HP64000 Logic Development System

Emulator/ Internal Analysis 8-Bit Reference Manual

HEWLETT PACKARD

CERTIFICATION

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

WARRANTY

This Hewlett-Packard system product is warranted against defects in materials and workmanship for a period of 90 days from date of installation. During the warranty period, HP will, at its option, either repair or replace products which prove to be defective.

Warranty service of this product will be performed at Buyer's facility at no charge within HP service travel areas. Outside HP service travel areas, warranty service will be performed at Buyer's facility only upon HP's prior agreement and Buyer shall pay HP's round trip travel expenses. In all other cases, products must be returned to a service facility designated by HP.

For products returned to HP for warranty service. Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country.

HP warrants that its software and firmware designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

LIMITATION OF WARRANTY

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environmental specifications for the product, or improper site preparation or maintenance.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED. HP SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

EXCLUSIVE REMEDIES

THE REMEDIES PROVIDED HEREIN ARE BUYER'S SOLE AND EXCLUSIVE REMEDIES. HP SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER BASED ON CONTRACT, TORT, OR ANY OTHER LEGAL THEORY.

ASSISTANCE

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

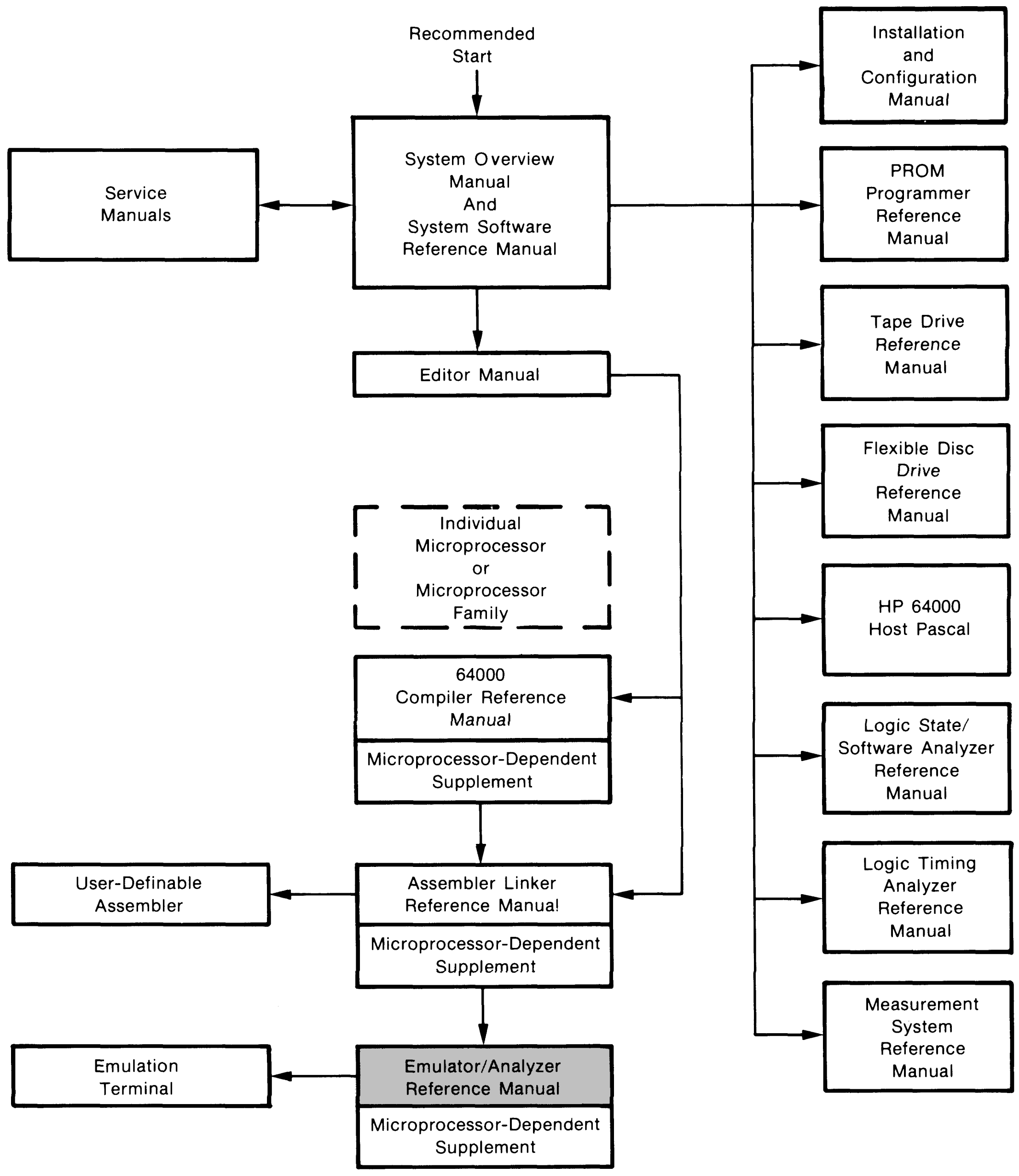
For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

CW&A 9/79

Model 64000 Reference Manuals

The following block diagram shows the documentation scheme for the HP Model 64000 Logic Development System. The interconnecting arrows show the recommended progression through the manuals as a way of gaining familiarity with the system.

Manual Map



Recommended Start
System Overview Manual And System Software Reference Manual
Service Manuals
Editor Manual
Individual Microprocessor or Microprocessor Family
64000 Compiler Reference Manual
Microprocessor-Dependent Supplement
User-Definable Assembler
Assembler Linker Reference Manual
Microprocessor-Dependent Supplement
Emulation Terminal
Emulator/Analyzer Reference Manual
Microprocessor-Dependent Supplement
Installation and Configuration Manual
PROM Programmer Reference Manual
Tape Drive Reference Manual
Flexible Disc Drive Reference Manual
HP 64000 Host Pascal
Logic State/ Software Analyzer Reference Manual
Logic Timing Analyzer Reference Manual
Measurement System Reference Manual

Printing History

Each new edition of this manual incorporates all material updated since the previous edition. Manual change sheets are issued between editions, allowing you to correct or insert information in the current edition.

The part number on the back cover changes only when each new edition is published. Minor corrections or additions may be made as the manual is reprinted between editions.

First Printing....... October 1980 (Part Number 64980-90991)
Reprinted ....February 1981
Second Edition .. September 1981 (Part Number 64980-90996)

Manual Conventions

The manual conventions and syntax conventions used in this book are presented below. For a full understanding of information in this manual, review the following conventions.

underlining

Where it is necessary to distinguish user input from computer output, the input is underlined.

Dashed line key symbols indicate a soft key on the keyboard. The physical labels for the soft keys appear on the CRT display. In text, the soft key label will appear within the symbol.

RETURN

Solid outlined key symbols are used in text to represent labeled keys on the keyboard.

[ ]

Parameters enclosed in square brackets are optional. Several parameters stacked inside a set of brackets indicate an either/or situation. You may select any one or none of the parameters.

The use of square brackets implies that a default value exists.

Example:

$$\begin{bmatrix}A\\B\end{bmatrix}$$

This indicates A or B may be selected.

{ }

Braces specify that the parameter enclosed is required information. When several parameters are stacked within a set of braces, you must select one and only one of the parameters.

Example:

$$\left\{\begin{matrix}A\\B\\C\end{matrix}\right\}$$

This example says one and only one of A, B, or C must be selected.

Choice of one and only one when several elements are enclosed.

Manual Conventions (Cont'd)

[ ] [ ]	Stacked square brackets indicate that enclosed parameters are optional and may be selected in any single occurrence, any combination, or may be omitted.
	Example: [A] [B] [C] A and/or B and/or C may be selected, or this option may be omitted.
< >	Angle brackets denote a syntactical variable. A syntactical variable is a defined parameter that you supply.
	Example: < FILE > This example says FILE is a variable that is input by the user.
lower-case bold type	Key words (soft key commands) are lower-case on the Model 64000. These key words will always be represented in text with **lower-case bold type**.
	Example: **edit** <FILE>
=>	Arrow indicates "is defined as."
. . .	An ellipses indicates a previous bracketed element can be repeated.

Manual Conventions (Cont'd)

UPPER-CASE PARAMETERS	Literal information which are parameters of a command, are represented in text with upper-case type. Literal information parameters are information that you enter as shown in text. An exception to this is any parameter enclosed with angle brackets < >, (e.g. <FILE> is a syntactical variable, not literal information).
Syntax symbols in color	Indicates symbols are used for definition purposes and do not appear on the CRT display.

Emulator/Internal Analysis 8-Bit Reference Manual

Table of Contents

Table of Contents (Cont'd)

Table of Contents (Cont'd)

Table of Contents (Cont'd)

Table of Contents (Cont'd)

Table of Contents (Cont'd)

List of Illustrations

List of Illustrations (Cont'd)

List of Illustrations (Cont'd)

List of Tables

Chapter 1 Introduction

The Impact of the Microprocessor

The impact of the microprocessor on the electronics industry, and on our society, can not be estimated. Not many years ago the electronics designer of digital equipment had to work at the gate level to incorporate many small scale integration packages to satisfy the requirements of his design. Today the designer's time is spent in an entirely different manner. Large scale integration, which has generated the microprocessor, now allows a designer to spend more time generating the function he wishes to create. Consequently, the digital designer is more of a systems engineer than ever before. The designer still has to worry about electrical interfacing and a minimized design, but the majority of time is spend analyzing ways to accomplish the function through the use of complex large scale integrated packages.

New Design Rules

Because of large scale integration, the development of new design has taken a different direction. Whereas, years ago, hardware was the only way to accomplish any particular function, today hardware really supports the sophisticated software package, which effects the desired result. Designs of today enjoy a higher cost to performance ratio than ever before. This is because the new design practices have resulted in a shift in the cost emphasis of development projects. In the past the major cost impact of development projects was primarily, if not totally, in the development of the hardware used to implement the function. Today, the costs have shifted from hardware to software. A rule of thumb often used is: the costs of software development usually exceeds the cost of hardware development by a factor of two. The consequence of this shift in cost is that a manager minimizes development costs by putting the money that was previously allocated to hardware development tools (oscilloscopes, logic analyzers, etc.) into the purchase of software development tools (microprocessor development systems). Now the designer is confronted with a new set of development tools for electronic equipment. The new tools are called assemblers, compilers, editors, and in-circuit emulators. Indeed, separation between the software designer and the hardware designer is becoming minimal.

It has become readily apparent to the manufacturers of microprocessors that new tool are needed to adequately utilize their products in new electronic designs. These manufacturers have introduced new development tools called microprocessor development systems. The original development systems were little more than software development aids. The proliferation of digital bus systems, prompted by the microprocessor, has generated the logic analyzer. One such logic analyzer was the HP5000A. Logic analyzers have allowed the digital designer to look at digital transactions on buses in complete byte time sequences. Because of the logic analyzer, digital design has become easier to cope with and the development cycle has been shortened.

New Development Systems

Microprocessor manufacturers have introduced a second generation of tools to support their microprocessor products. This generation of tools, the one we enjoy today, consists of development tools for both software and hardware. The software development tools allow the user to write text in machine language and edit text while resident on the computer. The user can take a file generated by the editor utility program and assemble the text into machine object code. When several software modules are to be combined in an operating system, a relocating linker attaches them together and automatically assigns addressing to each module. These operations are transparent to the user. Because the software developer needs to test software, even through the prototype hardware is not yet complete, the development system manufacturers have provided emulation systems. These emulation systems allow the user to run software without the target hardware. They also allow the hardware designer to run developed hardware under control of a device that will allow access to every memory or I/O transaction. Essentially, the emulation capability, with the added features of tracing, program single-stepping, or running, is the equivalent of a logic analyzer and a development system in one box.

Instrument manufacturers, and even small garage type shops, have begun to produce microprocessor development systems. The prices range from a few hundred dollars to several thousand dollars. In addition, well established instrument companies, such as Hewlett-Packard, have realized that the microprocessor revolution has brought new requirements in measurement and development tools. It has been also apparent that users of microprocessors wish to be free to develop products using more than one of the microprocessor architectures that are available on the market. This requires a development system that is universal; that is, able to support the development of more than one microprocessor. Clearly, microprocessor manufacturers are not inclined to encourage competition by supplying universal development systems. Consequently, the instrument companies have begun to build microprocessor development systems that are universal in nature and capable of handling more than one processor.

Disadvantages of Previous Development Systems

Previous microprocessor development systems, both universal and dedicated, have suffered disadvantages. Because the microprocessor development cycle was new and not completely understood, many functional implementations in the early development systems were difficult to use. In many ways they were reminiscent of computer technology ten to fifteen years ago. Many microprocessor developement systems, particularly those universal in nature, did not support processors as well as the manufacturer's systems. Development systems dedicated to a particular microprocessor supported that processor well; but universal systems have never supported any processor as well as the dedicated system built by its manufacturer. In many early development systems, if there was any emulation capability at all, it was usually scant. Companies developing electronic equipment have been inclined to buy one development system for the use of both hardware and software engineers. Often this has meant inadequate resources for everybody in the development lab. The development system has become a scheduled and time-shared resource resulting in inefficient development of microprocessor design and has caused both schedule and cost impacts to the development of new products. Until recently only machine or assembly level languages were available in development systems. For many applications, programming in machine language is inefficient and, consequently, wasteful of time and money. There was a time when no high level language such as Fortran or Pascal was available to the software designer. Consequently, programming in assembly level or machine level language caused software overruns and wasted time in software development.

Enter the Model 64000

With the Model 64000, Hewlett-Packard responded to these major disadvantages of microprocessor development systems. The result is a new development system that is easy to use and comprehensive in its capabilities. The 64000 enjoys the experience of a major logic analyzer manufacturer. Hewlett-Packard has been involved in the development of microprocessor-controlled instruments for several years and the 64000 was built by HP design engineers to satisfy HP requirements and to improve features found unsatisfactory on other development systems.

The Model 64000 is easy to use. Extensive use is made of the "soft key" concept. The soft keys allow easy access to the menu of commands and alternatives available at each step of the development process. For instance, the editor function is called by pressing the **edit** soft key, rather than typing the words EDIT or EDITOR. The microcomputer that controls the 64000 is a highly capable microprocessor built by Hewlett-Packard. As a result, the 64000 is capable of suporting most of the current 8- or 16-bit microprocessors with minimal change of hardware and software. The Model 64000 also has programming capability in Pascal. Pascal is a high level language that is gaining wide spread acceptance as a significant tool in the development of microprocessor software and firmware. The Model 64000 is also a multi-terminal system. A single disc system can be shared by up to six stations, thus minimizing utilization conflicts among software and hardware designers. Finally, Hewlett-Packard has used its experience as a logic analyzer manufacturer to develop an emulation and analysis capability previously unavailable. You will find that many of the emulation analysis features on the 64000 are similar to those available on the HP Logic Analyzers. The emulation and analyzer capabilities are easy to use in the Model 64000 and the system is capable of supporting a variety of emulators.

General Orientation of the Emulation and Analysis Manual

Chapter 2 of this manual briefly describes the the operation of the 64000. Chapter 3 covers the installation procedures of the emulation and analysis equipment. For information regarding installation of other pieces of equipment in the Model 64000, refer to the system manual. Chapter 4 covers emulator set up. Chapters 5 through 8 discuss emulation, analysis and simulated I/O commands.

Chapter 2
Theory of Operation

NOTE

Emulation and analysis equipment in the 64000 are optional. If the emulation and analysis options were ordered and not received, please contact the local Hewlett-Packard representative.

Introduction

This section briefly describes, the hardware and software that make up the emulation and analysis capabilities of 64000 Logic Development System.

What is emulation and analysis? Emulation is described as the "striving to be equal or to excel." In the case of microprocessor systems, to emulate means that the microprocessor and/or memory in a target system is replaced by another device which strives to look exactly like the original device being replaced. Advantage can be gained by building the tools necessary to control the execution of the replacement processor into the emulator. In controlling the operation of the microprocessor, the designer of microprocessor systems can effectively debug software or hardware problem areas.

Emulation is accomplished in the 64000 by a two-processor system. One processor is used by the 64000 operating system while the other one is used to emulate the desired microprocessor. Since the emulation processor and emulation memory are not shared with the operating system, the 64000 easily adapts to support different processors by changing the processor used for emulation. The same microprocessor which controls the operating system of the 64000 controls any processor module that is used for emulation.

An emulation and analysis module consists of four to five boards in a standard option package. Two levels of support are provided. The first level of support (called level one support), consists of four to five boards which are as follows:

- Emulation Control Board, Emulation Probe and Microprocessor Replacement Pod
- Analysis Board
- Memory Control Board
- Memory Board
- Memory Boards (Optional)

The emulation control board starts, stops and single-steps the emulating microprocessor. It controls the interaction between the 64000 operating system software and the emulation hardware. The control board is the major interface between the emulation pod and emulation bus. The emulation control board also allows the development system to interface to registers internal to the microprocessor chip.

The memory control board monitors the address and data buses to determine the type of memory that is to be accessed, (i.e., emulation memory or user memory). The memory control board is the major interface point from emulation memory to the 64000 operating system. In addition, the memory control board signals the analysis equipment and halts emulation if an access is attempted to PROM memory.

Each Memory Board contains enough sockets for 16K words (16 bits/word) of random access memory (RAM). A maximum of two Memory Boards may be used with 8-bit processors; up to four boards may be used with 16-bit processors. Options to these memory boards are available in which only a fourth of the board is loaded. It is possible, then, to buy emulation memory in units of 8K bytes. In an emulation system with the maximum emulation memory, there are two memory boards, each containing 32K bytes (16K words) of memory. The RAM in these boards can be configured as either random-access memory or read-only memory.

The analysis board in the Model 64000 is the equivalent of a logic analyzer. The analysis board accepts trigger specifications from the 64000 operating system software and then monitors the emulation bus to determine if the specified state has occurred. When the state occurs, the analysis board makes a trace of 256 contiguous events of program execution and stores them in a memory called trace memory or trace buffer, located on the analysis card. The trace data is available to the 64000 through the operating system and is displayed on the development station CRT.

An important point is that the emulation and analysis function is, in a sense, separate and independent from the rest of the 64000 system. That is, once the emulation and analysis equipment has been configured and set into operation, it can continue to operate without interference from the operating system until established conditions are met. In addition, the system software required to run the emulation and analysis capability does not take up any of the emulation RAM address space. Emulation memory may be configured to exist at any location in the address space of emulation processors.

You can use the CRT, keyboard, and printer as input and output devices for the emulated program. The input and output addresses of these devices are definable by the user. Note, however, that the actual interface to these devices is handled by the operating system of the 64000. Data ports, with assignable addresses, are available to the emulation processor as interfaces to the I/O devices. The actual transfer of data from these ports to the I/O devices is made under control of the the 64000 operating system software and is transparent to the emulation program.

When a linked and relocated file in the 64000 is loaded into the emulation system, a symbol table is generated with the object code. This symbol table allows you to specify addresses by specifying labels from the symbol table. For instance, it is possible to say "run from START" instead of "run from 0B00H". Symbols can be used to specify particular program memory locations but the mnemonics of memory contents will not include symbols from the symbol table in the disassembly.

Finally, it is important to note that the program code to be run under control of emulation and analysis must be object code that has been linked and relocated. Emulation and analysis will not execute source code or object code that has not been relocated.

Hardware Configuration

Emulation and analysis circuit boards are normally installed to the rear of the station cabinet. This is done to maximize the usable cable length to the emulation pod.

In general, the order of board insertion, from back to the front, is:

Emulation board
Analysis Board
Memory Control Board
Memory Board
Memory Board (Optional)

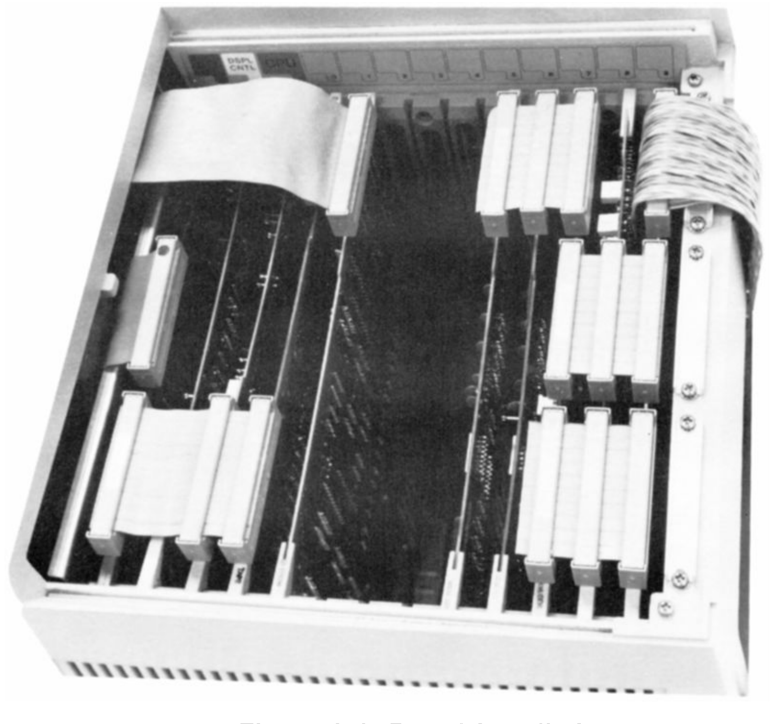


Figure 2-1. Board Installation

The emulation pod and socket normally extend from the emulation board. Refer to figure 2-1 for an illustration of the installed boards. For more information refer to Chapter 3 for the installation procedures.

Microprocessor Loading and Driving Capabilities

When installed in a target system, the emulation equipment will respond very nearly to the microprocessor it emulates. It is important you understand the output driving and input loading characteristics the emulation equipment presents to the target system.

