2
W addr1;addr2	W 1#0;1#377	Write (program) PROM from data buffer.	9.11.3
V addr1;addr2	V 74#100;74#377	Verify that addresses in data buffer and PROM are the same.	9.11.4
addr/	1#0/	Open specified location for examination or modification; specific line terminators may cause additional locations to be examined:	9.12.1
		<cr> close location.	9.12.2
		<lf> close location and open next one.	9.12.3
		. close location and reopen it.	9.12.4
		↑ close location and open previous one.	9.12.5

Command	Example	Function	Reference
D addr1;addr2	D 1#0;7#377	Dump specified address range on Teletype printer.	9.13.1
L addr1;addr2;[constant]	L 20*343;27#377;1	Load a segment of the data buffer with a specified constant.	9.13.1

APPENDIX F
BLOCK-OFFSET TO OCTAL CONVERSION

This appendix can be used if it is ever necessary to convert the block-offset notation in assembly language programs and output to octal notation. Only the first and last conversions are given for each block. To convert an offset within a given block, simply add the offset to the starting octal location in the block. For example:

Block	Offset	Octal
11	0	4400
.	.	.
.	.	.
.	.	.
11	377	4777

To convert block 11 offset 227 to octal, simply add 227 to 4400. The correct octal equivalent is thus 4627.

Block	Offset	Octal	Decimal	Block	Offset	Octal	Decimal
1	0	0400	0256	5	0	2400	1280
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
1	377	0777	0511	5	377	2777	1535
2	0	1000	0512	6	0	3000	1536
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
2	377	1377	0767	6	377	3377	1791
3	0	1400	0768	7	0	3400	1792
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
3	377	1777	1023	7	377	3777	2047
4	0	2000	0124	10	0	4000	2048
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
4	377	2377	1279	10	377	4377	2303

Block	Offset	Octal	Decimal	Block	Offset	Octal	Decimal
11	0	4400	2304	23	0	11400	4864
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
11	377	4777	2559	23	377	11777	5119
12	0	5000	2560	24	0	12000	5120
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
12	377	5377	2815	24	377	12377	5375
13	0	5400	2816	25	0	12400	5376
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
13	377	5777	3071	25	377	12777	5631
14	0	6000	3072	26	0	13000	5632
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
14	377	6377	3327	26	377	13377	5887
15	0	6400	3328	27	0	13400	5888
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
15	377	6777	3583	27	377	13777	6143
16	0	7000	3584	30	0	14000	6144
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
16	377	7377	3839	30	377	14377	6399
17	0	7400	3840	31	0	14400	6400
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
17	377	7777	4095	31	377	14777	6655
20	0	10000	4096	32	0	15000	6656
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
20	377	10377	4351	32	377	15377	6911
21	0	10400	4352	33	0	15400	6912
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
21	377	10777	4607	33	377	15777	7167
22	0	11000	4608	34	0	16000	7168
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
22	377	11377	4863	34	377	16377	7423

Block	Offset	Octal	Decimal	Block	Offset	Octal	Decimal
35	0	16400	7424	47	0	23400	9984
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
35	377	16777	7679	47	377	23777	10239
36	0	17000	7680	50	0	24000	10240
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
36	377	17377	7935	50	377	24377	10495
37	0	17400	7936	51	0	24400	10496
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
37	377	17777	8191	51	377	24777	10751
40	0	20000	8192	52	0	25000	10752
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
40	377	20377	8447	52	377	25377	11007
41	0	20400	8448	53	0	25400	11008
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
41	377	20777	8703	53	377	25777	11263
42	0	21000	8704	54	0	26000	11264
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
42	377	21377	8959	54	377	26377	11519
43	0	21400	8960	55	0	26400	11520
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
43	377	21777	9215	55	377	26777	11775
44	0	22000	9216	56	0	27000	11776
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
44	377	22377	9471	56	377	27377	12031
45	0	22400	9472	57	0	27400	12032
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
45	377	22777	9727	57	377	27777	12287
46	0	23000	9728	60	0	30000	12288
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
46	377	23377	9983	60	377	30377	12543