For the address, data, and control signals of the emulation processor at the pins of the processor probe, the input and output characteristics are identical to low power Schottky transistor-transistor logic (LS TTL). Reference may be made to the specifications of any LS TTL manufacturer for the parameters of input loading current, output driving current, and three-state loading characteristics.

The capacitive loading of the probe is the sum of the capacitive loading of LS TTL gates and the capacitance of the probe cable (which is approximately 20 pF).

$$C_{\text{TOTAL}} = C_{\text{LS GATE}} + C_{\text{CABLE}}$$

The clocks supplied to microprocessor chips generally come in one of three forms. Clocks are supplied at MOS levels as high as 15 volts, at TTL levels, or as direct inputs from a lead of a crystal. The emulation processor of the 64000 allows all of these methods for driving the emulation clock rates. It is possible to slow down the emulation processor to a desired speed by using a slow user clock, if within the specifications of the processor manufacturer. If the internal clock is selected, the emulation processor will operate at full rated speed. A table of loading values and specifications for the clock options are as follows:

CLOCK, High-level

Levels: as specified by processor
C = C spec + C cable

CLOCK, TTL

Levels: TTL as specified by processor
C = C spec + C cable

CLOCK, Crystal Inputs

Accepts frequency determining elements as specified by the processor manufacturer unless otherwise noted.

The timing of processor signals at the probe plug is designed to emulate exactly the timing of signals of a microprocessor chip normally inserted in the same plug. Specifically, the setup time, hold time, propagation delay time , and clock period will satisfy the published processor specification when referenced to any signal applied to the processor at a processor probe pin.

When emulating processors such as the Z80, which have an internal dynamic RAM refresh or DMA capability, the emulator pod will continue to execute these functions when emulation is not in process.

The emulation capability of the 64000 will support processors with a maximum clock cycle speed of 160 nanoseconds.

NOTE

The emulation probe presents greater drive capability and slightly greater capacitive loading to the target system than the processor being replaced. Consequently, it is conceivable that a user's system, which operates under emulation, may not operate properly when driven by a microprocessor IC. Noise margins and signal levels in marginally overloaded designs may not cause problems when driven by emulation but may be fatal to system operation under normal microprocessor drive conditions. Be sure that your design allows for the added drive and loading specifications of the Model 64000 emulation probe.

System Operation

Purpose

This section presents an overview of emulation and analysis in the 64000 system. Understanding this material will make it easier to understand the following chapters. Please note the following material is conceptual and is not a complete description of the design of the 64000.

Bus Structures

The 64000 contains four basic bus structures:

SYSTEM bus

EMULATION bus

MEMORY bus

IOD bus (input/output data bus)

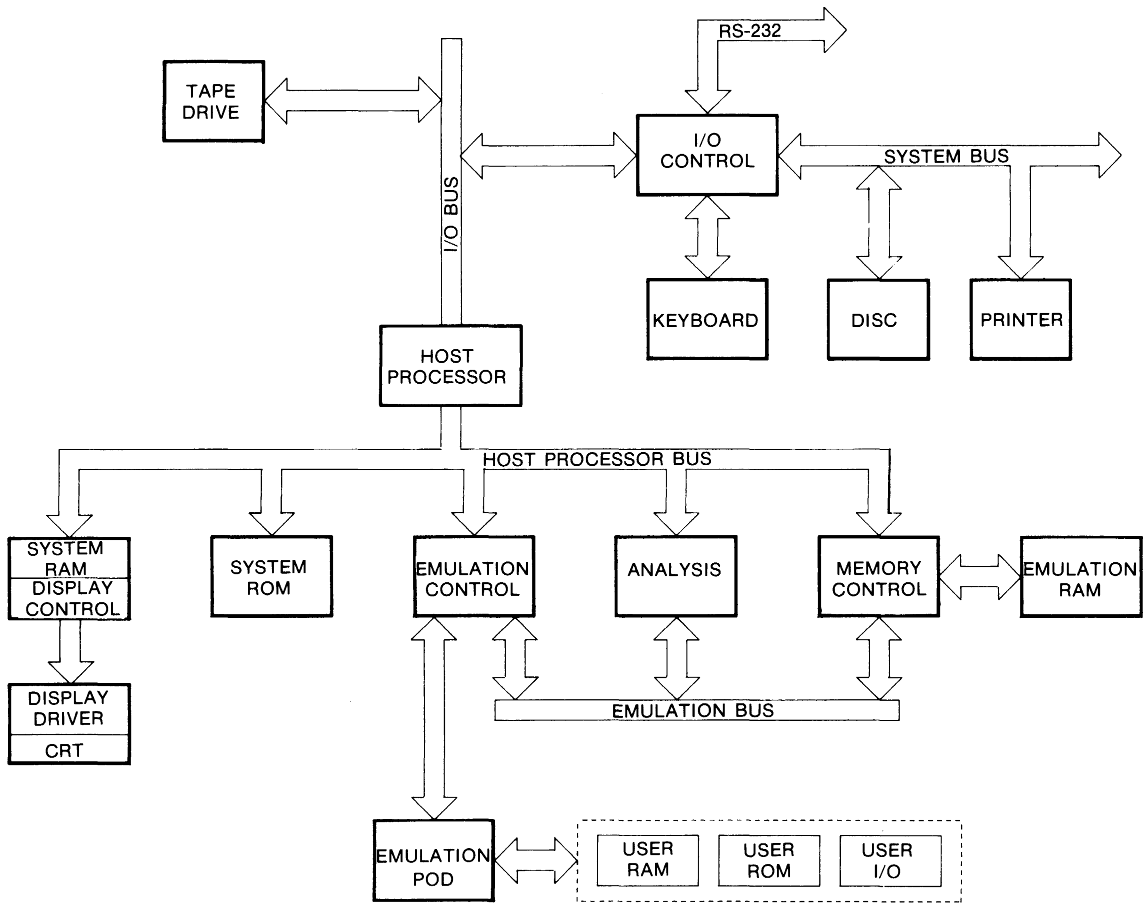


Figure 2-2. Bus Orientation

The last three buses in the list are visible in the system chassis as ribbon cables extending across the top of the circuit cards.

The SYSTEM Bus is the address, data and control buses of the 64000 system control processor. This bus carries the majority of the control signals, commands, and data to and from the system elements.

The EMULATION bus is the address, data, and control buses of the emulation processor. This bus is extended on the processor probe and is used to drive the microprocessor bus systems of the target system.

The MEMORY bus is the major artery through which both the system and emulator have access to emulation memory.

The IOD bus is dedicated to input and output devices of the 64000 system. It handles data to and from the minicartridge tape drive, the keyboard, the disc drive, the printer, and the system processor.

All data transfers in the system occur on these buses. For example, to display user RAM memory, a command is transmitted on the system bus from the system processor to the memory control circuit board. When memory is available, data is accessed from the memory boards and routed on the emulation bus to the memory control board. The memory control board transmits data to the system processor on the system bus and from there the data is passed via the IOD bus to the display controller for display on the CRT. To display emulation memory, data is accessed in the same manner except that data is transmitted from emulation memory to the memory control card through the memory bus. From that point on, data is transferred in the same sequence as above.

Loading Absolute Code

Program code to be emulated must be absolute code, that is, relocated and linked to form a single module. The entire object code module is loaded as a whole into memory from the system discs or tape. The only exception is the case in which user equipment contains program code in ROM's that are used by the emulated program.

The first step in emulation is to map the memory space. Mapping is done by specifying emulator or user RAM or ROM space and illegal memory.The resulting mapping information is loaded through the system bus to the memory controller. Mapping sets up address boundaries in the memory control hardware. The memory control board then determines the memory source or destination of data by monitoring the address bus of the emulator processor. If an address falls within the space specified by user RAM, the memory control allows the buffer surrounding the emulator processor to access the target system memory. Otherwise, the memory control directs access to the emulator RAM. If an attempt is made to access illegal memory or to write to read only memory, the memory control board sends a signal to the emulation control. This signal halts execution to allow inspection of the processor state that caused the illegal access. These functions happen automatically and do not require interaction with the user.

The input/output space must also be mapped. The 64000 allows access to the cathode ray tube (CRT), tape, disk, keyboard, and printer. Interface to these items are simulated as data ports whose addresses are assigned by the user before emulation run time. These interfaces are handled by the system controller and are not exact duplicates like interfaces that would be experienced outside the emulation system. Therefore, the user interface is, simple, but it is not, necessarily, exactly like the interface that you will finally see in your target system.

Once the memory and the I/O space is mapped, the emulation memory can be loaded. Loading is accomplished by reading an object code disc file and writing the code to the emulation or user memory. The portion of code specified as user RAM gets loaded into the target system. The portion of the code for emulation RAM goes directly to the emulation memory. The memory control board specifies the destination of the code loaded from disc. The operator does not have to give loading instructions to the system because this is automatically accomplished in the hardware. The data loaded will divert destination in accordance with the memory map that is previously entered by the user.

Running and Stepping the Processor

There are major differences in the method of emulation between stepping and running the same processor.

Once program code has been loaded into the emulation memory and a system command has directed the emulation microprocessor to begin execution, it continues to run in real time or as fast as the emulation processor clock source will operate. The emulation processor runs independently of the operating system. This means, consequently, that the operating system may be used to do other things (such as tracing events or displaying portions of memory) while the emulator is running. It also means, however, that when running in real time, the analysis function on the analysis board is the only method to monitor execution of the emulation processor. Satisfying a trace sequence in the analysis board will stop the processor so that the internal registers may be examined.

When stepping, the emulation processor executes one instruction at a time. Each step occurs with consecutive strokes of the return key after having entered the step mode. After each step, the condition of the registers and any memory transactions (if there were any) are reported on the step display.

The following occurs in the emulation system during the step mode. A small memory, called Background Memory, has been loaded with the previous register contents of the emulation microprocessor. (See figure 2-3.) The emulation processor is directed, by the operating system, to execute a series of instructions which restore the registers from the background memory. The next program instruction to be executed is loaded into the background memory followed by an instruction added by the operating system which causes a jump back to the stepping program. The controller forces the processor to execute the next instruction (in background memory). As the program counter increments and the next instructions are executed, the processor will return immediately to the operating system, having executed only one program instruction. The system program causes the emulation processor to execute instructions that store its register contents in background memory. The stored contents are then displayed. The emulator acontroller will then wait for a new command from the operating system. The procedure will restart and recycle every time the return is pressed.

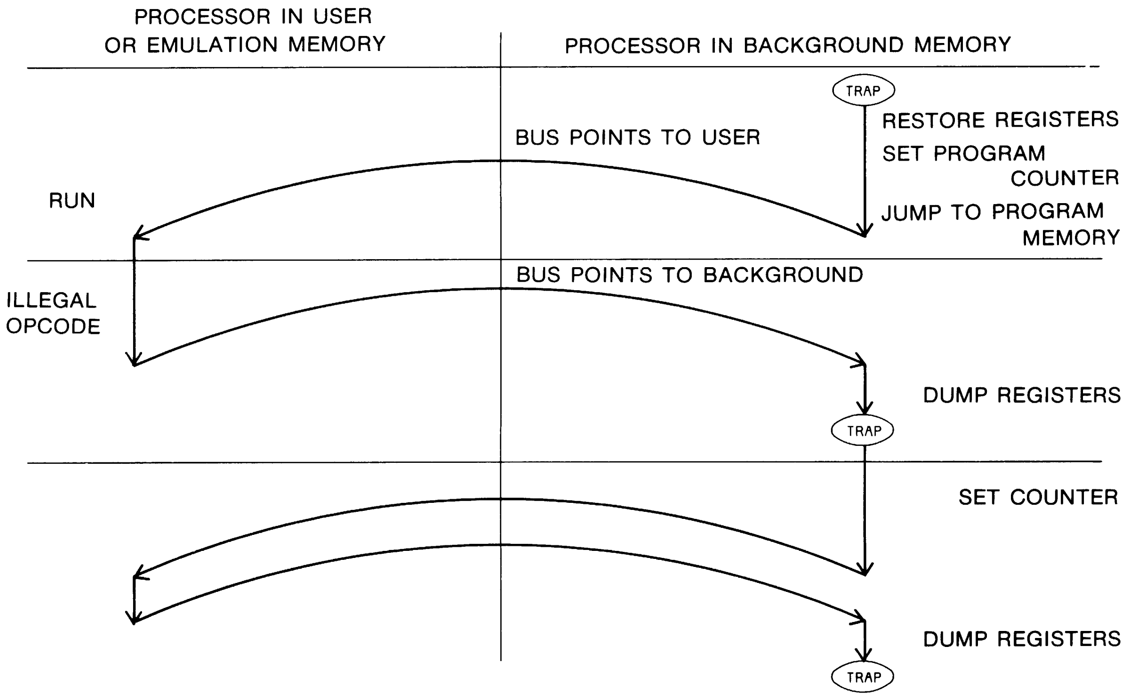


Figure 2-3. Run/Step Diagram

The Analysis Module

There are two lines of thought concerning the design philosophy of logic analyzers. One theory suggests that the power of a logic analyzer instrument is directly proportional to its memory capacity; that is, instrument usefulness is related to the number of executed states that can be remembered by the analyzer instrument. The other philosophy suggests that the power of a digital analyzer lies in its triggering and capturing capabilities. Hewlett-Packard has adopted the latter point of view and has made the analysis capabilities of the 64000 easily adaptable and versatile. Because of the wide range of triggering capabilities, the 64000 is an extremely efficient tool, in the process of debugging microprocessor hardware and software.

The analysis board may be viewed as a logic analyzer which uses the system CRT as its display. In many respects it is similar to the Hewlett-Packard HP Model 1611 Microprocessor Analyzer.

Once a trace has been requested by the operating system, the analysis module begins to look for "states" that satisfies the trace specification until stopped by a command from the operating system or until the trace specification is met. Trace specifications are entered into the operating system by the user. The entered specifications are loaded into the analysis board trigger hardware by the operating system via the system bus. The analysis board continually monitors the transactions between elements on the emulation bus. When the required trigger occurs on the emulation bus, the analysis board stores the previous or consecutive bus states in a dedicated analysis memory. When the display trace command is initiated from the console, the operating system accesses the analysis memory via the system bus. The content of trace memory is retrieved, formatted, and displayed on the CRT screen. If desired, the operating system will disassemble the code found in the trace memory into mnemonics of the processor being emulated.

Chapter 3
Installation

Installation Procedures

Before installing the emulation equipment, refer to the Model 64000 Overview Manual for instructions concerning the installation of the system work stations, disc, and printer. When the system has been installed and is fully operational, install the emulator hardware as described on the following pages.

CAUTION

Before installing any of the emulator boards into the work station ensure that the station power switch is set to "OFF".

Hardware Configuration

The emulation and analysis circuit boards can be installed in any card slot of the station chassis. See figure 3-1 for a typical card grouping. Installation of the Emulation Control board in the rear most slot of the board grouping maximizes the free cable length outside the work station chassis. In addition, by installing the Analysis, Memory Control, and Memory boards in the positions specified, the length of the emulation and memory bus connector cables is minimized.

The emulator pod and user socket will extend from the Emulation Control board when installation is complete.

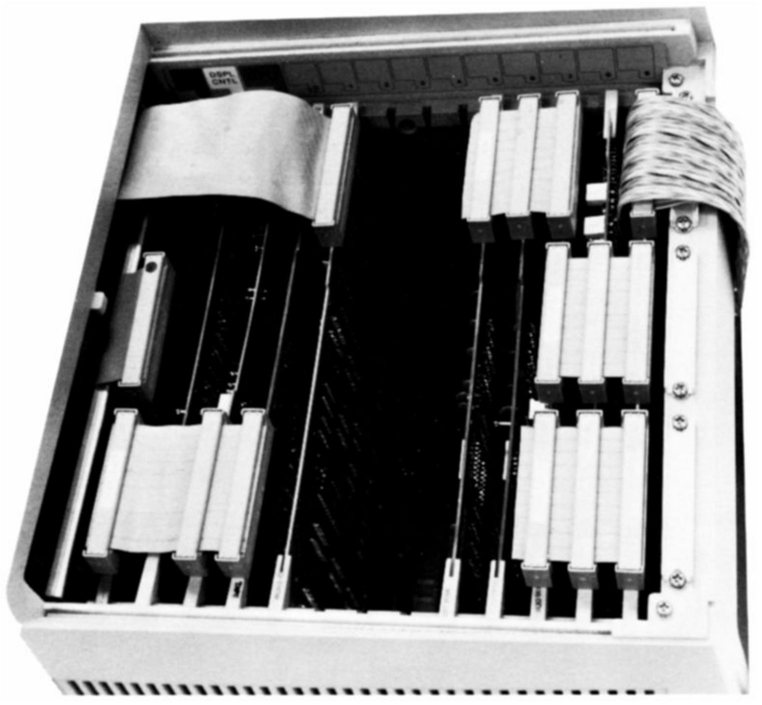


Figure 3-1. Card Slot Configuration

Installing the Emulation Control Board and Emulation Probe

The first step in the installation of the Emulation Control Board and the Emulation Probe is to connect the Emulation Probe to the Emulation Control Board. Two multi-colored ribbon cables are used to connect the control board to the probe. Each cable is terminated with a different type connector. Pin 1 on the cable connectors is indicated by a triangle molded into each connector. Mating male connectors are located on the top left of the control board (as you face the component side with the wide male edge connector at the bottom). Pin 1 on the board is indicated by "Pin 1" silk-screened on the board surface. Being careful to align pin 1 of the male and female connectors, connect the probe to the control board.

Install the Emulation Control board into the station chassis by aligning the board in the circuit card guides, with the component side of the board facing the front of the chassis, and applying a gentle downward pressure until the board is seated in the mother board connector. Be sure the ejector handles are in their full horizontal position when the board has reached its full downward travel.

Installing the Analysis Board

The Analysis Board must be installed in the next slot toward the front of the 64000 from the Emulation Control board. For example, if the Emulator Control board was installed in slot 9, the Analysis board should be installed in slot 8. Install the board in the station using the procedure described for the Emulation Control Board. As with the control board, assure that the ejector handles are in their full horizontal position when the board is seated in the chassis connector.

Installing the Memory Control Board

Set the "DATA BITS" switch to the proper data bus width of the emulation processor: 8 bits wide or 16 bits wide. Facing the component side of the Memory Control board, the "DATA BITS" switch is located just below the space between the connectors located on the top edge of the circuit board (see figure 3-2). Set the switch to the far right for 8-bit emulation. Be sure that the switch has traveled as far right as possible.

Set the "ADDRESS BUS WIDTH" jumper cable to the proper address bus width of the emulation processor. The jumper cable, which is located in the center of the component side of the board, has two connectors. The lower connector is soldered to the board and is not removable. The upper connector is movable and mates with a male connector block soldered permanently to the circuit board. Beside the male connector block is a series of horizontal bars labeled as "BIT 14", "BIT 15", etc., up to "BIT 20". The address bus size of the emulation microprocessor is selected by moving the jumper cable connector vertically on the male connector block until the desired bus size is indicated on the bit number scale. For example, if the emulation processor has a 16-bit address bus, the jumper cable connector is positioned so that "BIT 16" mark is not covered by the connector but "BIT 17" mark is. Likewise, for a 20-bit address bus, all marks are uncovered. In figure 3-2, the "ADDRESS BUS WIDTH" jumper is set for a 16-bit address bus.

The Memory Control board is installed in the next vacant slot forward of the Analysis board. If no Analysis board is used, the Memory Control board can be installed in the slot next to the Emulation Control board or in a slot two positions forward of the Emulation Control board, leaving a slot vacant where the Analysis board would normally go. Install the Memory Control board using the procedure described for the Emulation Control board and assure that the board is firmly seated in the chassis connector.