Block	Offset	Octal	Decimal	Block	Offset	Octal	Decimal
61	0	30400	12544	71	0	34400	14592
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
61	377	30777	12799	71	377	34777	14843
62	0	31000	12800	72	0	35000	14848
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
62	377	31377	13055	72	377	35377	15103
63	0	31400	13056	73	0	35400	15104
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
63	377	31777	13311	73	377	35777	15359
64	0	32000	13312	74	0	36000	15360
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
64	377	32377	13567	74	377	36377	15615
65	0	32400	13568	75	0	36400	15616
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
65	377	32777	13823	75	377	36777	15871
66	0	33000	13824	76	0	37000	15872
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
66	377	33377	14079	76	377	37377	16127
67	0	33400	14080	77	0	37400	16128
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
67	377	33777	14335	77	377	37777	16383
70	0	34000	14336				
.	.	.	.				
.	.	.	.				
.	.	.	.				
70	377	34377	14591				

APPENDIX G
7-BIT ASCII CODE

Octal Code	Char.	Octal Code	Char.	Octal Code	Char.	Octal Code	Char.
000	NUL	040	SP	100	@	140	`
001	SOH	041	!	101	A	141	a
002	STX	042	"	102	B	142	b
003	ETX	043	#	103	C	143	c
004	EOT	044	$	104	D	144	d
005	ENQ	045	%	105	E	145	e
006	ACK	046	&	106	F	146	f
007	BEL	047	'	107	G	147	g
010	BS	050	(	110	H	150	h
011	HT	051	)	111	I	151	i
012	LF	052	*	112	J	152	j
013	VT	053	+	113	K	153	k
014	FF	054	,	114	L	154	l
015	CR	055	-	115	M	155	m
016	SO	056	.	116	N	156	n
017	SI	057	/	117	O	157	o
020	DLE	060	0	120	P	160	p
021	DC1	061	1	121	Q	161	q
022	DC2	062	2	122	R	162	r
023	DC3	063	3	123	S	163	s
024	DC4	064	4	124	T	164	t
025	NAK	065	5	125	U	165	u
026	SYN	066	6	126	V	166	v
027	ETB	067	7	127	W	167	w
030	CAN	070	8	130	X	170	x
031	EM	071	9	131	Y	171	y
032	SUB	072	:	132	Z	172	z
033	ESC	073	;	133	[	173	{
034	FS	074	<	134	\	174	|
035	GS	075	=	135	]	175	}
036	RS	076	>	136	↑	176	~
037	US	077	?	137	←	177	DEL

digital

DIGITAL EQUIPMENT CORPORATION
WORLDWIDE SALES OFFICES

COMPONENTS GROUP HEADQUARTERS

ONE IRON WAY, MARLBOROUGH, MASSACHUSETTS 01752
(617) 481-7400 TWX 710-347-0348

For detailed information about products and policies, call 800-225-9480 toll-free (USA only). Massachusetts residents call (617) 481-7400

DOMESTIC

NORTHEAST

REGIONAL OFFICE:
235 Wyman Street, Waltham, Mass. 02154
Telephone: (617)-890-0330/0310 Dataphone: 617-890-3012 or 3013

CONNECTICUT
Meriden
240 Pomeroy Ave., Meriden, Conn. 06540
Telephone: (203)-237-8441/7466 Dataphone: 203-237-8205

Fairfield
1275 Post Road, Fairfield, Conn. 06430
Telephone: (203)-255-5991

NEW YORK
Rochester
130 Allens Creek Road, Rochester, New York
Telephone: (716)-461-1700 Dataphone: 716-244-1680

Syracuse
6700 Thompson Road, Syracuse, New York 13211
Telephone: (315)-437-1593/7085 Dataphone: 315-454-4152

MASSACHUSETTS
Marlborough
One Iron Way
Marlborough, Mass. 01752
Telephone: (617)-481-7400 Telex: 710-347-0348

MID-ATLANTIC

REGIONAL OFFICE:
U.S. Route 1, Princeton, New Jersey 08540
Telephone: (609)-452-2940

FLORIDA
Orlando
Suite 130, 7001 Lake Ellenor Drive, Orlando, Florida 32809
Telephone: (305)-851-4450 Dataphone: 305-859-2360