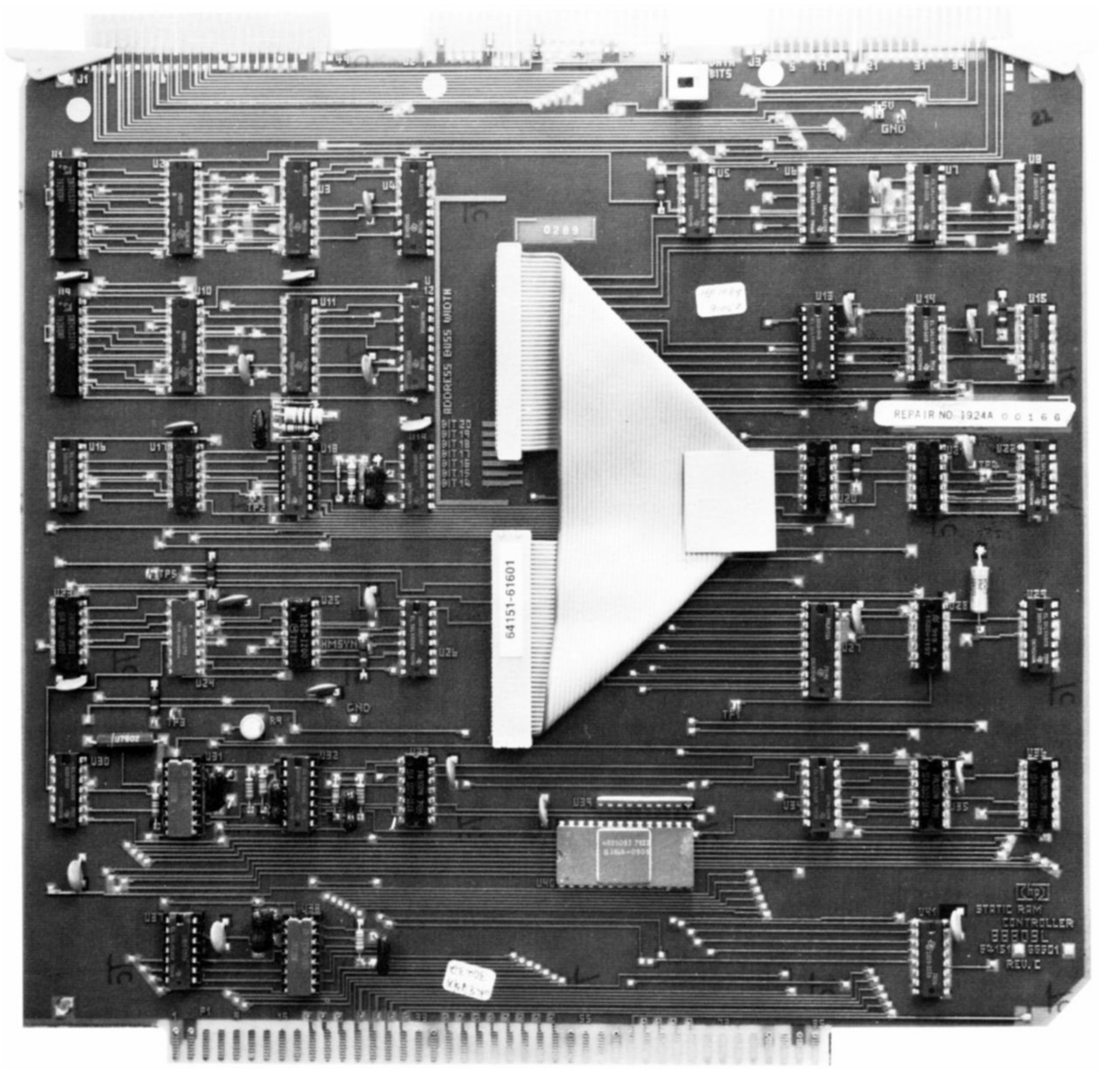


Figure 3-2. Memory Control Board

Installing the Memory Boards

Each Memory board may contain from 8K to 32K bytes (4K to 16K words) of Random Access Memory (RAM) in units of 8K bytes. The address space of each card must be specified by installing an 8-pin jumper plug in address select socket U11. Facing the component side of the memory board with J1 at the bottom, U11 is located on the upper right corner. Label boxes indicating the address range options in words are silkscreened on the board below U11 (see figure 3-2). An address range is selected by installing the 8-pin jumper plug into the half of the socket associated with the address range label being selected. For example, in figure 3-2, address range 0 to 16K (words) is selected because the jumper plug is in the four left most pin positions of U11.

The address range specification does not limit the address ranges which may be emulated. For instance, if there is one fully loaded emulation memory card, it may be used to emulate any combination of up to 32 1K address blocks. The address range specification is used only by the emulation system to distinguish one memory board from another. It does not establish addresses for emulation.

The address range specification rules are listed below:

a. One board with at least 8K bytes of memory must be specified as address range 0 to 16K.

b. If two memory boards are installed, one must be specified as the 0 to 16K range while the other is specified as 16K to 32K.

c. Only the two positions of U11 may be used for address range selection. The full address range of 0 to 64K bytes (0 to 32K words) is available from these two selection positions. U10 must remain open for 8-bit emulation.

d. Only one jumper plug can be inserted into socket U11 on each board.

Install the Memory board(s) in slot(s) adjacent to the Memory Control board with the component side of the board facing the front of the work station. Assure that the board(s) are firmly seated in the chassis connector(s).

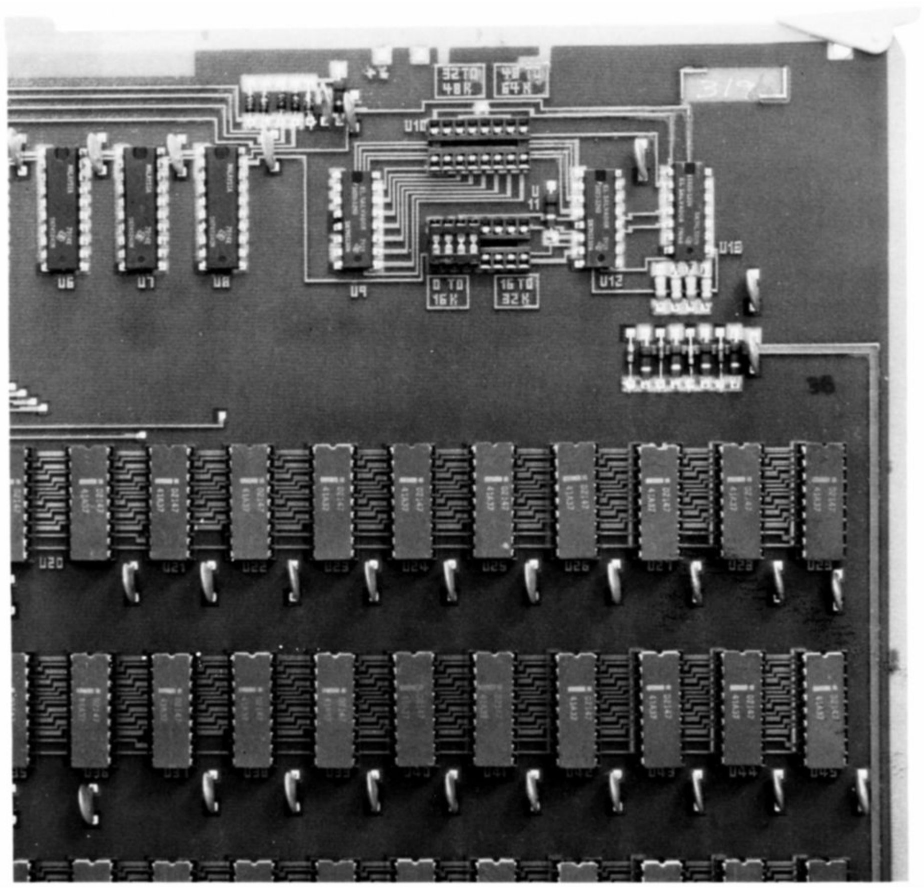


Figure 3-3. Address Range Selection Sockets

Installing the Bus Jumpers

After all the emulation and analysis circuit boards have been installed in the work station card cage, the emulation and memory bus jumpers must be installed across the top of the board set. The emulation bus connects together the Emulation Control board, the Analysis board, and the Memory Control board and consists of two jumper cables each of which contains three female connecters. Connect one of the cables to the edge connecters in the center and the other to the connecters on the right side of the top edge of each board as viewed from the front of the station (figure 3-4). Be sure that the key on the edge connectors of the circuit boards aligns with the key on the cable connector blocks.

The memory bus connects the Memory Control board to the Memory board(s) and consists of one cable with either two or three female connectors. Connect the Memory bus jumper to the connecters along the left side of the top edge of the Memory Control and Memory boards assuring that pin "1" on the board edge connector aligns with pin "1" of the connector block (figure 3-4).

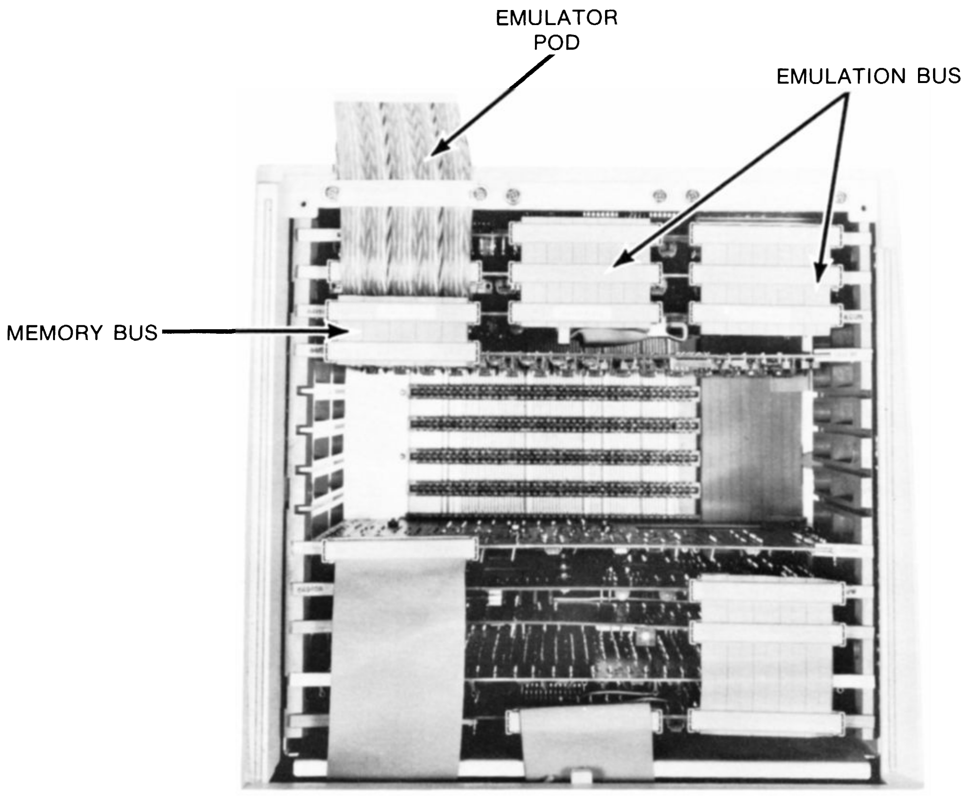


Figure 3-4. Memory and Emulation Bus Cabling

Emulation Probe Installation

When the emulator is used with a target system, the user socket extending from the the Emulator Probe should be installed into the target system as follows:

a. Switch the target system power supply off.

b. Remove the processor chip to be emulated from its socket in the target system.

CAUTION

The emulator pod contains devices which are susceptible to damage by static discharge. Therefore, operators should take precautionary measures before handling the user socket to avoid damage to the emulator.

c. Insert at least one of the pin extenders, which have been provided, onto the user socket. If the socket is to be installed on a densely populated circuit board, there may not be enough room to accommodate the plastic shoulders of the socket and additional extenders may need to be inserted.

CAUTION

The 40-pin extenders protect the socket from damage while it is being inserted and removed from the microprocessor socket. DO NOT use the socket without at least one 40-pin extender installed.

d. Install the user socket into the microprocessor socket using light downward pressure. The red-edged side of the cable must align with the end of the socket corresponding to pin one of the processor as shown in figure 3-5.

CAUTION

Do not install the emulation probe socket into the processor socket with power applied to the target system. The probe can be damaged if power is not removed before installation.

In addition, be sure the socket is inserted into the processor socket so that the red edge of the cable aligns with the pin 1 end of the processor socket. Damage to the emulator can result if the socket is installed in reverse.

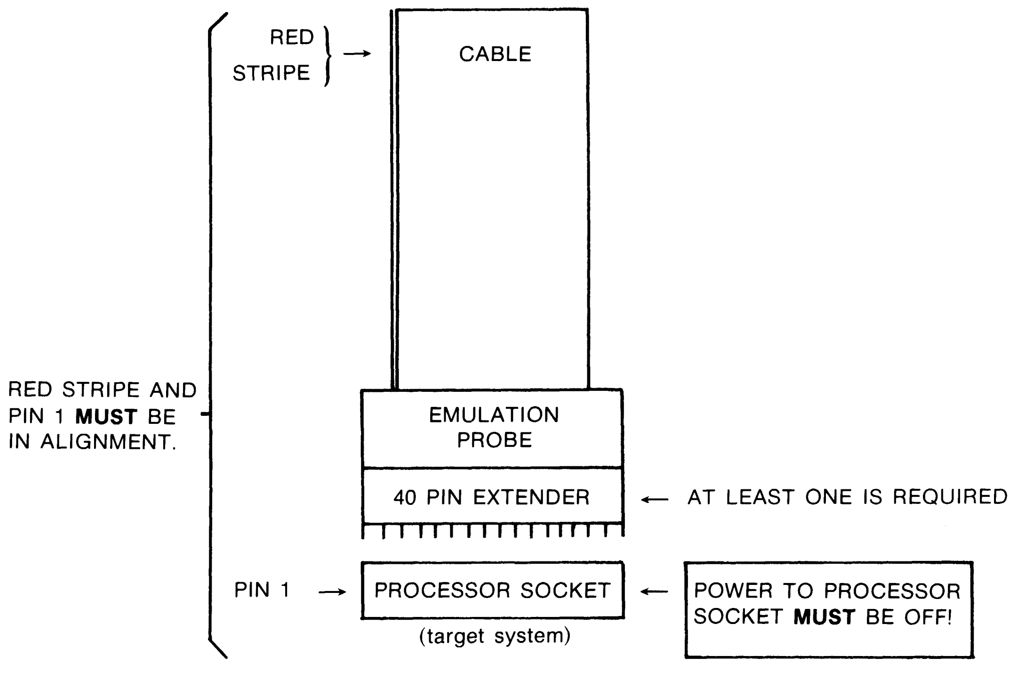


Figure 3-5. Installing the Emulation Probe

Grounding Considerations

Within the work station, the electronic and chassis grounds are tied together. If problems occur within the target system as a result of induced noise from the power distribution lines, refer to the section entitled "Grounding Considerations" in Chapter 9 of the Overview Manual.

Radio Frequency Interference

With an emulation system installed in the Model 64000, several methods of operation (physical setup) may result in an increased emission of radio frequency noise. To reduce the rf noise level, any of the following techniques may be used:

a. When the emulator is used infrequently, disconnect the Emulator Pod and cables from both the host system and target system.

b. For systems that use the emulator intermittently, select "external clock" and disconnect the user socket from the target system when not in use.

c. Consistent with design needs, minimize the time that the emulator is used without being connected to a target system.

d. All 64000 system covers should be in place and properly attached to the mainframe (all housing screws tight).

NOTE

Running the emulator while connected to a target system produces little additional rf noise above that generated by the target system itself.

Chapter 4
Setting Up Emulation

Emulation Software Requirements

The emulation processor executes absolute code, which is supplied in one of four ways:

a. from ROMs contained in the target system,

b. by assembling source code and linking the resulting relocatable code,

c. by compiling a high level language and linking the resulting relocatable code, or

d. by using a combination of the above three methods.

Methods b and c have certain definite advantages in system debugging. The assembler and the compiler generate local symbol files (asmb_sym) for the relocatable code. Local symbols are those symbols declared in each source file program module. In assembly language programs, symbols are defined in the label field of a source statement or the label field of an EQU pseudo instruction.

The linker creates a global symbol file (link_sym) and stores the file under the same file name as the assigned absolute image file name. Global symbols are those symbols declared as global with the GLB pseudo instruction in assembly language programs and/or with the GLOBPROC and GLOBVAR directives in Pascal programs. The global symbol file, and the asmb_sym file from each program module are passed to the 64000 development station when the absolute code generated by the linker is loaded into emulation memory or target system memory. These symbols are used for symbolic referencing during emulation. Symbolic referencing allows use of a symbol where an address location would normally be specified.

If absolute code is generated and programmed directly into user ROMs external to the 64000, local and global symbol files will never be passed to the 64000. Although the emulation processor will execute code resident in user ROMs, symbolic addressing will not be available since no symbol tables exist.

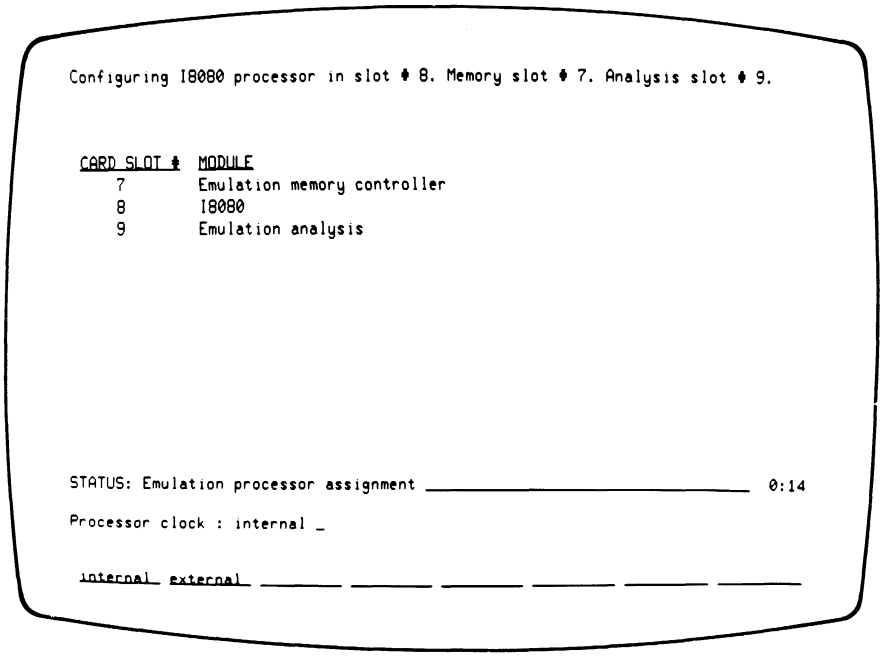
Soft Keys and Directed Syntax

One of the most notable features of the 64000 is the implementation of soft keys and directed syntax. As with the other system soft keys, the emulation soft keys are identified by displayed command labels. When a soft key is pressed to initiate a particular command, the soft key label moves to the command line and the soft key labels change to a list from which the next entry can be selected. This process continues until an entire command has been entered.

If a soft key is in capital letters and is surrounded by an angle bracket (e.g., <FILE>), a message will be displayed on the status line when the key is pressed. The message briefly describes the expected entry.

Emulation Initiation

Emulation is initiated by pressing the **emulate** key followed by RETURN. Although the emulate command has many additional options, only the simplest form of the command, **emulate** RETURN, is discussed here. Additional emulate command options are explained in the emulate command syntax section of this chapter.



```
Configuring I8080 processor in slot # 8. Memory slot # 7. Analysis slot # 9.


CARD SLOT #  MODULE
     7       Emulation memory controller
     8       I8080
     9       Emulation analysis




STATUS: Emulation processor assignment ____________________ 0:14
Processor clock : internal _

internal  external
```

Figure 4-1. Emulation Configuration Display Board Slot Assignments

Once the the emulate command has been entered, the display will change to that shown in figure 4-1. At the top of the display, the physical configuration of the emulation circuit boards are stated. In the center of the display, the physical configuration is listed in tabular form. That is, the emulation analysis board, emulation memory controller, etc, and their corresponding 64000 slots are listed. This information allows the user to answer the configuration questions displayed if more than one memory controller is installed in the 64000.

Emulation Configuration Question-Answer Sequence

Emulation configuration consists of a question and answer sequence which prepares the emulator for a particular application. Once the questions have been answered for the particular application, the answers can be stored on disc so that the question and answer sequence need not be repeated for each emulation session. If changes to an emulation command file are desired, the file can be edited using the **edit_conf** key. This allows changing only specified answers. At the end of the edit_conf sequence, a new file name can be assigned to the edited configuration, or the old file can be written over with the new information.

Throughout this discussion, the available soft key entries for each question are listed following the question. If an emulation command file is being edited to reconfigure the emulator, the default responses provided are the responses that were entered when the command file was originated or last edited.

The questions are divided into the six sections listed below.

a. Emulation Control Board and Memory Control Board Selection

b. Clock Selection

c. Real Time Selection

d. Memory Configuration

e. Simulated I/O

f. Command File Designation

These sections are discussed on the following pages. The questions discussed in the first section are only presented when more than one type of emulation control board and/or more than one memory control board is installed in the 64000.

Emulation Control Board and Memory Control Board Selection

The questions presented in this section determine which emulation control board and memory control board are to be used for the current emulation session only when more than one type of emulation control board and/or more than one memory control board is installed in the 64000.

Emulation processor type?

The response to this question determines which type of emulator is to be used for this session. The soft keys will display the types of emulation processor controllers installed in the 64000. The emulator type is selected by pressing the appropriate soft key and the RETURN key.

Emulator card slot?

This question is presented when two or more emulator control boards of the selected type are installed in the 64000. The soft keys will display the numbers of those slots containing the control boards. The emulation control board to be used for the current session is selected by pressing the appropriate soft key and the RETURN key.

Memory Controller slot?

This question is presented if more than one memory control board is installed in the 64000. The selection is similar to that for the emulation control board slot selection described above.

NOTE

If one memory control board is installed, the system assumes that it is connected to the selected emulation control board and/or vice versa. If the selected memory and emulation control boards are not connected, an error will occur when memory access is attempted and the emulator will not function.

When all slot and processor type selections have been made, the system updates the configuration statement at the top of the display.

Clock Selection

The emulation processor can be driven by the clock contained in the emulator pod (internal clock) or by a clock which is supplied by the user (external clock). The response to the following question determines which of the clocks will be used.

Processor clock? internal external

The internal clock is normally used when the emulator is to be run out-of-circuit; i.e., with no target system. The internal clock is provided by the emulator for use in development of the target system software when the target system hardware is not available.

An external clock is used when the emulator is connected to a user target system. The external clock is part of the target system hardware and should be used to provide timing to the target system during in-circuit emulation applications. In the case of processors which do not have on-board oscillators, use of the external clock is mandatory during target system exercises.

Clock Speed Selection

As discussed above, the emulator provides a choice of an internal clock or an external clock for the target processor. If "external" is selected, the following question is presented.

Is clock speed greater than X MHz? yes no

(where "X" is dependent on the processor being emulated)

If the answer is "yes", one or two "wait" states are added for each access to emulation memory.

"No" can be selected even though the target system clock is greater than the speed specified in the question. This will force the emulator to not add "wait" states for emulation memory accesses. Because worst case conditions are used in determining the maximum clock speed specification, the typical system may operate satisfactorily without "wait" states. This is acceptable, but the user must be aware that this operation is beyond specification and is not guaranteed.

Real Time Selection

Restrict to real-time runs? no yes

This question provides an opportunity to restrict the emulator to real-time program execution.

Some of the emulation features can not be executed in real time. Therefore, the limitations which are imposed when runs are restricted to real time should be understood.