GEORGIA
Atlanta
2815 Clearview Place, Suite 100
Atlanta, Georgia 03040
Telephone: (404)-451-7411 Dataphone: 305-859-2360

NORTH CAROLINA
Durham/Chapel Hill
Executive Park
3700 Chapel Hill Blvd.
Durham, North Carolina 27707
Telephone: (919)-489-3347 Dataphone: 919-489-7832

NEW JERSEY
Fairfield
253 Passaic Ave., Fairfield, New Jersey 07006
Telephone: (201)-227-9280 Dataphone: 201-227-9280

Metuchen
95 Main Street, Metuchen, New Jersey 08840
Telephone: (201)-549-4100/2000 Dataphone: 201-548-0144

MID-ATLANTIC (cont.)

Princeton
U.S. Route 1, Princeton, New Jersey 08540
Telephone: (609)-452-2940 Dataphone: 609-452-2940

NEW YORK
Long Island
1 Huntington Quadrangle
Suite 1S07 Huntington Station, New York 11746
Telephone: (516)-694-4131, (212)-895-8095
Dataphone: 516-293-5693

Manhattan
810 7th Ave., 22nd Floor
New York, N.Y. 10019
Telephone: (212)-582-1300

PENNSYLVANIA
Philadelphia
Digital Hall
1740 Walton Road, Blue Bell, Pennsylvania 19422
Telephone: (215)-825-4200

TENNESSEE
Knoxville
6311 Kingston Pike, Suite 21E
Knoxville, Tennessee 37919
Telephone: (615)-588-6571 Dataphone: 615-584-0571

WASHINGTON D.C.
Lanham 30 Office Building
4900 Princess Garden Parkway, Lanham, Maryland
Telephone: (301)-459-7900 Dataphone: 301-459-7900 X53

CENTRAL

REGIONAL OFFICE:
1850 Frontage Road, Northbrook, Illinois 60062
Telephone: (312)-498-2500 Dataphone: 312-498-2500 Ex. 78

INDIANA
Indianapolis
21 Beachway Drive, Suite G
Indianapolis, Indiana 46224
Telephone: (317)-243-8341 Dataphone: 317-247-1212

ILLINOIS
Chicago
1850 Frontage Road
Northbrook, Illinois 60062 Dataphone: 312-498-2500

LOUISIANA
New Orleans
3100 Ridgelake Drive, Suite 108
Metairie, Louisiana 70002
Telephone: (504)-837-0257 Dataphone: 504-833-2800

CENTRAL (cont.)

MICHIGAN
Ann Arbor
230 Huron View Boulevard, Ann Arbor, Michigan 48103
Telephone: (313)-761-1150 Dataphone: 313-769-9883

Detroit
23777 Greenfield Road
Suite 189
Southfield, Michigan 48075 Dataphone: 313-557-3063

MINNESOTA
Minneapolis
8030 Cedar Ave. South, Minneapolis, Minnesota 55420
Telephone: (612)-854-6562-3-4-5 Dataphone: 612-854-1410

MISSOURI
Kansas City
12401 East 43rd Street, Independence, Missouri 64055
Telephone: (816)-252-2300 Dataphone: 816-461-3100

St. Louis
Suite 110, 115 Progress Parkway
Maryland Heights, Missouri 63043
Telephone: (314)-878-4310 Dataphone: 816-461-3100

OHIO
Cleveland
2500 Euclid Avenue, Euclid, Ohio 44117
Telephone: (216)-946-8484 Dataphone: 216-946-8477

Dayton
3101 Kettering Boulevard
Dayton, Ohio 45439
Telephone: (513)-294-3323 Dataphone: 513-298-4724

OKLAHOMA
Tulsa
3140 S. Winston
Winston Sq. Bldg., Suite 4, Tulsa, Oklahoma 74135
Telephone: (918)-749-4476 Dataphone: 918-749-2714

PENNSYLVANIA
Pittsburgh
400 Penn. Center Boulevard, Pittsburgh, Pennsylvania 15235
Telephone: (412)-243-9404 Dataphone: 412-824-9730

TEXAS
Dallas
Plaza North, Suite 513
2880 LBJ Freeway, Dallas, Texas 75234
Telephone: (214)-620-2051 Dataphone: 214-620-2061