If runs are not restricted to real time, then all of the emulation features can be used. The implications of nonreal-time program execution to operation of the user system must be understood, however. Some real time control systems may malfunction or not run at all if the program is halted or if the monitor is entered during code execution.

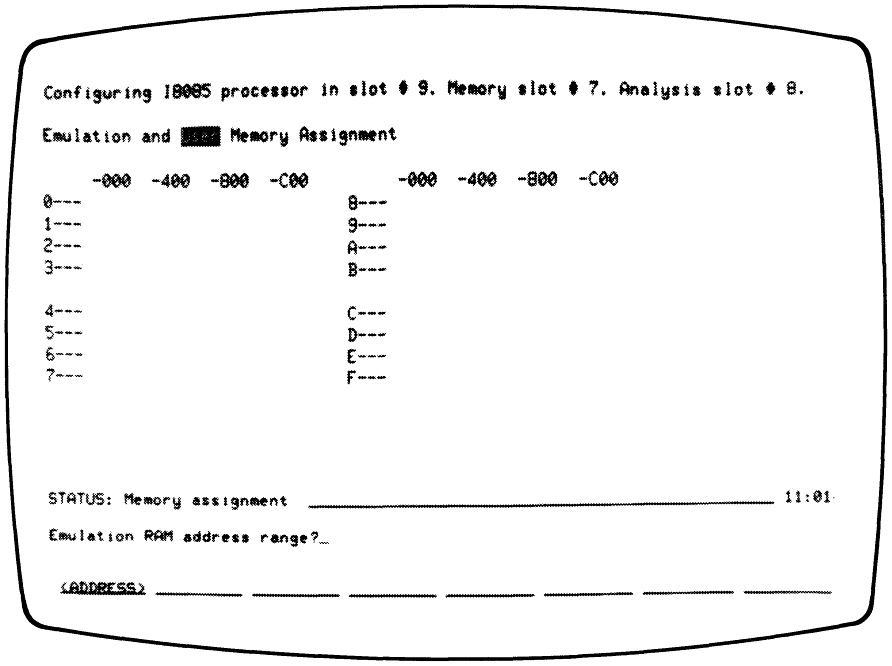
If operation is restricted to real-time runs, emulation features which require the host processor to access emulation memory can not be implemented because emulation memory can not be accessed during a user program run without interfering with real-time program execution.

Stop processor on illegal opcodes? yes no

This option helps find unexpected executions in absolute code. If yes is selected, the processor will stop emulation if an invalid opcode is fetched. If no is selected, the emulation processor will attempt to execute the opcode in the same manner as the microprocessor unit being emulated.

Memory Configuration

During the memory configuration question and answer session, the display changes to the memory matrix shown in figure 4-3. The configuration statement is displayed at the top of the screen and below it the memory matrix is displayed. The STATUS line indicates that the system is expecting memory assignments, the results of which will be displayed in the matrix.



```
Configuring I8085 processor in slot # 9. Memory slot # 7. Analysis slot # 8.

Emulation and User Memory Assignment

     -000  -400  -800  -C00         -000  -400  -800  -C00
0---                          8---
1---                          9---
2---                          A---
3---                          B---

4---                          C---
5---                          D---
6---                          E---
7---                          F---



STATUS: Memory assignment  ______________________________________ 11:01

Emulation RAM address range?_

 <ADDRESS> ________ ________ ________ ________ ________ ________ ________
```

Figure 4-2. Emulation Configuration Display - Memory Assignment

The displayed memory matrix is divided into two, 4-column by 8-row matrices. Each of the 64 row-column intersections correspond to a 1024-byte memory section. The column headings are the low order addresses and the row headings are the high order addresses used to assign the corresponding byte section (or sections). The sections may be assigned as either emulation RAM, emulation ROM, user RAM, user ROM, or illegal memory address range.

"Emulation Memory" is memory physically located in the 64000 system. "User Memory" is memory physically located in the target system. Memory can be assigned entirely to user, entirely to emulation, or to a combination of the two intermixed.

The memory control board determines the memory source or destination of data by monitoring the address bus of the emulator processor. If an address falls within the space specified as user memory, the memory controller allows the buffer surrounding the emulator processor to access the target system memory. Otherwise, the memory controller directs access to the emulation memory.

The RAM, ROM, or illegal memory assignment determines how memory is protected, as shown in table 4-1. If an attempt is made to access illegal memory or to write to memory mapped as ROM, the memory control board sends a signal to the emulation controller. This signal halts execution to allow inspection of the processor state that caused the illegal access.

Table 4-1. Memory Types Read/Write Protected

Memory Type	Write Protected	Read Protected
RAM	NO	NO
ROM	YES	NO
ILLEGAL	YES	YES

Memory protection is provided to help detect programming bugs. For example, if a memory location is specified as ROM memory, it is write protected. Any attempt to write to a ROM location will cause emulation to be halted. A warning message will be written on the STATUS line indicating the reason for the halted emulation. Similarly, any read or write to memory labeled "Illegal memory" will halt emulation and display the appropriate message.

Memory assignments are made by answering the following questions, which are presented one at a time.

Emulation RAM address range? <ADDRESS>

Emulation ROM address range? <ADDRESS>

User RAM address range? <ADDRESS>

User ROM address range? <ADDRESS>

Illegal memory address range? <ADDRESS>

The soft key displayed is <ADDRESS>. If the soft key is pressed, the following prompt is displayed:

STATUS: Valid memory address

The prompt indicates that an address is expected by the system. An address may be expressed as a decimal, hex, octal, or binary number.

When the first address of the range is entered, the soft key display changes to **thru**, which indicates that the last address in the range must be entered. To enter the last address, press the **thru** key, enter the address, and press RETURN. The address assignment is displayed in the memory matrix (see figure 4-2). At this point, another address range may be assigned or the current assignment may be terminated, and the next assignments initiated.

To assign the next block of memory in the current memory assignment, repeat the above procedure.

To terminate the current memory assignment, and initiate the next assignment, press RETURN with no address range specified.

Guidelines for answering the memory mapping questions are listed below.

- Memory assigned as emulation memory is indicated by RAM or ROM being displayed in normal video.

- Memory assigned as user memory is indicated by RAM or ROM being displayed in inverse video.

- Memory must be assigned as a range of addresses; i.e., address through address. However, the address range may be within one 1024-byte section.

- If an address range includes any location in a section, the entire section is assigned as the current memory type.

- Memory sections may be mapped in separate, disconnected sections. After the entry of each address range, the prompting question will return to request another address range. Memory types may be mixed in adjacent sections throughout the matrix.

- Each type memory question is terminated by pressing RETURN, with no entry specified. If no memory is to be allocated to a particular function, no entries are made in response to <ADDRESS>. The RETURN key is pressed to continue to the next question (i.e., if no user RAM memory is to be mapped, the RETURN key is pressed when the question "User RAM address range?" is asked. The "User ROM address range?" question will then be asked).

- If there is insufficient emulation memory to support an attempted memory assignment, only the available memory within the specified address range is assigned to the current memory type and a status message to that effect is displayed. All other memory remains illegal unless defined as user memory.

- All unassigned memory defaults to illegal memory.

Simulated I/O

NOTE

Chapter 8 of this manual covers simulated I/O exclusively. It should be read, studied, and understood before attempting to use the simulated I/O feature.

The simulated I/O feature of the 64000 allows the user to develop programs for, but without actually using, the target system's I/O hardware. To accomplish this, the I/O hardware in the 64000 is used to "simulate" the hardware of the target system. To use this feature, the question and answer session described below is performed.

NOTE

During program emulation, all simulated I/O files must be opened by the user program. However, during the time that either the simulated I/O display or keyboard file is opened, the standard 64000 keyboard has no control over the display. To regain control over the display and/or keyboard before the user's program closes the file, press the soft key labeled **simulate** .

Simulate I/O?

If simulated I/O is not to be used during an emulation session, press the **no** key and proceed to the Command File Designation paragraph following this simulate I/O question and answer session.

If simulated I/O is to be used, press the yes key. The questions listed below will be asked, in the sequence given.

Simulate display?

Simulate printer?

Simulate rs232?

Simulate keyboard?

Simulate disc files?

If the 64000. is not to be used to simulate a particular I/O, press the no key and then the RETURN key as each question arises. Proceed with the next question.

If the 64000 is to be used to simulate a display, printer, rs232, or keyboard I/O for the users program, press the yes key when the particular question arises. Then enter the control address in response to the "Control address?" prompt (see figure 4-3).

NOTE

The control address is the location to which all I/O handshaking codes are sent by both the user and the 64000 programs. This address must first be defined in the user's program. Then, the control address is entered into the 64000 in answer to the "control address?" question. Refer to Chapter 8 for a complete description of the control address.

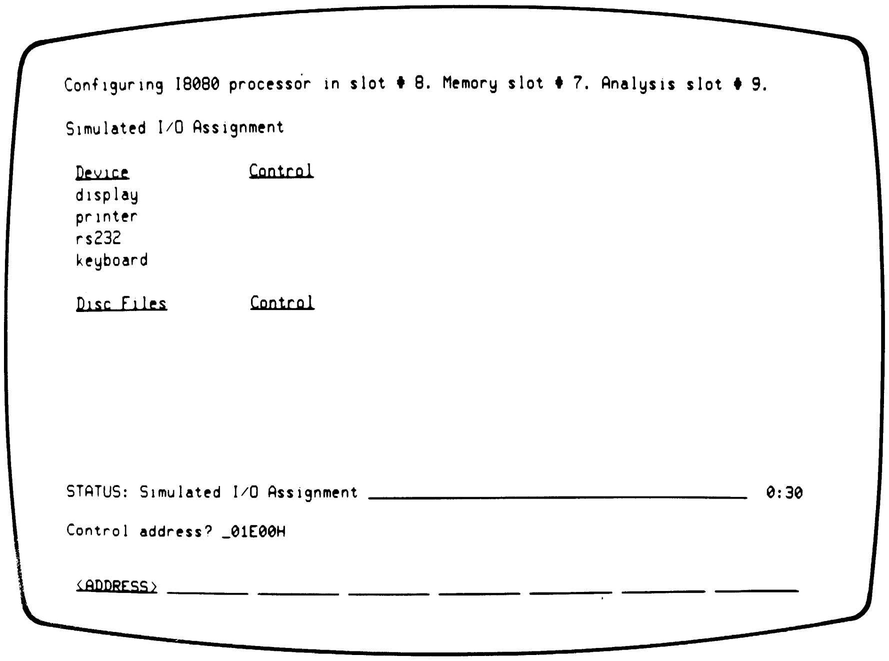
If the 64000 is to be used to simulate disc files, the entries are similar to the previous entries except multiple disc files may be set up. Also, each file must first be given a file name before assigning a control address. The file name may be either a name of an existing file or it may be a pseudo name that may later be changed by the user program (refer to Chapter 8, Disc File I/O Interface, Changing File Name Command).

Once the yes key has been pressed, the following two questions are displayed, in sequence.

File name? (See figure 4-4.)

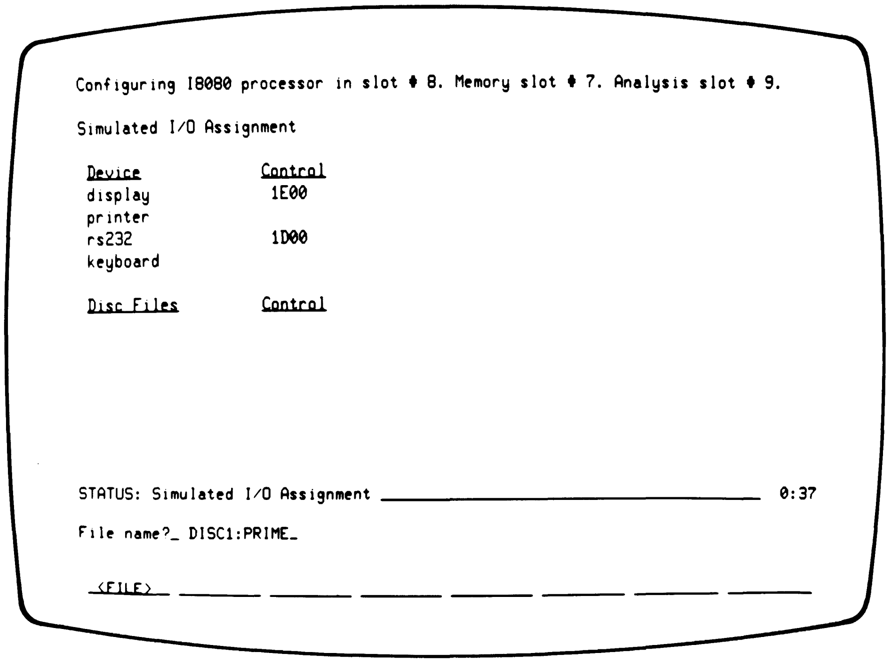
Control address? (See figure 4-3.)

These questions are repeated so that multiple files may be set up. To terminate file assignments, press the RETURN key without entering a file name.



```
Configuring I8080 processor in slot # 8. Memory slot # 7. Analysis slot # 9.

Simulated I/O Assignment

 Device            Control
 display
 printer
 rs232
 keyboard

 Disc Files        Control



STATUS: Simulated I/O Assignment ______________________________ 0:30

Control address? _01E00H

 <ADDRESS> ______ ______ ______ ______ ______ ______ ______
```

Figure 4-3. Simulate Display Control Address Assignment



```
Configuring I8080 processor in slot # 8. Memory slot # 7. Analysis slot # 9.

Simulated I/O Assignment

Device          Control
display          1E00
printer
rs232            1D00
keyboard

Disc Files      Control




STATUS: Simulated I/O Assignment ____________________________ 0:37

File name?_ DISC1:PRIME_

 <FILE>
```

Figure 4-4. Simulated Disc Files/File Name Assignment

Command File Designation

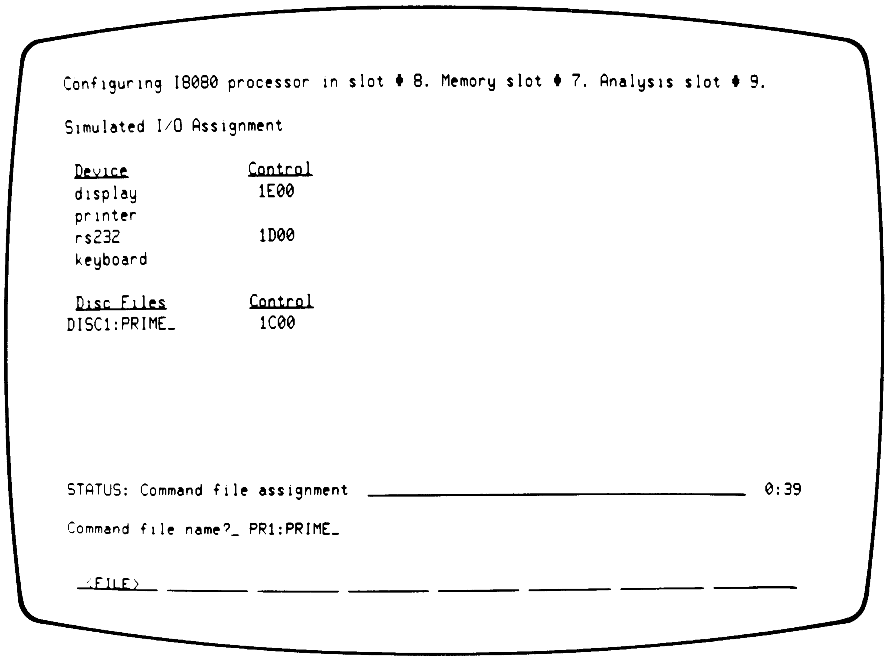
The emulation configuration is now complete. Information concerning the configuration questions answered during the above question and answer session is now ready for use during an emulation session. Rather than making it necessary to go through the question and answer session each time an emulation session is performed, the answers to emulation configuration questions can be stored on disc and called up during any future emulation session. Further, if it is desired to change some of the answers, the disc file can be edited instead of going through the complete question and answer session again.

At the end of the question and answer session, the following question is asked:

Command file name?

If it is desired to create a disc file of the emulation configuration, enter a file name from the keyboard and press RETURN. (See figure 4-5 for an illustration of the display.)

If no disc file is to be created, press RETURN without entering a file name.



```
Configuring I8080 processor in slot # 8. Memory slot # 7. Analysis slot # 9.

Simulated I/O Assignment

Device          Control
display          1E00
printer
rs232            1D00
keyboard

Disc Files      Control
DISC1:PRIME_     1C00




STATUS: Command file assignment  ____________________________  0:39

Command file name?_ PR1:PRIME_

<FILE>
```

Figure 4-5. Command File Name Assignment

Loading Absolute Code

Once emulation configuration is complete, the program code to be emulated must be loaded into user or emulation memory. Program code must be absolute code; that is, it must be relocated and linked to form a single module. The entire object code module is loaded as a single unit into memory from the system disc or tape. The only exception is the case in which the user system contains program code in ROM's that will be used by the emulated program.

Program code can be loaded into memory by using the load command described at the end of this chapter. The code will automatically be loaded following emulation configuration if the "load" option of the emulate command is selected. The portion of code in the address range specified as user memory is loaded into the target system, while the portion specified as emulation memory goes directly into emulation memory. The memory control board specifies the destination of the code by diverting the data in accordance with the memory map that was previously entered by the user. No specific loading instructions for the system are necessary because loading is automatically accomplished by the hardware.

Emulation Configuration Question-Answer Sequence Summary

The question-answer sequence required to define a new emulation configuration file is summarized in table 4-2.

The complete, detailed quetion-answer sequence is described in the preceding section entitled "Emulation Configuration Question-Answer Sequence".

Table 4-2. Emulation Configuration Question-Answer Sequence Summary

a. Press **emulate** or **edit_cnfg**

The following sub-questions are asked only if more than one type of emulation control board and/or more than one memory control board are installed in the 64000.

Emulation processor type?

Display indicates type(s) of emulation control board(s) installed in the system. Press the appropriate soft key.

Emulator card slot?

Display indicates the card slot(s) of the emulation control board(s) installed in the system. Press the appropriate soft key.

Memory controller slot?

Display indicates the card slot(s) of the memory control board(s) installed in the system. Press the appropriate soft key.

b. **Processor clock?** **internal** or **external**

If **external** : Is clock speed greater than x MHz?

yes or **no**

c. **Restrict processor to real time runs?** **yes** or **no**

d. **Stop processor on illegal op codes?** **yes** or **no**

Table 4-2. Emulation Configuration Question-Answer Sequence Summary (Cont'd)

NOTE

For 64000 systems without emulation memory, questions e and f are not asked.

e. **Emulation RAM address range?**	<ADDRESS> thru <ADDRESS>
	↓ ↓
	<ADDRESS> thru <ADDRESS>
f. **Emulation ROM address range?**	<ADDRESS> thru <ADDRESS>
	↓ ↓
	<ADDRESS> thru <ADDRESS>
g. **User RAM address range?**	<ADDRESS> thru <ADDRESS>
	↓ ↓
	<ADDRESS> thru <ADDRESS>
h. **User ROM address range?**	<ADDRESS> thru <ADDRESS>
	↓ ↓
	<ADDRESS> thru <ADDRESS>
i. **Illegal memory address range?**	<ADDRESS> thru <ADDRESS>
	↓ ↓
	<ADDRESS> thru <ADDRESS>
j. **Simulate I/O?** **yes** or **no**	
If **yes** :	
Simulate display? **yes** or **no**	
If **yes** : control address?	<ADDRESS>
Simulate printer? **yes** or **no**	
If **yes** : control address?	<ADDRESS>

Table 4-2. Emulation Configuration Question-Answer Sequence Summary (Cont'd)

Simulate rs232? yes or no	
If yes : control address?	<ADDRESS>
Simulate keyboard? yes or no	
If yes : control address?	<ADDRESS>
Simulate disc files? yes or no	
If yes : file name? <FILE>	
control address?	<ADDRESS>
file name? <FILE>	
control address?	<ADDRESS>
k. **Command file name?** <FILE>	
l. STATUS=xxxx ---- Ready or Program loaded	

Edit Emulation Configuration

An existing emulation configuration command file can be edited during an emulation session by pressing the edit_cnfg key. This action allows review and/or edit of the question-answer sequence defining the current command file.

NOTE

If the emulation configuration command file is changed during the command file review, the absolute code should be reloaded into memory before proceeding with emulation. (See load syntax at the end of this chapter.)

The edit configuration is initiated by pressing the edit_cnfg key, then RETURN. The 64000 responds with a display similar to that shown in figure 4-1. However, the questions are shown with the answers selected during the original question-answer session or a subsequent edit configuration session. Each answer may be left as it is or changed.

To leave an answer as it is, simply press RETURN without making an entry in answer to the current question.

To change an existing answer, either press the appropriate soft key or make a new keyboard entry as required by the current question.

When the review/edit session is completed, the following question is displayed: <CMDFILE> already exists, delete old?. The options displayed are **yes** and **no**.

If the existing file is not deleted, the changed version just created must be renamed. In this case, press RETURN.

When the filename question reappears, type in the name for the revised command file and press RETURN. Both command files (the old as well as the new) are now on disc and either can be called up for an emulation session.

If the old command file is no longer of any value and is to be deleted, press the **yes** key. The old command file is written over with the new answers. The file name remains the same as before.

Reload the absolute file if any changes were made to the emulation configuration. Refer to the load syntax at the end of this chapter for a description of the load options.

The system is now ready to start, or resume, emulation. The emulate and load command syntax is given on the following pages.