HOUSTON
6656 Hornwood Drive
Monterey Park, Houston, Texas 77036
Telephone: (713)-777-3471 Dataphone: 713-777-1071

WISCONSIN
Milwaukee
8531 West Capitol Drive, Milwaukee, Wisconsin 53222
Telephone: (414)-463-9110 Dataphone: 414-463-9115

WEST

REGIONAL OFFICE:
310 Soquel Way, Sunnyvale, California 94086
Telephone: (408)-735-9200 Dataphone: 408-735-1820

ARIZONA
Phoenix
4358 East Broadway Road, Phoenix, Arizona 85040
Telephone: (602)-268-3488 Dataphone: 602-268-7371

CALIFORNIA
Santa Ana
2110 S. Anne Street, Santa Ana, California 92704
Telephone: (714)-979-2460 Dataphone: 714-979-7850

San Diego
6154 Mission Gorge Road
Suite 110, San Diego, California
Telephone: (714)-280-7880/7970 Dataphone: 714-280-7825

San Francisco
1400 Terra Bella, Mountain View, California 94040
Telephone: (415)-964-6200 Dataphone: 415-964-1436

Oakland
7850 Edgewater Drive, Oakland, California 94621
Telephone: (415)-635-5453/7830 Dataphone: 415-562-2160

West Los Angeles
1510 Cotner Avenue, Los Angeles, California 90025
Telephone: (213)-479-3791/4318 Dataphone: 213-478-5626

COLORADO
7801 E. Bellevue Avenue
Suite 5, Englewood, Colorado 80110
Telephone: (303)-770-8150 Dataphone: 303-770-8628

NEW MEXICO
Albuquerque
10200 Menual N.E., Albuquerque, New Mexico 87112
Telephone: (505)-296-5411/5428 Dataphone: 505-294-2330

OREGON
Portland
Suite 168
5319 S.W. Westgate Drive, Portland, Oregon 97221
Telephone: (503)-297-3761/3765

UTAH
Salt Lake City
429 Lawn Dale Drive, Salt Lake City, Utah 84115
Telephone: (801)-487-4669 Dataphone: 801-467-0535

WASHINGTON
Bellevue
13401 N.E. Bellevue, Redmond Road, Suite 111
Bellevue, Washington 98005
Telephone: (206)-545-4058/455-5404 Dataphone: 206-747-3754

INTERNATIONAL

EUROPEAN HEADQUARTERS

Digital Equipment Corporation International Europe
81 route de l'Aire
1211 Geneva 26, Switzerland
Telephone: 42 79 50 Telex: 22 683

FRANCE

Digital Equipment France
Centre Silic — Cidex L 225
94533 Rungis, France
Telephone: 687-23-33 Telex: 26840

GRENOBLE
Digital Equipment France
Tour Mangin
16 Rue Du Gal Mangin
38100 Grenoble, France
Telephone: (76)-87-56-01 Telex: 212-32682

GERMAN FEDERAL REPUBLIC

Digital Equipment GmbH
MUNICH
8 Muenchen 13, Wallensteinplatz 2
Telephone: 0811-35031 Telex: 524-226

COLOGNE
5 Koeln 41, Aachener Strasse 311
Telephone: 0221-44-40-95 Telex: 888-2269
Telegram: Flip Chip Koeln

FRANKFURT
6078 Neu-Isenburg 2
Am Forstaus Gravebruch 5-7
Telephone: 06102-5526 Telex: 41-76-82

HANNOVER
3 Hannover, Podbielskistrasse 102
Telephone: 0511-69-70-95 Telex: 922-952

STUTTGART
D-7301 Kemnat, Stuttgart
Marco-Polo-Strasse 1
Telephone: (0771)-45-50-65 Telex: 841-722-393

AUSTRIA

Digital Equipment Corporation Ges.m.b.H.
VIENNA
Mariahilferstrasse 136, 1150 Vienna 15, Austria
Telephone: 85 51 88

UNITED KINGDOM

Digital Equipment Co. Ltd.
U.K. HEADQUARTERS
Fountain House, Butts Centre
Reading RG1 7QN, England
Telephone: (0734)-583555 Telex: 8483278