SYNTAX

```
         ┌                                                              ┐
         │ <CMDFILE> [load <ABSFILE>] [options  ⎧ edit     ⎫ ]          │
emulate  │                                      ⎩ continue ⎭            │
         │ load <ABSFILE>                                               │
         └                                                              ┘
```

Default Values

none

Examples:

emulate

emulate TEMP

emulate TEMP load EXPL options continue

emulate load EXPL

emulate TEMP options edit

FUNCTION

The emulate command initiates an emulation session using either an existing emulation command file or the emulation configuration question-answer sequence. It can also initiate the loading of an absolute file into either emulation or user memory.

Parameters

<CMDFILE>	<CMDFILE> is the file identifier (equivalent to <FILE>) of an existing emulation configuration command file. (An emulation configuration command file is a record of a previous emulation configuration question-answer session.)
<ABSFILE>	<ABSFILE> is file identifier (<FILE>) of an assembled (or compiled) and linked absolute file.
load	load is a key word which causes the specified <ABSFILE> to be loaded into emulation or user memory as specified by the emulation command file.
options	Pressing options causes the continue and edit soft keys to be displayed.
edit	The edit option allows a review and edit of the question-answer sequence of the specified <CMDFILE>. The answers can either be left as they were previously entered or changes can be entered for the current emulation session.
continue	The continue option allows reentry into an emulation session in which a user program is running. The execution of the user program is not affected by the reentry procedure.

DESCRIPTION

The emulate command is used to enter an emulation session and may be used to load an absolute file. When the command consists of only the word "emulate", the question-answer sequence which defines a new emulation configuration file is initiated. (Refer to the "Emulation Configuration" section of this chapter for more information.) After the question-answer sequence is completed, the record (i.e. copy) of this sequence is assigned a file identifier (<CMDFILE>).

When a previously defined <CMDFILE> is specified as a parameter for the emulate command, the command file having that name is used by the 64000 to automatically set up the current emulation configuration. This configuration can then either be used unchanged or it can be reviewed and edited, if required, using the edit option which is described later.

If load <ABSFILE> is entered, the absolute file identified by <ABSFILE> is loaded into emulation or user memory upon execution of the emulate command. The exact destination of the file is determined by the memory map which was set up during the question-answer session. If a <CMDFILE> is specfied prior to a load <ABSFILE> entry, the absolute file is loaded after the 64000 has been configured for emulation by the command file. When the specified <CMDFILE> is reviewed using "options edit" or when no <CMDFILE> is specified as a part of the emulate command, the loading occurs immediately following the current emulation configuration question-answer session.

Selecting options edit provides an opportunity to review and edit the specified <CMDFILE> before it is used to configure the development station for emulation. The review-edit procedure is described in the "Edit Emulation Configuration" section of this chapter.

A user program which is running during an emulation session will continue to run undisturbed if the session is terminated. Options continue allows an emulation session in which a user program is running to be reentered without disturbing the execution of the user program. If a "load <ABSFILE>" specification is included as part of the emulate command, only the load filename is logged, so that reference can be made to the symbol file. Memory is not disturbed.

SYNTAX

load <FILE>

Default Values

none

Example:

load_memory JR8085

FUNCTION

The load command transfers absolute code from the 64000 system memory into user RAM or emulation memory. The destination of the absolute code is determined by the memory configuration map which was set up during emulation configuration.

For example, if user RAM is mapped from 0000H through 2000H and if the absolute code in <FILE> occupies locations 0000H through 1500H, then the load_memory <FILE> command will load all of the absolute code into user RAM.

Parameters

<FILE>	<FILE> is the identifier of the absolute file to be loaded from the 64000 system memory into user RAM or emulation memory. The syntax requirements for <FILE> are discussed in Appendix A.

Chapter 5
Operational Commands

Introduction

The operational commands allow the user to control emulation. These commands consist of the following groups:

- Program control group: edit_cnfg, end, restart, run, step, and stop
- Display group: refer to chapter 6
- Analysis (trace and count) group: refer to chapter 7
- Memory management group: modify
- Record request group: copy and list

The syntax for each command in the groups listed above is described in this chapter except for the display and analysis groups. The commands for those groups are described in chapters 6 and 7, respectively.

Operational Command Syntax

The syntax listings on the following pages are intended to acquaint the user with the different operational commands. The syntactical variables used in this discussion are described in detail in appendix A.

SYNTAX

copy <ADDRESS>**[thru** <ADDRESS>**] to** <FILE>

Default Value

none

Examples:

copy 10A0H **to** TEMP1

copy 800H **thru** 20FFH **to** TEMP2

copy EXEC **thru** DONE **to** TEMP3

FUNCTION

The copy command is used to store the contents of specific memory locations in an absolute file on a disc without altering the contents of memory. Either a single memory location (<ADDRESS>) or a series of locations (<ADDRESS> thru <ADDRESS>) can be specified for transfer.

<FILE> determines the name under which the absolute file is to be stored. The copy command creates a new file having the specified name as long as there is no absolute file presently on the disc with that name. In the cases where a file represented by the <FILE> variable already exists, the system asks whether the old file is to be deleted. If the response is yes, the new file replaces the old one. If the response is no, then the copy command is cancelled and no copy is made.

Parameters

<ADDRESS>	<ADDRESS> determines the memory locations from which data is to be copied into the specified absolute file. The syntax requirements for <ADDRESS> are equivalent to those for <VALUE> as described in appendix A.
<FILE>	<FILE> is the identifier for the absolute file in which the data is to be stored. The syntax requirements for <FILE> are described in appendix A.

edit_cnfg

SYNTAX

edit_cnfg

Default Value

none

Example:

edit_configuration

FUNCTION

The edit_configuration command allows the question-answer sequence of the current emulation configuration to be reviewed and edited. Each of the configuration questions is presented with the response previously entered. The prior response can be entered as displayed by pressing RETURN , or modified as necessary and then entered by pressing RETURN .

end

SYNTAX

end

Default Value

none

Example:

end_emulation

FUNCTION

The end_emulation command teminates the current emulation session and returns the 64000 operating system to the monitor mode.

Execution of program code by the emulation processor is unaffected by the end command.

SYNTAX

$$\textbf{list} \begin{Bmatrix} \text{printer} \\ \text{<file>} \end{Bmatrix}$$

Default Value

none

Examples:

list_display_to printer

list_display_to TEMP1

FUNCTION

The list command produces a copy of the information currently displayed on the CRT. The copy can be either a listing file stored in the 64000 memory or a hardcopy produced by the printer. If the displayed page is written to an existing listing file, the old file is overwritten by the new information.

list appears on the display as "list_display_to".

Parameters

printer	printer causes a hardcopy of the single page currently displayed to be printed.
<FILE>	<FILE> causes the single page currently displayed to be copied to either a new or an existing file identified by <FILE>. The syntax for <FILE> is discussed in appendix A.

modify

SYNTAX

```
         ⎧ memory <ADDRESS> to <VALUE>[,<VALUE>]...              ⎫
modify   ⎨        <ADDRESS> thru <ADDRESS> to <VALUE>            ⎬
         ⎩ register (X) to <VALUE>[(X) to <VALUE>]...            ⎭
```

FUNCTION

The modify command is used to modify either the contents of memory or the contents of the processor registers. For the purpose of this discussion, the modify command options (modify memory and modify registers) are treated as separate commands and are described as such on the following pages.

modify memory

SYNTAX

```
               ⎧ <ADDRESS> to <VALUE>[,<VALUE>]...        ⎫
modify memory  ⎨                                          ⎬
               ⎩ <ADDRESS> thru <ADDRESS> to <VALUE>      ⎭
```

Default Value

none

Examples:

modify memory 800H to 10H

modify memory 910H to 0CH,56H,36H

modify memory 0A10H thru 0AFFH to 00

FUNCTION

The modify memory command can modify the contents of each memory location in a series to an individual value or the contents of all of the locations in a memory block to the same value.

A series of memory locations is modified by specifying the address of the first location in the series to be modified (<ADDRESS>) and the list of the <VALUE>s to which the contents of that location and the succeeding locations are to be changed. The first <VALUE> listed replaces the contents of the specified memory location, the second <VALUE> replaces the contents of the next location in the series, and so on until the list has been exhausted. If only one number or symbol is specified, only the single address indicated is modified. When more than one <VALUE> is listed, the <VALUE> representations must be separated by commas.

An entire block of memory can be modified such that the contents of each location in the block is changed to the single specified <VALUE>. This type of memory modification is achieved by entering the limits of the memory block to be modified (<ADDRESS> thru <ADDRESS>) and the <VALUE> to which the contents of all locations in the block are to be changed.

Parameters

<ADDRESS>	<ADDRESS> determines which memory location or series of locations are to be modified. The syntax for <ADDRESS> is equivalent to that for <VALUE> as described in appendix A.
<VALUE>	<VALUE> is the number which is to be loaded into the specified memory location or locations. The syntax for <VALUE> is described in appendix A.

modify registers

SYNTAX

modify register (X) to <VALUE>**[**(X) to <VALUE>**]**...

Default Value

none

Examples:

modify register a to 39H

modify register h to 0AH l to 50H a to 18H

FUNCTION

The modify register command is used to modify the contents of one or more of the microprocessor's internal registers. The entry for (X) determines which register is modified; the entry for <VALUE> is the number to which the contents of that register are changed.

Parameters

<VALUE>	<VALUE> is the number which is to be loaded into the specified processor register. The syntax for <VALUE> is described in appendix A.
(X)	(X) represents the name of one of the registers to be modified. The possible entries for (X) are listed as soft keys after register has been pressed.

restart

SYNTAX

restart

Default Value

none

Example:

restart_processor

FUNCTION

The restart command causes the emulation processor to go through a processor-dependent power_up sequence. The restart command does not modify any of the processor registers and does not restart a program run.

run

SYNTAX

```
run [from <ADDRESS>][until  { trc_cmplt                  }
                            { <ADDRESS>         ]        }
                            { <TERM>[or <TERM>]...       }
```

Default Values

from <ADDRESS>	If the from <ADDRESS> option is omitted, the emulator will begin program execution at the current address specified by the processor's next program counter.
until	If the until option is omitted, the processor will continually execute program instructions until halted by a stop run command.

Examples:

run

run from 810H

run from 810H **until** address = 84AH data = 60H

FUNCTION

The run command controls the execution of the emulation program. The program can either be run from a specified <ADDRESS> or from the address currently stored in the processor's next program counter. In addition, if the until option is selected, the run can be signaled to terminate when a trace is completed, a particular <ADDRESS> is reached, or a specified state or series of states appears on the emulation buses.

Parameters

<ADDRESS>	<ADDRESS> represents a state on the address bus which can be used to start or stop a program run. The syntax requirements for <ADDRESSS> are equivalent to those for <VALUE> as defined in appendix A.
from	from is used to specify the address from which the emulation processor will start program execution.
<TERM>	<TERM> specifies a state or series of states which cause termination of the program run when they appear on the emulation buses. The syntax for <TERM> is described in appendix A.
trc_cmplt	trace_complete causes the emulation program run to be terminated when the trace specification is satisfied and "trace complete" is displayed.
until	until is used to specify the state or status on the emulation bus which will terminate the program run.

SYNTAX

step [<# STEPS>]**[from** <ADDRESS>**]**

Default Values

<# STEPS>	If no value is entered for number of times, only one instruction is executed each time the RETURN key is pressed. However, multiple instructions can be executed by holding down RETURN key.
from <ADDRESS>	If the from <ADDRESS> option is omitted, stepping begins at the next program counter address.

Examples:

step

step from 810H

step 20 **from** 0A40H

FUNCTION

The step command allows program instructions to be sequentially analyzed by causing the emulation processor to execute a specified number of instructions. The contents of the processor registers are displayed after each instruction is executed and the contents of memory can be displayed upon completion of the step command.

Parameters

<# STEPS>	<# STEPS> determines how many instuctions will be executed by the step command.
from	The from parameter is used to specify the address from which the emulation processor will step.
<ADDRESS>	<ADDRESS> represents the address from which stepping will start. The syntax for <ADDRESS> is equivalent to the syntax for <VALUE> as described in appendix A.

SYNTAX

$$\textbf{stop} \quad \left\{ \begin{array}{l} \textbf{run} \\ \textbf{trace} \end{array} \right\}$$

Default Value

none

Examples:

stop run

stop trace

FUNCTION

The stop command terminates either the current program run or trace command.

Parameters

run	The run parameter stops the execution of a run command.
trace	The trace parameter stops the execution of the trace command. That is, the system stops searching for trigger and trace states.

Chapter 6 Display Commands

Display Command Capabilities

There are four basic types of information which may be viewed by using the display commands. These are:

- Memory and register data.
- Trace and run specifications previously entered by the operator.
- Absolute or relative counts based upon the trace specifications.
- Global and local symbols.

Memory and Register Data

Data may be accessed and displayed from memory, microprocessor registers, or the trace buffer. Previously entered trace and run specifications and program symbols can also be displayed.

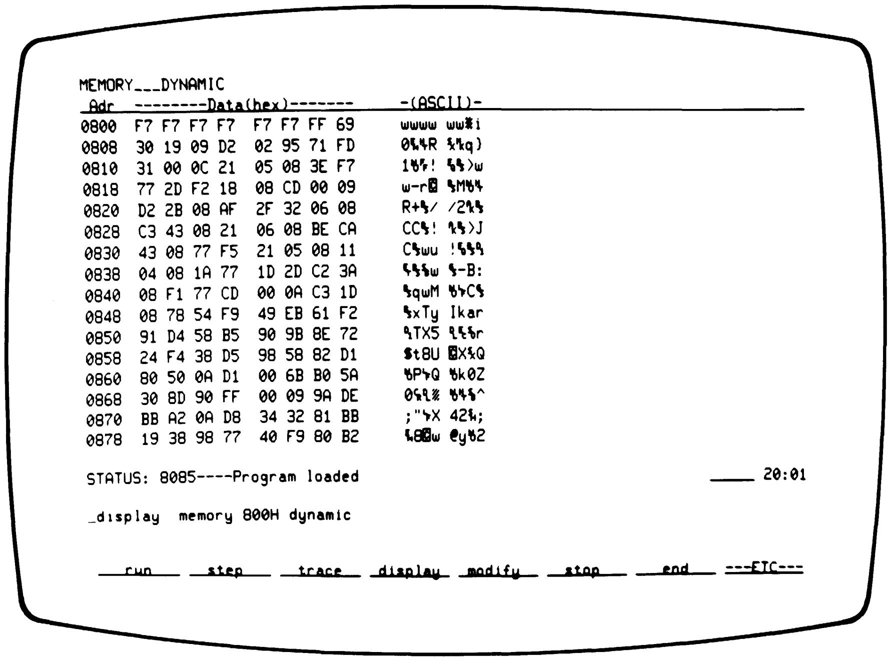
Memory Data. For data taken from memory, the starting address in memory may be specified. Whether the data comes from emulation or user memory depends upon the memory map assignments made during the configuration of the emulation command file. Unless otherwise specified by the user, memory data is displayed statically. (The static display shows the memory contents existing when the display command is executed.) The data is displayed in hexadecimal form with the corresponding ASCII characters as shown in figure 6-1.

The user may modify the display command so that the memory data is displayed using one, two, or all three of the following techniques:

a. Data may be viewed dynamically which causes the display to be continuously updated. This is useful if the data in the memory is continually changing. However, the display is not updated in real time.

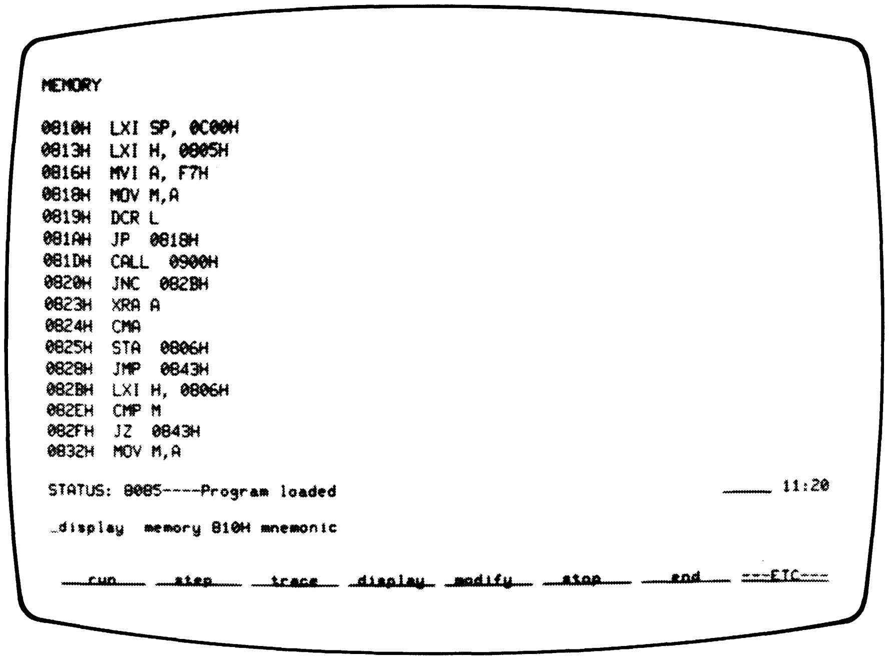
b. Data may be viewed in mnemonic form rather than in hexadecimal form as shown in figure 6-2. However, it is advisable to use a form consistent with the data being displayed. For instance, it makes sense to display memory containing program code in mnemonic form, but mnemonic form does not make sense for viewing memory locations containing arithmetic values.

c. Memory addresses may be displayed "offset" from the actual value. The address offset allows the actual addresses to be offset by a value specified by the user. If the value is correctly chosen*, the address space displayed will start at location 0000H and will correspond to the listing generated by the assembler program. (*When absolute files are linked by the relocating linker program, program modules are combined and may be relocated to produce a program. If a module, orginating at address X is linked with other modules, it may be assigned a new starting address X+Y where Y is a value that depends on the number and size of the other modules being linked. Offset, therefore, allows the user to subtract "Y" so that the addresses appear the same as in the assembly listing.)



```
MEMORY___DYNAMIC
 Adr  --------Data(hex)-------      -(ASCII)-
0800  F7 F7 F7 F7  F7 F7 FF 69      wwww ww*i
0808  30 19 09 D2  02 95 71 FD      0▒▒R ▒▒q}
0810  31 00 0C 21  05 08 3E F7      1▒▒! ▒▒>w
0818  77 2D F2 18  08 CD 00 09      w-r█ ▒M▒▒
0820  D2 2B 08 AF  2F 32 06 08      R+▒/ /2▒▒
0828  C3 43 08 21  06 08 BE CA      CC▒! ▒▒>J
0830  43 08 77 F5  21 05 08 11      C▒wu !▒▒▒
0838  04 08 1A 77  1D 2D C2 3A      ▒▒▒w ▒-B:
0840  08 F1 77 CD  00 0A C3 1D      ▒qwM ▒▒C▒
0848  08 78 54 F9  49 EB 61 F2      ▒xTy Ikar
0850  91 D4 58 B5  90 9B 8E 72      ▒TX5 ▒▒▒r
0858  24 F4 38 D5  98 58 82 D1      $t8U █X▒Q
0860  80 50 0A D1  00 6B B0 5A      ▒P▒Q ▒k0Z
0868  30 8D 90 FF  00 09 9A DE      0▒▒▒ ▒▒▒^
0870  BB A2 0A D8  34 32 81 BB      ;"▒X 42▒;
0878  19 38 98 77  40 F9 80 B2      ▒8█w @y▒2

STATUS: 8085----Program loaded                          ____ 20:01

_display  memory 800H dynamic
```

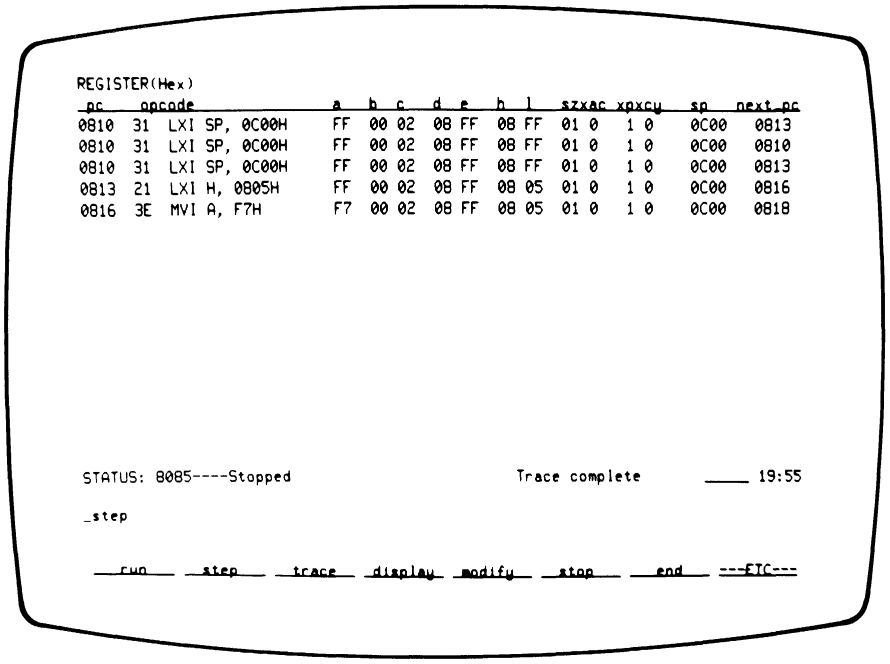
Figure 6-1. Display Memory 800H Dynamic



```
MEMORY

0810H  LXI SP, 0C00H
0813H  LXI H, 0805H
0816H  MVI A, F7H
0818H  MOV M,A
0819H  DCR L
081AH  JP  0818H
081DH  CALL  0900H
0820H  JNC  082BH
0823H  XRA A
0824H  CMA
0825H  STA  0806H
0828H  JMP  0843H
082BH  LXI H, 0806H
082EH  CMP M
082FH  JZ  0843H
0832H  MOV M,A

STATUS: 8085----Program loaded                                  ____ 11:20

_display  memory 810H mnemonic

  run    step    trace   display  modify   stop    end    ---ETC---
```

Figure 6-2. Display Memory 810H Mnemonic

Register Data. Unless otherwise specified by the user, register data is displayed statically as shown in figure 6-3. Register data may also be displayed dynamically and the program counter (PC) value can be offset by a value specified by the user. The dynamic display and offset are done for the same reason as described above for memory data. The display of register data is processor dependent and defaults to the mnemonic form. An example of a register data display is shown in figure 6-3.



```
REGISTER(Hex)
 pc    opcode                 a   b  c   d  e   h  l   szxac xpxcy  sp    next_pc
0810  31  LXI SP, 0C00H      FF  00 02  08 FF  08 FF  01 0    1 0  0C00   0813
0810  31  LXI SP, 0C00H      FF  00 02  08 FF  08 FF  01 0    1 0  0C00   0810
0810  31  LXI SP, 0C00H      FF  00 02  08 FF  08 FF  01 0    1 0  0C00   0813
0813  21  LXI H, 0805H       FF  00 02  08 FF  08 05  01 0    1 0  0C00   0816
0816  3E  MVI A, F7H         F7  00 02  08 FF  08 05  01 0    1 0  0C00   0818




STATUS: 8085----Stopped                      Trace complete       ____ 19:55

_step

 ___run___ ___step___ ___trace___ _display_ _modify_ ___stop___ ___end___ ---ETC---
```

Figure 6-3. Display Registers

Trace Data. The trace data stored in the trace buffer as a result of a trace command may also be displayed using the display command. Refer to Chapter 7 for a description of the trace display. The trace data is normally displayed in mnemonic form with the operators and operands packed onto a single line and with the actual absolute addresses shown.