BIRMINGHAM
Maney Buildings
29/31 Birmingham Rd., Sutton Coldfield
Warwickshire, England
Telephone: 021-355-5501 Telex: 337-060

BRISTOL
Fish Ponds Road, Fish Ponds
Bristol, England BS163HQ
Telephone: Bristol 651-431

EALING
Bilton House, Uxbridge Road, Ealing, London W.5.
Telephone: 01-579-2334 Telex: 22371

EDINBURGH
Shiel House, Craigshill, Livingston,
West Lothian, Scotland
Telephone: 32705 Telex: 727113

LONDON
Management House
43 Parker St., Holborn, London
WC 2B 5PT, England
Telephone: 01-405-2814/4067 Telex: 27560

MANCHESTER
Arndale House
Chester Road, Stretford, Manchester M32 9BH
Telephone: (061)-865-7011 Telex: 668666

UNITED KINGDOM (cont.)

READING
Fountain House, Butts Centre
Reading RG1 7QN, England
Telephone: (0734)-583555 Telex: 8483278

NETHERLANDS

Digital Equipment N.V.
THE HAGUE
Sir Winston Churchillian 370
Rijswijk/The Hague, Netherlands
Telephone: 94 9220 Telex: 32533

BELGIUM

Digital Equipment N.V./S.A.
BRUSSELS
108 Rue D'Arlon
1040 Brussels, Belgium
Telephone: 02-139258 Telex: 25297

SWEDEN

Digital Equipment AB
STOCKHOLM
Englundavagen 7, 171 41 Solna, Sweden
Telephone: 98 13 90 Telex: 170 50
Cable: Digital Stockholm

NORWAY

Digital Equipment Corp. A/S
OSLO
Trondheimsveien 47
Oslo 5, Norway
Telephone: 02/68 34 40 Telex: 19079 DEC N

DENMARK

Digital Equipment Aktiebolag
COPENHAGEN
Hellerupveg 66
2900 Hellerup, Denmark

FINLAND

Digital Equipment AB
HELSINKI
Tittismaantie 6
SF-00710 Helsinki 71
Telephone: (090) 370133
Cable: Digital Helsinki

SWITZERLAND

Digital Equipment Corporation S.A.
GENEVA
20, Quai Ernest Ansermet
Boite Postale 23, 1211 Geneva 8, Switzerland
Telephone No. 022/20 40 20 and 20 58 93 and 20 68 93
Telex: 28 92 01

ZURICH
Digital Equipment Corp. AG
Schaffhauserstr. 315
CH-8050 Zurich, Switzerland
Telephone: 01-46-41-91 Telex: 56059

ITALY

Digital Equipment S.p.A.
MILAN
Corso Garibaldi 49, 20121 Milano, Italy
Telephone: (02)-879-051/2/3/4/5 Telex: 843-33615

SPAIN

Digital Equipment Corporation Ltd.
MADRID
Atalo Ingenieros S.A., Enrique Larreta 12, Madrid 16
Telephone: 215 35 43 Telex: 27249

BARCELONA
Atalo Ingenieros S.A., Granduxer 76, Barcelona 6
Telephone: 221 44 66

ISRAEL

DEC Systems Computers Ltd.
TEL AVIV
Suite 103, Southern Habakuk Street
Tel Aviv, Israel
Telephone: (03) 443114/440763 Telex: 922-33-3163

CANADA

Digital Equipment of Canada, Ltd.
CANADIAN HEADQUARTERS
P.O. Box 11500
Ottawa, Ontario, Canada
K2H 8K8
Telephone: (613)-592-5111 TWX: 610-562-8732

TORONTO
2550 Goldenridge Road, Mississauga, Ontario
Telephone: (416)-270-9400 TWX: 610-492-7118

MONTREAL
9045 Cote De Liesse
Dorval, Quebec, Canada H9P 2M9
Telephone: (514)-636-9393 Telex: 610-422-4124

CALGARY/Edmonton
Suite 140, 6940 Fisher Road S.E.
Calgary, Alberta, Canada
Telephone: (403) 435-4881 TWX: 403-255-7408

VANCOUVER
Suite 202
644 S.W. Marine Dr., Vancouver
British Columbia, Canada V6P 5Y1
Telephone: (604)-325-3231 Telex: 610-929-2006