Trace and Run Specifications

The trace and run specifications are stored by the system each time they are entered. This display command may be used to recall the last run or trace command. In the event that no run or trace specification has been made, the display command will show a blank display area.

Absolute/Relative Counts

Counts are displayed only if the 64000 system is equipped with the optional analysis board. The absolute counts (time or state) are normally displayed with the trace data. The absolute count may be changed to relative count by the display command. Refer to Chapter 7 for a complete description of relative and absolute, time and state counts.

Global and Local Symbols

Global and local symbols can be veiwed on the display. Local symbols are symbols defined in the source file for a single program module. Global symbols are those declared to be global in the source file. They are defined using the assembler psuedo instruction, GBL, or by the Pascal GLOBVAR directive. When the display command is used to examine either of these symbol types, the display will contain the symbol name and its present value.

Display Command Syntax

The display command syntactical composition is summarized on the following pages. The commands are listed in the same order as displayed by the 64000 system soft keys.

SYNTAX

```
         ⎧ count  { absolute | relative }
         ⎪ global
         ⎪ local    <FILE>
display  ⎨ memory[<ADDRESS>][dynamic][mnemonic] [offset_by<ADDRESS>]
         ⎪ registers [dynamic][offset_by <ADDRESS>]
         ⎪ run_spec
         ⎪ trace [absolute][unpacked][offset_by <ADDRESS>]
         ⎩ trc_spec
```

FUNCTION

The display command initiates the display of the time or state counts, local or global symbols, the contents of registers or memory, the current run or trace specification, or the contents of the trace buffer. For the purpose of this discussion, the display command options (display count, display global_symbols, display memory, ...etc.) are treated as separate commands and are described as such on the following pages.

display count

SYNTAX

$$\textbf{display count} \begin{Bmatrix} \text{absolute} \\ \text{relative} \end{Bmatrix}$$

Default Value

none

Examples:

display count absolute

display count relative

FUNCTION

The display count command is used after a trace has been obtained to change the current display of time or state counts to one in which the counts are displayed either relative to the previous event or as an absolute count measured from the trigger event. If time counts are currently displayed, the display count command causes an absolute or relative time count to be displayed. If the current display contains state counts, a relative or absolute state count results.

Parameters

absolute	absolute causes the state or time count for each event of the trace to be displayed as the total count measured from the trigger event.
relative	relative causes the state or time count for each event of the trace to be displayed as the count measured relative to the previous event.

SYNTAX

display global

Default Value

none

Example:

display global_symbols

FUNCTION

The display global_symbols command displays the global symbols defined for the current absolute file and the present values of those symbols as listed in the global symbol table.

Parameter

global	global represents the symbols and labels defined as global in the source program from which the current absolute file was generated. **global** appears on the screen as "global_symbols".

SYNTAX

display local <FILE>

Default Value

none

Example:

display local_symbols_in TEMP1

FUNCTION

The display local_symbols command displays the local symbols and their current values as defined in the source file identified by the syntactical variable <FILE>.

Parameters

local	local refers to the symbols and labels in the source file identified by <FILE>. **local** appears on the screen as "local_symbols_in".
<FILE>	<FILE> represents the source file name that contains the local symbols to be displayed. Refer to Appendix A for the syntax requirements of <FILE>.

SYNTAX

display memory[<ADDRESS>**][**dynamic**][**mnemonic**][**offset_by <ADDRESS>**]**

Default Values

<ADDRESS>	-	The default value for <ADDRESS> is 0000.
dynamic	-	If dynamic is omitted, the display is static and is not updated as the contents of memory change.
mnemonic	-	If mnemonic is omitted, the memory contents are displayed in hexadecimal form.
offset_by	-	If offset_by is omitted, the actual memory addresses are displayed.

Examples:

display memory 800H mnemonic

display memory dynamic mnemonic

display memory 810H dynamic offset_by 810H

FUNCTION

The display memory command displays the contents of the specified memory location or series of locations. The memory contents can be viewed either statically or dynamically and either in mnemonic or hexadecimal form. In addition, the memory addresses can be displayed offset by a value which allows the information to be easily compared to the absolute file listing.

Parameters

<ADDRESS>	<ADDRESS> as it is first listed in the syntax represents the address at which the memory display begins. In the second occurrence, it is the address by which displayed memory addresses are offset from the actual address values. The syntax requirements for <ADDRESS> are equivalent to those for <VALUE> as described in Appendix A.
dynamic	dynamic causes the display to be periodically updated with the current contents of memory. The program may be interrupted in order to fetch the memory data and update the display. Therefore, "pseudo run" is displayed on the STATUS line of the display to indicate that this is a nonreal-time operation.
mnemonic	mnemonic causes the mnemonics for the opcodes stored in the specified memory locations to be displayed.
offset_by	offset_by causes the system to subtract the specified <ADDRESS> from each of the actual absolute addresses before the addresses and the corresponding memory contents are displayed. The value of <ADDRESS> can be selected such that each module in a program appears to start at address 0000. The display of the memory contents will then appear similar to the assembly listing.

SYNTAX

display registers [dynamic][offset_by <ADDRESS>]

Default Values

dynamic	- If dynamic is omitted, the display is static and is not updated as the contents of the registers change.
offset_by	- If offset_by is omitted, the actual values of the program counter are displayed.

Examples:

display registers

display registers offset_by 810H

display registers dynamic offset_by 0A10H

FUNCTION

The display registers command displays the current contents of the program counter, the mnemonic of the opcode presently being executed, and the current contents of the processor's registers. This process does not occur in real time; therefore, if the registers are to be displayed while the processor is running, the system must be configured to allow nonreal-time operations.

The registers can be displayed either statically or dynamically. In addition, the displayed value of the program counter can be offset from the actual value by a number which allows the register information to be easily compared with the absolute file listing.

Parameters

dynamic	dynamic causes the display of the register data to be continuously updated. The register data is sampled periodically and the last 16 states to be sampled are displayed on the screen. The displayed states are not necessarily the last 16 states executed by the processor. The term "pseudo run" is displayed on the command line to indicate that this is a nonreal-time operation.
offset_by	offset_by causes the system to subtract the specified <ADDRESS> from the current value of the program counter to arrive at the value which is displayed on the screen. The actual value of the program counter remains unchanged.
<ADDRESS>	<ADDRESS> represents the value by which the displayed program counter address is offset from the actual program counter address. The syntax for <ADDRESS> is equivalent to the syntax for <VALUE> as described in Appendix A.

SYNTAX

display run_spec

Default Value

none

Example:

display run_specification

FUNCTION

The display run_specification displays the last active run command. Run command specifications are described in Chapter 5.

SYNTAX

display trace [absolute]**[**unpacked**][**offset_by <ADDRESS>**]**

Default Values

absolute	-	If absolute is omitted, the data is displayed in opcode mnemonic form.
unpacked	-	If unpacked is omitted, the operators and operands are displayed together on a single line. For example, STA 8200H would be displayed as: STA 8200H
offset_by	-	If offset_by is omitted, the actual addresses are displayed.

Examples:

display trace

display trace absolute

display trace unpacked offset_by 900H

FUNCTION

The display trace command displays the contents of the trace buffer. The information can be presented as absolute hexadecimal code or in mnemonic form and can follow either a packed or unpacked format. If the unpacked format is selected, the trace data can only be displayed in single column form.

The offset_by option causes the system to subtract the specified <ADDRESS> from the addresses of the executed instructions before the trace is displayed. With an appropriate entry for <ADDRESS>, each instruction in the displayed trace will appear as it does in the assembled program listing.

Parameters

absolute	absolute directs the system to display the trace information in absolute hexadecimal format rather than in mnemonic format.
unpacked	unpacked specifies a format in which the operators and operands are displayed on consecutive lines and in absolute hexadecimal code. For example, the command STA 8200H would appear as: 32H (opcode for STA) 00H 82H
offset_by	offset_by causes the system to subtract the specified <ADDRESS> from each of the actual absolute addresses before the trace data is displayed.
<ADDRESS>	<ADDRESS> represents the number by which the address displayed for an executed instruction is offset from the actual address of the instruction. The syntax for <ADDRESS> is equivalent to the syntax for <VALUE> as described in Appendix A.

SYNTAX

display trc_spec

Default Value

none

Example:

display trace_specification

FUNCTION

The display trace_specification displays the last active trace command. Trace specifications are described in Chapter 7.

Chapter 7
Analysis Commands

Introduction

The Analysis Board

The 64000 system has an optional Analysis board for real-time analysis of the emulation processor. The Analysis board is a logic state analyzer which uses the system CRT as its display. Trace specifications are entered into the emulation system software by the user and are loaded into the Analysis board trigger hardware by the emulation system via the host processor bus. Once a trace has been requested by the emulation system, the analyzer begins to look for "states" that satisfy the trace specification until stopped by a command from the user or until the trace specification is met.

The analyzer contains a trace buffer (memory) that stores 256 events of the trace. The analyzer monitors the address, data, and status and control lines of the emulation bus and stores the states indicated by the trace specification in a dedicated analysis memory. When the required trigger occurs on the emulation bus and all of the specified states have been gathered, the emulation system accesses the analysis memory via the host processor bus and the contents of trace memory are retrieved, formatted, and displayed on the CRT screen. If desired, the emulation system will disassemble the code found in the trace memory into the mnemonics of the processor being emulated.

Real-time Analysis

Although the analyzer operates in conjunction with the emulator, it is a separate function that is optional to the emulation system. It is the analyzer that allows for real-time analysis of emulation bus activity.

Real-time analysis is the ability to monitor address, data, and status and control lines of the target processor for proper execution with software and hardware. As in the case of real-time emulation, real-time means full speed operation in the range of the target processor.

If the Analysis board is absent from the emulator configuration, the emulator can still monitor the emulation bus, but not in real time. This can increase analysis time substantially because the analysis functions are not passive when there is no Analysis board in the system.

Certain logic analyzer operations require intervention on processor operation. As an example, most processors do not allow external access of register contents. Therefore, the ability to interrogate the processor registers requires gaining control or stopping the processor and causing the processor to output its register contents. Real-time processor operation may have to be suspended in other trace operations in order to obtain register information. Memory read operations in real time are not allowed by some processors due to timing implementation of processor operation. In these cases the processor must have "wait" states inserted in order to allow a memory read operation by the emulator analyzer.

Analysis Commands

There are two commands used in analysis. They are the trace command (real-time and nonreal-time analysis) and the count command (real-time analysis only). The trace command allows the user to specify a particular part of a program where execution is to be traced and displayed. The count command can be used in combination with the trace command to count time or states of the program trace.

Trace Command

The trace command allows the user to specify the particular part of a program that is to be measured and displayed. The trace measurements may be made once and displayed statically, or the same measurements may be made repetitively and the results continuously updated.

In general, the trace command causes 256 states to be measured and stored in the trace buffer. However, the number of lines displayed may be less than 256 since one line may be made up of several states. The sequential states are displayed in a time progression order. (Note: The "PREV PAGE" or "NEXT PAGE" keys can be used to view all measured states a page at a time, or, if desired, the "ROLL UP" or "ROLL DOWN" keys can be used to view the measured states a line at a time.)

Trace Command Capabilities

The trace command allows the user to specify any one, all, or none of the following. (If none are specified; i.e., trace entered alone, then the default conditions described in the following steps are automatically used.)

a. The type of "states" to be placed into the trace buffer may be specified. This is done with the "trace only <TERM>**[**or <TERM>**]**...".

If a "trace only <TERM>[or <TERM>]..." is not specified, the sequential program states on the emulation bus (address, data, status) that are routinely executed are placed into the trace buffer. The qualified states (or all states if not qualified by trace only) are continuously placed into the trace buffer until the trigger occurs. When the trigger occurs, the 64000 determines whether the trace measurement is or is not complete depending upon the trigger <POSITION> as described in subparagraph b, below.

b. The program "states" that trigger the saving of the trace buffer contents and the time relationship of the trace buffer contents with respect to the trigger may also be specified. This is specified as: "Trigger <POSITION><TERM>[or <TERM>]...".

   The trigger <POSITION> is used to save the states that occur either "before", "after", or "about" the trigger.

   The trigger <TERM>[or <TERM>]... is used to specify the states in the program which cause the 64000 to determine if the trace measurements are or are not complete (depending upon the trigger <POSITION>).

   If the trigger <POSITION><TERM>[or <TERM>]... is not specified , the states that occur immediately following the initiation of the trace command are saved in the trace buffer. Thus, for the default case, the trigger <POSITION> is "after" and the trigger <TERM> [or <TERM>]... is "don't care".

c. A pretrigger sequence that must precede the trigger may also be specified. Additionally, those <TERM>s that must not precede the trigger in the pretrigger sequence may also be specified.

   The pretrigger sequence is specified as "trace in sequence <TERM>[or <TERM>]... then <TERM>[or <TERM>]...", etc. The states that must not occur in the pretrigger sequence are specified as "restart on <TERM>[or <TERM>]...". The "restart" can only be specified as part of the "trace in sequence" series. If a "restart <TERM>" is encountered, the 64000 restarts the "trace in sequence" series from the very beginning.

d. The trace measurements may be made repetitively and the results continually updated. This is specified as "continue".

Trace Command Descriptions

From the user's point of view, the trace command has four basic forms which are summarized in the following paragraphs. The descriptions progress from the simplest form of trace to the most complex form and should be read in the order presented. (See figure 7-1 for trace command flow diagram.)

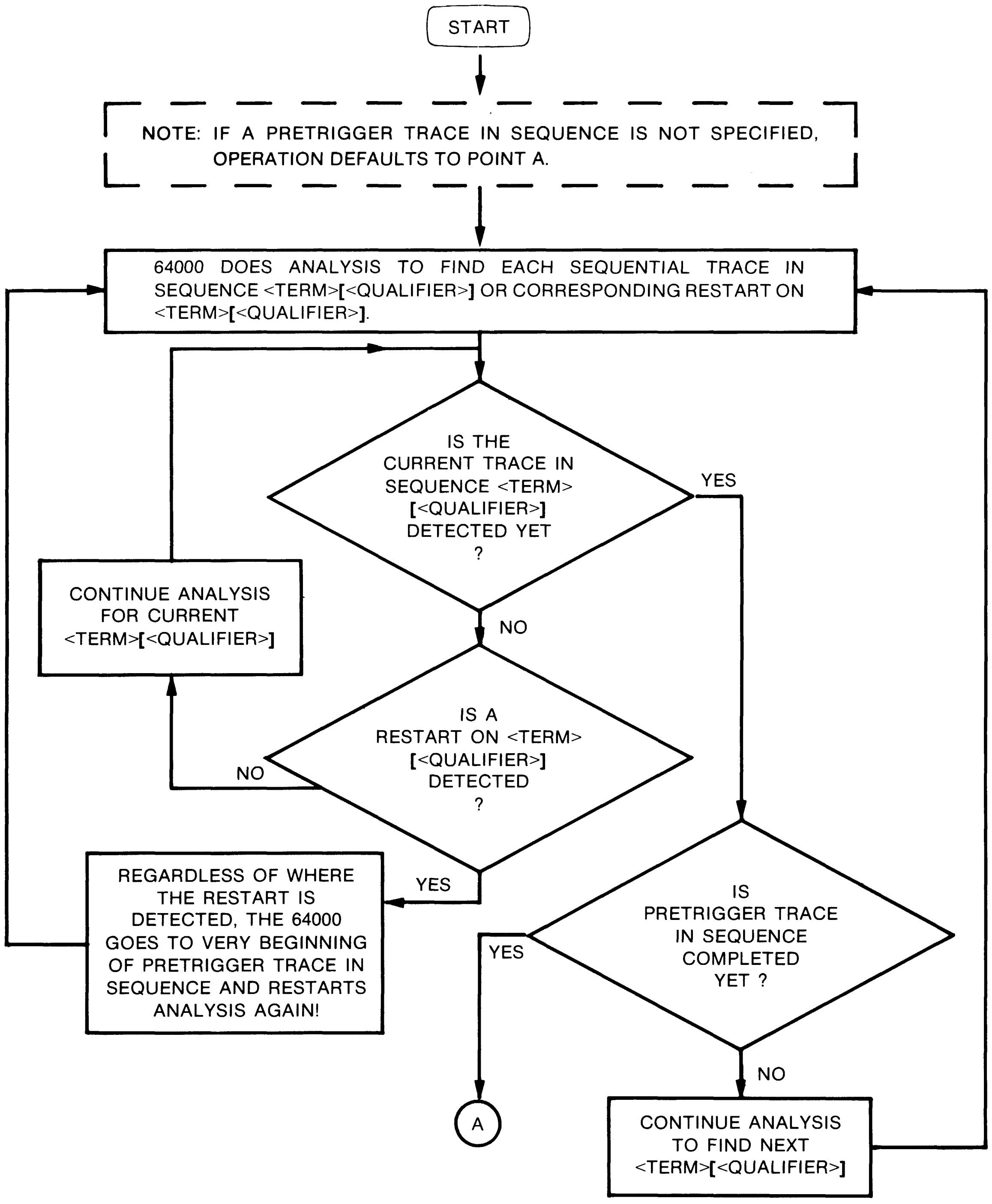

START
NOTE: IF A PRETRIGGER TRACE IN SEQUENCE IS NOT SPECIFIED, OPERATION DEFAULTS TO POINT A.
64000 DOES ANALYSIS TO FIND EACH SEQUENTIAL TRACE IN SEQUENCE <TERM>[<QUALIFIER>] OR CORRESPONDING RESTART ON <TERM>[<QUALIFIER>].
IS THE CURRENT TRACE IN SEQUENCE <TERM> [<QUALIFIER>] DETECTED YET ?
YES
NO
CONTINUE ANALYSIS FOR CURRENT <TERM>[<QUALIFIER>]
IS A RESTART ON <TERM> [<QUALIFIER>] DETECTED ?
NO
YES
REGARDLESS OF WHERE THE RESTART IS DETECTED, THE 64000 GOES TO VERY BEGINNING OF PRETRIGGER TRACE IN SEQUENCE AND RESTARTS ANALYSIS AGAIN!
IS PRETRIGGER TRACE IN SEQUENCE COMPLETED YET ?
YES
NO
A
CONTINUE ANALYSIS TO FIND NEXT <TERM>[<QUALIFIER>]