GENERAL INTERNATIONAL SALES

REGIONAL OFFICE
146 Main Street, Maynard, Massachusetts 01754
Telephone: (617) 897-5111
From Metropolitan Boston, 646-8600
TWX: 710-347-0217/0212
Cable: DIGITAL MAYN
Telex: 94-8457

AUSTRALIA

Digital Equipment Australia Pty. Ltd.
ADELAIDE
6 Montrose Avenue
Norwood, South Australia 5067
Telephone: (08)-42-1339 Telex: 790-82825

BRISBANE
133 Leichhardt Street
Spring Hill
Brisbane, Queensland, Australia 4000
Telephone: (072)-293088 Telex: 790-40616

CANBERRA
27 Collie St.
Fyshwick, A.C.T. 2609 Australia
Telephone: (062)-959073

MELBOURNE
60 Park Street, South Melbourne, Victoria 3205
Australia
Telephone: (03)-699-2888 Telex: 790-30700

PERTH
643 Murray Street
West Perth, Western Australia 6005
Telephone: (092)-21-4993 Telex: 790-92140

SYDNEY
P.O. Box 491, Crows Nest
N.S.W. Australia 2065
Telephone: (02)-439-2566 Telex: 790-20740

NEW ZEALAND

Digital Equipment Corporation Ltd.
AUCKLAND
Hilton House, 430 Queen Street, Box 2471
Auckland, New Zealand
Telephone: 75533

JAPAN

Digital Equipment Corporation International
Kowa Building No. 16 — Annex, First Floor
9-20 Akasaka 1-Chome
Minato-Ku, Tokyo 107, Japan
Telephone: 586-2771 Telex: J-26428
Rikei Trading Co., Ltd. (sales only)
Kozato-Kaikan Bldg.
No. 18-14 Nishishimbashi 1-Chome
Minato-Ku, Tokyo, Japan
Telephone: 5915246 Telex: 781-4208

PUERTO RICO

Digital Equipment Corporation De Puerto Rico
407 del Parque Street
Senturce, Puerto Rico 00912
Telephone: (809)-723-8068/67 Telex: 385-9056

ARGENTINA

BUENOS AIRES
Coasin S.A.
Virrey del Pino, 4071, Buenos Aires
Telephone: 52-3185 Telex: 012-2284

BRAZIL

RIO DE JANEIRO — GB
Ambriex S.A.
Rua Ceara, 104, 2 e 3 andares ZC - 29
Rio De Janeiro — GB
Telephone: 264-7406/0461/7625

SÃO PAULO
Ambriex S.A.
Rua Tupi, 535
Sao Paulo — SP
Telephone: 52-7806/1870, 51-0912

PORTO ALEGRE — RS
Rua Coronel Vicente 421/101
Porto Alegre — RS
Telephone: 24-7411

CHILE

SANTIAGO
Coasin Chile Ltda. (sales only)
Casilla 14588, Correo 15,
Telephone: 396713 Cable: COACHIL

INDIA

BOMBAY
Hinditron Computers Pvt. Ltd.
69/A, L. Jagmohandas Marg.
Bombay-6 (WB) India
Telephone: 38-1615: 36-5344 Telex: 011-2594 Plenty
Cable: TEKHIND

MEXICO

MEXICO CITY
Mexitek, S.A.
Eugenia 408 Deptos. 1
Apdo. Postal 12-1012
Mexico 12, D.F.
Telephone: (905) 536-09-10

PHILIPPINES

MANILA
Stanford Computer Corporation
P.O. Box 1808
416 Desmarinas St., Manila
Telephone: 49-68-96 Telex: 742-0352

VENEZUELA

CARACAS
Coasin, C.A.
Apartado 50939
Sabana Grande No. 1, Caracas 105
Telephone: 72-8662, 72-9637
Cable: INSTRUVEN

digital

COMPONENTS GROUP

DIGITAL EQUIPMENT CORPORATION, COMPONENTS GROUP, ONE IRON WAY, MARLBOROUGH, MASSACHUSETTS 01752
(617) 481-7400 TWX 710-347-0348

PRINTED IN U.S.A. 11174 531009/04/14 20