Figure 7-1. Trace Command Flow Diagram

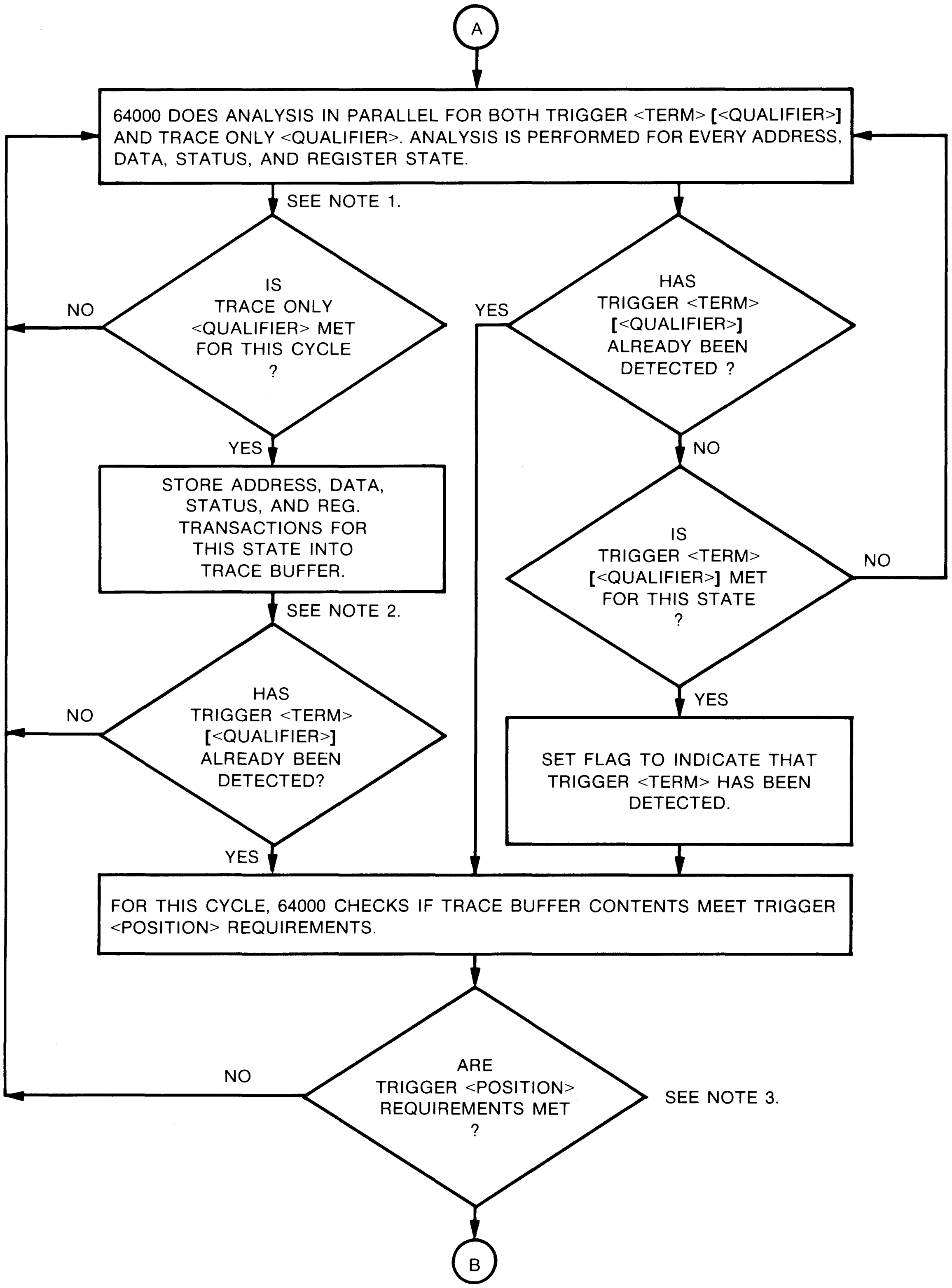
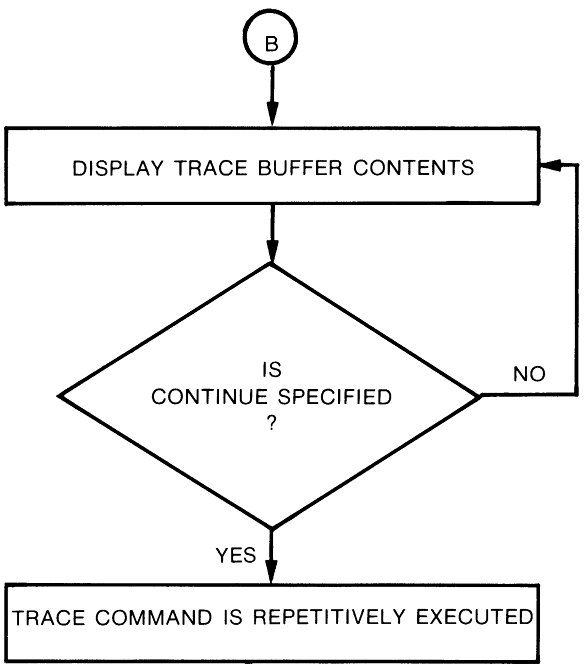

A
64000 DOES ANALYSIS IN PARALLEL FOR BOTH TRIGGER <TERM> [<QUALIFIER>] AND TRACE ONLY <QUALIFIER>. ANALYSIS IS PERFORMED FOR EVERY ADDRESS, DATA, STATUS, AND REGISTER STATE.
SEE NOTE 1.
IS TRACE ONLY <QUALIFIER> MET FOR THIS CYCLE ?
NO
YES
STORE ADDRESS, DATA, STATUS, AND REG. TRANSACTIONS FOR THIS STATE INTO TRACE BUFFER.
SEE NOTE 2.
HAS TRIGGER <TERM> [<QUALIFIER>] ALREADY BEEN DETECTED?
NO
YES
HAS TRIGGER <TERM> [<QUALIFIER>] ALREADY BEEN DETECTED ?
YES
NO
IS TRIGGER <TERM> [<QUALIFIER>] MET FOR THIS STATE ?
NO
YES
SET FLAG TO INDICATE THAT TRIGGER <TERM> HAS BEEN DETECTED.
FOR THIS CYCLE, 64000 CHECKS IF TRACE BUFFER CONTENTS MEET TRIGGER <POSITION> REQUIREMENTS.
ARE TRIGGER <POSITION> REQUIREMENTS MET ?
NO
SEE NOTE 3.
B

Figure 7-1. Trace Command Flow Diagram (Cont'd)


B
DISPLAY TRACE BUFFER CONTENTS
IS CONTINUE SPECIFIED ?
NO
YES
TRACE COMMAND IS REPETITIVELY EXECUTED

Notes:

1. If trace only <QUALIFIER> is not specified, operation defaults to the "yes" path.

2. If trigger <TERM>**[**<QUALIFIER>**]** is not specified, operation defaults to the "yes" path.

3. If trigger <POSITION> is not specified, <POSITION> defaults to "after".

Figure 7-1. Trace Command Flow Diagram (Cont'd)

Trace [continue]

This is the simplest form of the trace command. When trace is specified alone, the following occurs. Since there is neither a pretrigger "trace-in sequence" nor a "trigger <TERM> **[**or <TERM>**]**..." specified, the storing of states in the trace buffer begins at whatever address happens to be in the program counter when trace is initiated. Also, because there is no "trigger <POSITION> <TERM>**[**or <TERM>**]**..." specified, the sequential program states on the emulation bus (address, data, status) that are routinely executed after trace is initiated are saved in the trace buffer.

The "trace **[**continue**]**" command causes the trace to be repetitively executed and the displayed results to be continuously updated. The trace continues until it is manually stopped by the stop-trace soft keys. However, because there are no parameters specified in this form of trace, the trigger will probably vary for each repetitive sequence.

Trace only <TERM>[or <TERM]...[continue]

This form of trace is similar to "trace **[**continue**]** in that neither the pretrigger "trace in sequence" nor a "trigger <POSITION><TERM>**[**or <TERM>**]**..." is specified by the user. Thus, the trigger position defaults to "after" and the collection of the qualified trace states begin at whatever address exists in the program counter when the trace command is initiated. However, the difference is that only those states that meet the "trace only <TERM>**[**or <TERM>**]**..." conditions are placed into the trace buffer.

Trace (trigger) <POSITION><TERM>[or <TERM>]... [trace only <TERM> [or <TERM>]]...[continue]

This form of trace is similar to "trace only <TERM>**[**or <TERM>**]**...**[**continue**]**" in that the user specifies what states are to be stored in the trace buffer. The difference is that the "trigger <POSITION><TERM>**[**or <TERM>**]**... is also specified by the user.

The <TERM>**[**or <TERM>**]**... defines the state or states at which the trigger occurs. (When the trigger states are detected, the 64000 determines whether the trace measurements are or are not complete depending upon trigger <POSITION>.)

Trace in_sequence...Restart_on...

This prefix is used to define the sequential states that must precede, and those that must not precede the trace trigger...and trace only...parameters. This is a useful tool for checking program branches, loops, etc.

Trace Display

The trace data stored in the trace buffer as a result of a "trace" command can be displayed with the "display trace" command. The default condition for the display trace command is the mnemonic form with the operators and operands packed onto a single line along with the absolute addresses.

The "display" command may be used to specify one or more of the following: data displayed in absolute (hexadecimal) format; data displayed in mnemonic form; data with operators and operands unpacked and on separate lines ; and addresses displayed with an offset value. See figures 7-2 through 7-5 for examples of the display trace formats.

```
TRACE                                                               COUNT TIME    ABSOLUTE
    ADDRESS,DATA,STATUS
AFTER 009AH CALL  00C4H   sp-1 1083H (sp-1,sp-2) 009DH                    +  0.         US
+001  00C4H CALL  0159H   sp-1 1081H (sp-1,sp-2) 00C7H                    +  5.         US
+002  0159H PUSH PSW      sp-1 107FH a 03H flag 04H                       + 11.         US
+003  0:5AH DCX H                                                         + 15.         US
+004  015BH MVI A, F5H                                                    + 17.         US
+005  015DH MOV A,H                                                       + 19.         US
   .
   .
   .
```

Figure 7-2. Display Trace Format

```
TRACE                                                               COUNT TIME    ABSOLUTE
    ADDRESS,DATA,STATUS
AFTER  009AH CDH 07H                                                      +  0.         US
+001   009BH C4H 06H                                                      +  1.         US
+002   009CH 00H 06H                                                      +  2.         US
+003   1083H 00H 15H                                                      +  3.         US
+004   1082H 9DH 15H                                                      +  4.         US
+005   00C4H CDH 07H                                                      +  5.         US
   .
   .
   .
```

Figure 7-3. Display Trace Absolute Unpacked Format

```
TRACE                                                               COUNT TIME    ABSOLUTE
    ADDRESS,DATA,STATUS
AFTER 009AH CDH 07H  009BH C4H 06H  009CH 00H 06H  1083H 00H 15H          +  0.         US
                     1082H 9DH 15H
+001  00C4H CDH 07H  00C5H 59H 06H  00C6H 01H 06H  1081H 00H 15H          +  5.         US
                     1080H C7H 15H
+002  0159H F5H 07H  107FH 03H 15H  107EH 04H 15H                         + 11.         US
+003  015AH 2BH 07H                                                       + 15.         US
   .
   .
   .
```

Figure 7-4. Display Trace Packed Format

```
TRACE                                                                   COUNT TIME   ABSOLUTE
    ADDRESS,DATA,STATUS         ADDRESSES OFFSET BY 009AH
AFTER  0000H CALL  002AH        sp-1 0FE9H (sp-1,sp-2) 0003H                 +  0.         US
+001   002AH CALL  00BFH        sp-1 0FE7H (sp-1,sp-2) 002DH                 +  5.         US
+002   00BFH PUSH PSW           sp-1 107FH a 03H flag 04H                    + 11.         US
+003   00C0H DCX H                                                           + 15.         US
+004   00C1H MVI A, F5H                                                      + 17.         US
+005   00C3H MOV A,H                                                         + 19.         US
  .
  .
  .
```

Figure 7-5. Format for Display Trace Offset by 009AH

In some trace displays and memory displays, a string of asterisks (****) will be present in a display list. When displaying a trace, the emulator does not test for read or write operations, but simply looks for the beginning of a cycle (the occurrence of an opcode or an interrupt). After detecting a beginning of a cycle in mnemonic display mode, the emulator then searches for the next beginning of cycle in the trace list. If the next beginning of cycle is not detected within a specified number of states (processor dependent), "****" is displayed, indicating that the emulator was unable to interpret the operand of the current opcode. This condition can be caused by an error in software execution or by qualifying a trace. In memory displays, asterisks are displayed when an attempt is made to display unmapped memory locations.

Count Command

The count command may be used to count any of the following parameters:

- count time absolute
- count time relative
- count <STATES> absolute
- count <STATES> relative

Each of the above parameters are described in the following paragraphs.

Count Time Absolute

Absolute time counts are automatically displayed after a trace command is issued and when the states stored in the trace buffer are displayed.

The absolute time count is the total elapsed time between the trigger and a measured state. If a minus time is shown, then the states in the line occurred before the trigger and the elapsed time is measured from the beginning of the state until trigger detection.

Count Time Relative

Absolute time counts may be replaced with relative time counts by pressing the "display count relative" soft keys. Once relative time counts are selected, they are displayed until "display count absolute" is selected.

The relative time measures the interval from the beginning of the previous state (line) to the beginning of the currently displayed line. The relative time count does not use the trigger as a reference point. This time count can be used to determine how often a state or states occur within a trace specification.

Relative/Absolute Time Counts Compared

For equivilent measurements, absolute time shown for any single line will approximate the total of the relative times shown for all preceding lines. However, the two values will not usually equate exactly. This is because the 64000 may require up to 640 nanoseconds to measure the states in any line. This measuring time is included in the absolute time but it is not included in the relative time.

Count <STATES>

This command may be used to determine either: (1) whether a specific <STATE> did or did not occur within the overall trace specification (relative count) or (2) how many times a specific <STATE> occurred within the overall trace specification (absolute count).

A count <STATE> is similar to an <EVENT> except register <STATES> can not be counted and the not equal (< >) relationship can not be used.

The count <STATE> may be specified as any one of the following:

a. Count address = <ADDRESS>**[**data = <VALUE>**][**<STATUS>**]**

b. Count data = <VALUE> **[**<STATUS>**]**

c. Count <STATUS>

where:

<STATUS>, <VALUE>, and <ADDRESS> are the same as previously defined for <EVENT>.

Real-time/Nonreal-time Analysis Guidelines

Real-time Analysis

In general, real-time analysis is done with hardware and nonreal-time analysis is done with software techniques. The real-time analysis hardware is contained on the analysis board.

In general, real-time analysis can be performed for trace specifications consisting of address, data, and status states, provided:

a. no address, data, or status states are connected by an "OR".

b. the data and status states are specified as "=" to a value and not specified as "< >" to a value.

c. the complexity of the trace specification does not exceed the real-time processing capability of the hardware.

Real-time analysis is indicated on the display by the message "XXXX----running", where XXXX is the microprocessor type number.

Nonreal-time Analysis

The following trace specifications can not be done in real-time:

a. real-time analysis can not be performed for a trace specification which includes a register state. This is because the microprocessor program must be interrupted to sample internal register data.

b. real-time analysis can not be performed for a trace specification containing two or more events connected by an "OR".

c. real-time analysis can not be performed for any trace specification containing a data or status state specified as "< >" to a value.

Nonreal-time analysis is indicated on the display by the message "single cycling" or "pseudo run".

Using Analysis Commands

The source modules, written by the user, must be assembled into relocatable object modules and then linked into an absolute file or program. The absolute program must then be loaded into either the 64000 emulation memory or the target system memory. In some instances, the program may be loaded entirely into, and run from emulation memory. In other instances, it may be desirable to run code resident in the user's target system. In still other instances, a combination of both user memory and emulation memory may be used. Before loading memory, be sure the user/emulation memory and simulated I/O devices have been mapped to the configuration required to emulate the target system.

Analysis may be performed either by first initiating the program run and then specifying the trace parameters or by specifying the trace parameters first and then initiating the program run. In either case, once a trace command is initiated, the analysis module or software monitors the system buses of the emulation processor to detect the state sequence specified in the trace command. When the trace specification has been satisfied, a message will appear on the station display stating "trace complete". At that time the contents of the trace buffer are displayed. If the trace buffer content exceeds the page size of the display, the "NEXT PAGE", "PREV PAGE", "ROLL UP", or "ROLL DOWN" keys may be used to display all the trace buffer contents.

After the user examines the contents of the trace buffer, minor problems may be corrected by changing or inserting new absolute code into emulation memory. Such modifications can be made with the modify command. After the changes have been made, the processor and object code can be run again and the results examined. This cycle is continued until the software and hardware "play" together as expected. When the software is functioning correctly, it may be corrected in the source file, reassembled, relinked and reloaded into the target system.

After the software is debugged, the emulation cable may be removed from the target system, and the processor chip re-installed. The debugged software should operate as it did under emulation.

Trace/Count Command Syntax

The syntax listing on the following pages are intended primarily to acquaint the user with the different operational commands. The commands are listed alphabetically to facilitate quick referencing.

SYNTAX

$$\textbf{count} \begin{bmatrix} \text{time} \\ \text{<STATES>} \end{bmatrix}$$

Default Value

If neither time nor a set of conditions for <STATES> is entered, the 64000 defaults to counting time.

Examples:

count time

count 810H,,30H

count address = 900H data = 74H

FUNCTION

The count command is used to measure the elapsed time between the states of a trace or to determine the number of times a specified <STATE> occurs within a trace.

A count time command measures both absolute and relative time. The absolute time count displayed for a traced state is the total elapsed time measured between the trigger event and the traced state. The relative time count for a particular state is the measurement of the time interval between that state and the previously traced state.

A count <STATES> command determines if a specific <STATE> occurs during a trace and the number of times that that <STATE> occurs. A display of absolute state counts indicates both the points at which the state occurred in the trace and the number of times the state had occurred up to that point. For example, if the count corresponding to a traced state is "0", the specified state had not occurred up to that point. The first time that the displayed count is a "1" is the first time that the specified <STATE> appears in the trace. Likewise, the first occurrence of a count of "2" indicates the second occurrence of the <STATE> and so on.

count
(Cont'd)

The display count command determines whether absolute or relative counts are displayed. Absolute time or <STATE> counts are displayed by default if no display count command is entered. Absolute counts can be exchanged for relative counts by pressing the display count relative soft keys. Once relative counts have been selected, displays of either time or <STATE> counts are in the relative format until display count absolute is reselected or emulation is ended.

Parameters

time	time specifies that the time between events (relative) and the time elapsed since the trigger event will be measured and displayed.
<STATES>	<STATES> specifies the particular state or states which are to be counted when they occur simultaneously on the emulation bus. The syntax for <STATES> is described in Appendix A.

trace

SYNTAX

```
trace [[sequence <TERM>[ {or}   <TERM>]...
                         {then}
    [restart_on <TERM> [or <TERM>]...]
    [then <TERM>[ {or} <TERM>]...[restart_on <TERM>
                  {then}
    [or <TERM>]...]]...] (trigger) <POSITION><TERM>
    [or  <TERM>]...][(trc_)only <EVENT>[or <EVENT>]...]
    [continue]
```

Default Value

If the pretrigger and trigger specifications are omitted, the trigger <POSITION> defaults to after and the trigger <TERM> defaults to "don't cares".

Examples:

trace

trace continuous

trace only 810H or 900H,10H

trace after address = 54H continuous

trace in_sequence ,21H occurs 5 then 800H or 801H restart on data = 25H trigger about 10A0H

trace in_sequence 610H or 654H occurs 3 then 680H trigger after 685H trace_only data = 0FFH

FUNCTION

The trace command is used to analyze and display a particular portion of a program run. Each state of the program run is examined and if that state helps fulfill the requirements of the trace specification, it is stored in a trace buffer which can store up to 256 states. When the trace specification is satisfied, the contents of the trace buffer is displayed on the CRT.

Parameters

continue	continue instructs the 64000 to constantly collect and display the data indicated by the trigger specification. The specified pretrigger sequence, if any, must be satisfied prior to the collection of any data, but it need be satisfied only once.
<EVENT>	The <EVENT> variable specifies a particular state or set of states on the emulation bus and/or the contents of the internal registers of the emulation processor. The syntax requirements for <EVENT> are described in Appendix A.

or	or represents the logical OR condition. When used in a pretrigger sequence to separate the <TERM>s of a series, the occurrence of any one of the <TERM>s will satisfy that portion of the pretrigger sequence in which the series appears.
<POSITION>	<POSITION> determines whether the displayed trace states are those which occurred before, after, or about (both before and after) the specified trigger <TERM>.
restart_on	restart_on is used to specify <TERM>s which are not to occur during the search for the pretrigger condition. If any of the <TERM>s listed immediately following restart_on are detected before the current condition is satisfied, the search for the pretrigger sequence starts again from the beginning.
sequence	in_sequence indicates the beginning of a pretrigger sequence.
<TERM>	The <TERM> variable represents an <EVENT> which must occur a specified number of times. Refer to the definitions for <TERM> and <EVENT> which appear in Appendix A.
then	then is used in the pretrigger sequence to specify that the given conditions must occur in the order listed before the pretrigger sequence will be satisfied.
trc_only	trace_only is used to select the <EVENT>s or types of <EVENT>s (io_reads, opcodes, etc.) that are to be saved in the trace buffer for display when the trace is complete. The trc_only soft key appears as "only" if no pretrigger sequence is entered.
trigger	trigger is used with <POSITION> to determine whether the displayed trace states are to be those which occurred before, after, or about the trigger <TERM>. trigger only appears as a softkey when a pretrigger sequence is selected. In all other cases "trigger" is implied when a <POSITION> is specified.

DESCRIPTION

The trace command consists of an optional pretrigger sequence and a trigger specification. The pretrigger sequence is used to pinpoint the exact <EVENT> about which the analyzed states are to be saved. The trigger specification determines which particular states occurring before and after the trigger <EVENT> are to be saved in the trace buffer.

The pretrigger sequence begins with "in_sequence" and includes a series of conditions connected by "then" which must be satisfied in the order listed before the search for the trigger <EVENT> will begin. A condition consists of a list of one or more <TERM>s connected by the logical OR. If any one of the specified <TERM>s is detected, the current condition will be satisfied and the 64000 will begin searching for the states required to satisfy the next condition. The <TERM> which satisfies the condition is stored in the trace buffer and displayed on the CRT to indicate that the particular condition has been satisfied.

A condition can also include a list of <TERM>s which are not to occur before the current condition is satisfied. This list is an optional entry which is specified using the restart_on parameter. If any one of the restart <TERM>s occurs before the condition requirements are fulfilled, the 64000 will restart its search for the pretrigger sequence beginning with the states required to satify the first condition.

For example, suppose that the pretrigger sequence consists of the following two conditions:

```
trace in_sequence 800H or 835H then ,45H restart_on 838H ...
```

The first condition consists of "800H or 835H" and is satisfied when either an address of 800H or an address of 835H is detected. Once that requirement has been fulfilled, the 64000 begins to search for a state which will satisfy the second condition. The second condition consists of ",45H restart_on 838H". If the data = 45H appears before, or at the same time as, an address equal to 838H is detected, the search for the trigger <EVENT> will begin. However, if 838H appears on the address bus before 45H appears on the data bus, the trace buffer is cleared and the search for the state which satisfies the first condition of the pretrigger sequence begins again.

NOTE

> If a specification condition and a "restart_on" condition occur simultaneously, the specification condition takes precedence.

The trigger specification determines which of the traced states will be stored in the trace buffer for display upon completion of the trace. The trace buffer can be filled by those states which occur immediately before or immediately after the specified trigger <EVENT>, or half of the buffer can be filled by states which precede the trigger and half by those which follow the trigger <EVENT>. The storage option used is determined by the entry made for the <POSITION> parameter.

<EVENT>s can be selectively saved by pressing trc_only (or only) and entering the specific <EVENT>s to be saved. When this option is used, only the indicated states occurring in the specified <POSITION> relative to the trigger <TERM> are stored in the trace buffer.

Entry of the continuous parameter causes a continuous update of the display of the trace buffer contents (also continuously updated) once the pre-trigger sequence has been satisfied. If no pre-trigger sequence is entered, the continuous display of the trace buffer contents starts immediately after the trace begins. The trace will continue until the run is complete or until either a stop trace or stop run command is issued.

Chapter 8
Simulated I/O

Introduction

The "Simulated I/O" feature of the 64000 System allows the user to develop programs for, but without actually using, the target systems I/O hardware. To do this, the 64000 systems I/O hardware is used to "simulate" the target systems I/O hardware. This provides a double benefit. First, programs may be developed concurrently with hardware development, and second, if the target systems hardware exists but is not available to the programmer, program development can continue uninterrupted.

The following 64000 system hardware may be used to "simulate" the target system hardware during user-program development. (The 64000 hardware is listed in the order of description.)

- Printer
- Display
- Keyboard
- Disc
- RS-232 Communications Channel

Simulated I/O is described in this section as follows. First an overview is presented. The overview describes the common attributes of the five simulated I/O interfaces, and then briefly, the interfaces themselves. The intent of the overview is to acquaint the reader with the simulated I/O features.

Following the overview, each interface is described in detail. The intent of the detail descriptions is to provide sufficient information to allow a user to write the programs that will interface with the 64000 I/O devices. Following the detailed descriptions is a list of error codes, sample programs and file formats.

After the user has written, assembled (or compiled) and linked his/her I/O programs, they may be incorporated into an emulation configuration, executed, and tested.

Emulation configuration is described in Chapter 4 of this manual. Running and testing the programs is done with the commands described in sections 5 through 7 of this manual.

Overview

A general description of each of the simulated I/O interfaces is described in the following paragraphs. However, all of the interfaces have common attributes. These are described first.

Common Attributes

Each simulated I/O interface requires a unique memory location to which all I/O handshaking codes are sent by both the user and the 64000 programs. The address for this location is generically referred to as the control address, or CA. The 64000 samples these addresses periodically looking for commands. Location CA must be initially defined in the users program and in the emulation configuration. If more than one simulated I/O interface is to be implemented, then the user must make sure that each I/O program assigns a unique address for the CA. Additionally, the user program must allow for contiguous buffer spaces following the CA. The exact amount of, and use of this buffer space, is determined by the type of I/O interface. These requirements are specified in the detail descriptions of the interfaces.

The addresses for the different CA locations are entered into the 64000 program during emulation configuration. The user must NOT restrict the processor to real time runs when using simulated I/O. The CA locations must be located in memory space assigned as either user or emulator RAM. It is recommended that the CA locations be in emulation RAM since this will allow the user programs to run faster. Mapping the CA locations to user RAM will cause the emulator to single cycle while polling the CA locations for commands and or data. In the latter mode, if a "display registers" command is issued while simulated I/O is enabled (in the emulation configuration), the display will appear to be the same as from "display registers dynamic" command. This is normal and is a result of the pseudo run forced by the 64000 simulated I/O programs.

Certain of the I/O codes sent to location CA must also include supplemental information. This supplemental information is contained in the locations following CA, i.e., CA+1 through CA+n. The supplemental information must be placed in locations CA+1 through CA+n BEFORE the corresponding control code is placed in CA. If this is not done, the 64000 may respond to the control code in CA before the supplemental data is set into locations CA+1 through CA+n.

The user program must initiate the request to open the simulated I/O interface. To do this, after setting up the supplemental information in locations CA+1 through CA+n, the user program places the appropriate code into location CA. (Code 80H opens all interfaces except the disc file where it creates a file.) If the 64000 program successfully executes the request, it returns the appropriate code to location CA. (Usually a 00 is returned, but not always.) If the 64000 program cannot execute the request, an error code is returned to location CA. A group of predefined error codes is used. Within this group only a portion of the codes apply to each interface. These error codes are defined in general terms in table 8-8 which is located toward the end of this chapter. For those interfaces where the error codes also have specific meanings, the meanings are defined in the detailed descriptions of the interface. When the user is finished with the system resources, he should "close" the appropriate interfaces with the proper commands. All devices will automatically be closed by an "end simulation" command or by execution of a reset-reset.

Printer I/O Interface (See figure 8-1)

This is the simplest of the five I/O interfaces. Only three user-control codes are used to interface with the printer. These are: (1) open printer file, (2) write to the printer, and (3) close printer file.

A buffer space contiguous to location CA contains a value indicating the number of bytes (characters) to be printed followed by the characters themselves.

Display I/O Interface (See figure 8-2)

This is somewhat more complex than the printer I/O interface since it requires five user control codes. These codes are used to: (1) open the display file, (2) roll to and write line 18 (this is used to scroll lines up on the display), (3) select a starting line and column, (4) write from the selected line and column, and (5) close the display.

Depending upon the control code issued, a buffer space contiguous to location CA is required to hold one of the following parameter groups: (1) line length in bytes followed by the bytes to be displayed, (2) line and column number at which record display is to begin, or (3) record length in bytes followed by the record bytes to be displayed. The open and close codes use no additional buffer space other than location CA.

Keyboard I/O Interface (See figure 8-3)

The keyboard interface uses two user control codes and two keyboard input command word codes. Additionally, the 64000 returns one of 24 keyboard output command word codes.

The user control codes are used to open or close the keyboard interface file. The two keyboard input command codes are used to either: (1) clear the currently displayed line upon receipt of a keyboard character, or (2) append the character to the existing line.

When the keyboard file is opened, a buffer space contiguous to location CA is required to hold the keyboard input command word and the maximum record length specification. This specification defines the maximum record length that will be accepted from the keyboard. Thus, the buffer must be large enough to accept the keyboard output parameters and the maximum record length specified.

The keyboard output command word defines the manner in which the input line was terminated or the status of the keyboard output record. The output record consists of ASCII coded character bytes.

Disc Files I/O Interface (See figure 8-4)

CAUTION

The disc file simulated I/O control codes can be used to access critical system files. Extreme care should be used if any of the following types of files are accessed:

Emulation Command Files (Type 6)

Linker Command Files (Type 7)

Linker Configuration Files (Type 8)

Assembler Configuration Files (Type 10)

Incorrectly accessing these files may destroy them and cause serious system problems!

The simulated disc file interface uses ten user control codes. These codes allow the user program to: (1) create, open, close, or delete a file; (2) advance to, backup to, or randomly select a record position within a file; (3) automatically select record postion 1 in the file; and (4) read from, or write into any selected record postion in the file. The user may also assign a different file name to be associated with an already existing CA.

Depending upon the control code issued, a buffer space contiguous to location CA is required to hold one of the following parameter groups: (1) file type number, (2) disc number, (3) record number, (4) maximum number of words to read or write, or (5) the actual number of words read or written, followed by the words themselves.

No buffer space is required following the control codes used to close the file and to automatically select record position 1 in a file.

RS-232 I/O Interface (See figure 8-5)

This is the most complex of the five I/O interfaces. To use this interface, the following distinct events MUST be implemented between the user and 64000 programs: (1) the RS-232 interface must be opened; (2) the 8251 Universal Synchronous/Asynchronous, Receiver/Transmitter, or USART, is initialized; (3) using the appropriate command word, an 8251 operating mode is selected; (4) data may be written to, or read from, the 8251; and (5) when data transfer is complete, the RS-232 file may be closed.

To implement the interface, the user program must allow for control space contiguous to location CA as shown in figure 8-5. During 8251 initialization, locations CA+1 through CA+5 hold the command and status words used to initialize and select the operation of the 8251.

The user program may read or write single bytes or multiple-byte records. When reading or writing single bytes, the single byte is passed through location CA+1. If multiple byte records are to be handled, the user program must set up read and write buffers as shown in figure 8-5.

When writing multiple byte records, locations CA+6 through CA+14 hold the write buffer pointers and the actual number of bytes sent by the 8251. This data is used interactively between the user and 64000 programs to transfer write data from the users program, via the users and 64000 write buffers, to the 8251.

When reading multiple-byte records, location CA+15 through CA+23 hold the read buffer pointers and the actual number of bytes received by the 8251. This data is used interactively between the user and 64000 programs to transfer read data from the 8251, via the 64000 and users read buffers, to the user program.

The read and write buffers may be updated individually or together by the user program.

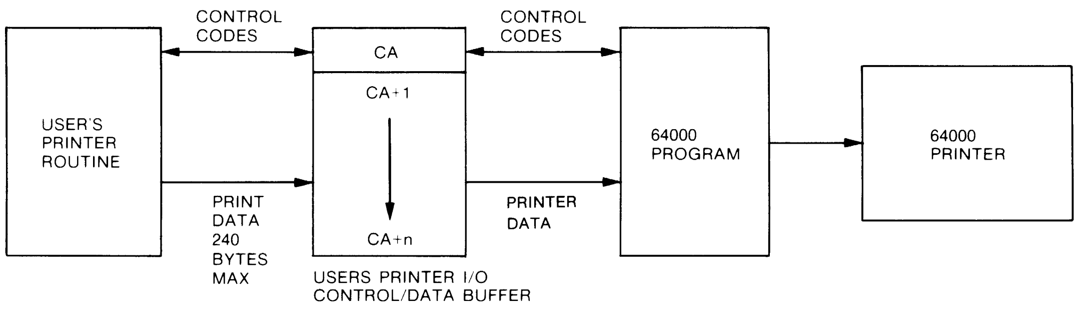


Figure 8-1. Simulated Printer I/O Interface Diagram

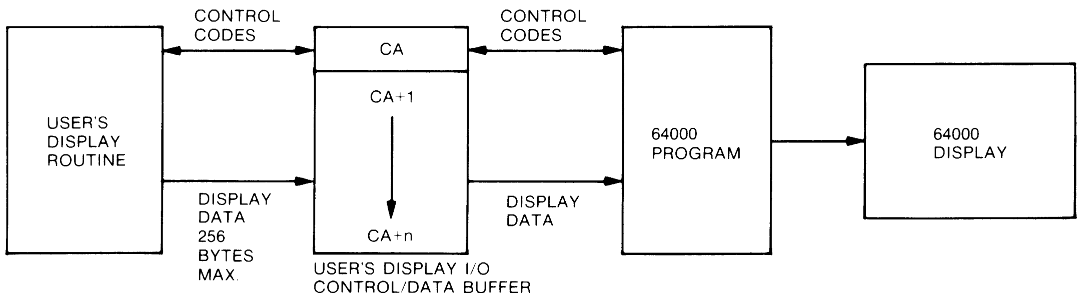


Figure 8-2. Simulated Display I/O Interface Diagram

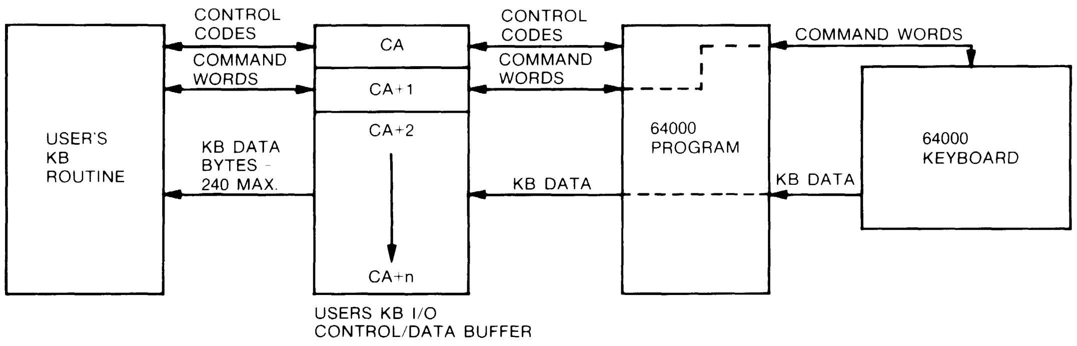


Figure 8-3. Simulated Keyboard I/O Interface Diagram

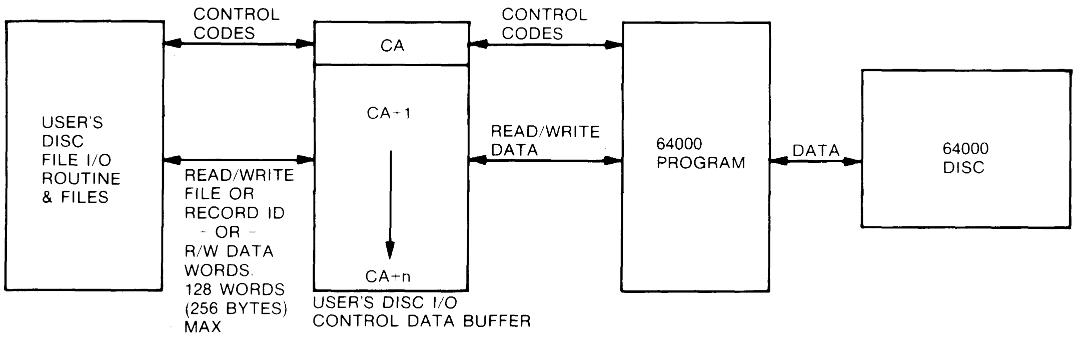
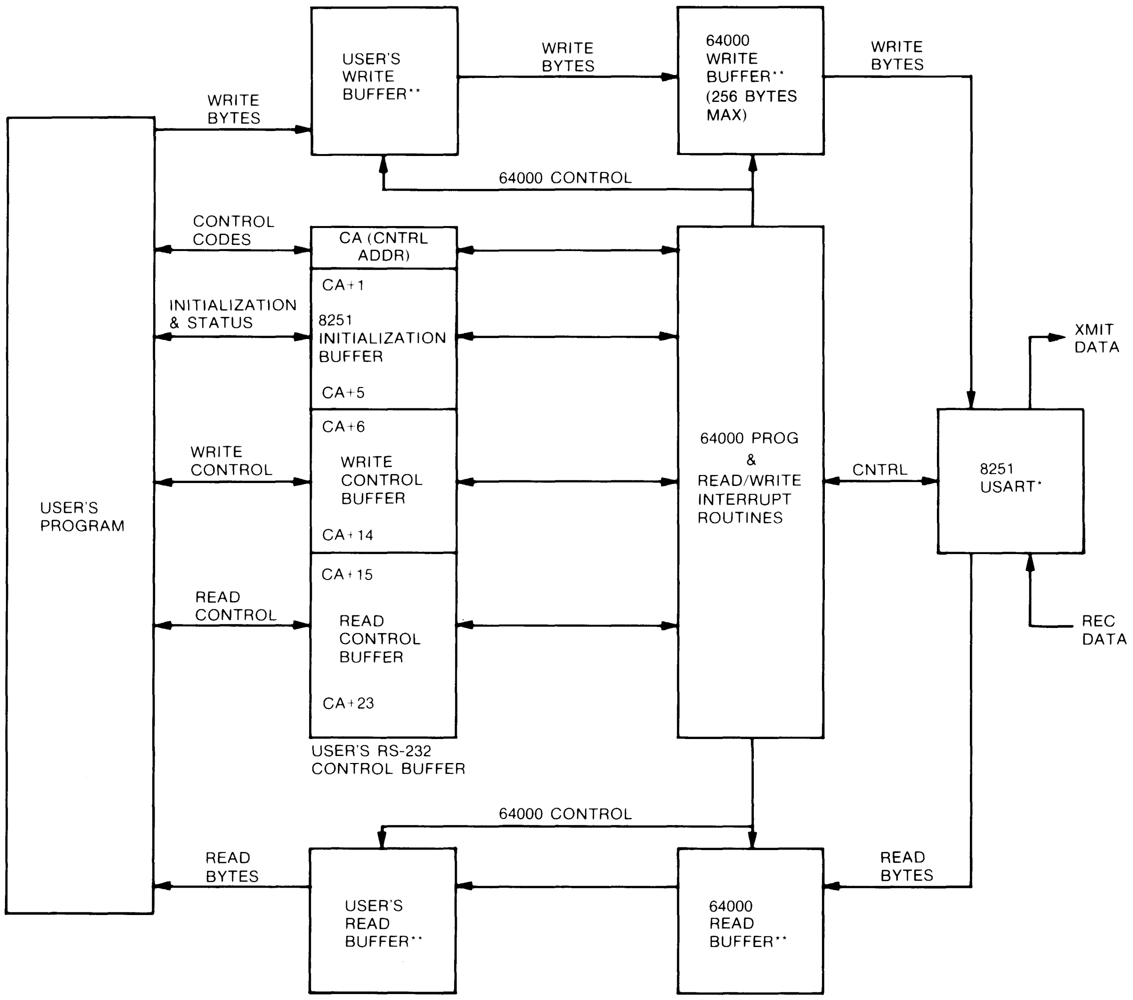

CONTROL CODES
USER'S DISC FILE I/O ROUTINE & FILES
READ/WRITE FILE OR RECORD ID - OR - R/W DATA WORDS. 128 WORDS (256 BYTES) MAX
CA
CA+1
CA+n
USER'S DISC I/O CONTROL DATA BUFFER
CONTROL CODES
READ/WRITE DATA
64000 PROGRAM
DATA
64000 DISC

Figure 8-4. Simulated Disc File I/O Interface Diagram



*USART = Universal Synchronous/Asynchronous Receiver/Transmitter.
**Buffers are required only if records are to be read or written. Single bytes do not require these buffers.

Figure 8-5. Simulated RS-232 I/O Interface Diagram

Printer I/O Interface

The following paragraphs describe the events which must be implemented between the user and the 64000 program for printer I/O to occur. The events are:

- Open Printer File
- Write to Printer
- Close Printer File

The above events, the corresponding control codes, and parameters, where applicable, are summarized in table 8-1.

NOTE

During the time that a simulated I/O printer file is open, no other user can access the printer. Thus, be sure to close the file when finished.

Open Printer (80H)

Before using a "write to printer" code, the user program must request that the printer interface be opened. This is done by placing code 80H into location CA.

NOTE

CA represents the memory location to which all printer I/O "handshaking" codes are sent by both the user and the 64000 program. The actual address for the printer is defined in the user program and entered into the 64000 program during the configuration of the emulation CMDFILE. Each I/O interface - printer, RS-232, display, etc. - requires its own unique CA address.

Certain of the I/O codes sent to location CA must also include supplemental information. This supplemental information is generally contained in the locations following CA, i.e., CA+1 through CA+n. The supplemental information must be placed into locations CA+1 through CA+n BEFORE the corresponding control code is placed in CA. If this is not done, the 64000 may respond to the control code in CA before the supplemental data is set into locations CA+1 through CA+n.

The 64000 program responds by opening the printer file and returning a 00 to location CA. If the file cannot be opened, error codes are returned as shown in table 8-1.

After the file is opened, the user program may issue a write-to-printer code as described in the next paragraph.

Write to Printer (82H)

To send a write record to the printer, the user program places the following parameters into locations CA+1 through CA+n and then after setting up locations CA+1 through CA+n, places code 82H into location CA.

The record length in bytes is entered into location CA+1. The record length must be a minimum of two bytes and may be a maximum of 240 bytes in two byte increments. That is - the record must always contain an even number of bytes. Odd bytes should be padded with a space (20H).

Locations CA+2 through (CA+2)+n contain the ASCII codes of the character to be printed.

The 64000 responds by supplying the write record to the printer and returning a 00 to location CA. The 64000 automatically sends a carriage return/linefeed to the printer following the user data. If the write-to-printer record is not accepted, an error code is returned as listed in table 8-1.

Close Printer File (81H)

The user program closes the printer file by placing code 81H into location CA. The 64000 responds by closing the file and returning code 00 to location CA. The 64000 will perform a form feed automatically.

If the close file is not accepted, an error code is returned to location CA as shown in table 8-1.

Display I/O Interface

The following paragraphs describe the events which must be implemented between the user and the 64000 programs for display I/O to occur. The events are:

- Open Display File
- Roll To / Write line 18 (scroll and write)
- Select line and column
- Write from selected line/column
- Close Display File

The above events, the corresponding control codes and parameters, where applicable, are summarized in table 8-2. Display techniques are shown in figure 8-6.

NOTE

During the time that the simulated I/O display file is open, the standard 64000 keyboard has no control over the display. To regain control, press the **SIMULATE** soft key which closes the file. If the keyboard file is also open, it too is closed when the soft key is pressed.

Table 8-1. Printer I/O Codes

Request Name	User Program Request		64000 Response To:		
			Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
OPEN PRINTER FILE	CA	80H	CA	00	01 thru 08 09: file is already open. 10-14: NA
CLOSE PRINTER FILE	CA	81H	CA	00	01 thru 08 09: file is already closed. 10-14: NA
WRITE TO PRINTER	CA	82H	CA	00	01 thru 08
	CA+1	Record Length in bytes (240 max.)	The 64000 accepts the record and causes it to be printed.		09: file is not open. 10, 11, 13 & 14: NA
	CA+2 ↓ (CA+2) +n	Record byte 1* ↓ Record byte n*			12: Record length exceeded 240 bytes.

*All display characters must be formatted in ASCII code. A code greater than 0F0H will not be accepted by the 64000 program.

NA= Not Applicable.
See table 8-8 for complete error code listing.

Open Display File (80H)

Before any writing can be done on the display, the user program must request that the display file be opened. This is done by placing code 80H into location CA.

NOTE

CA represents the memory location to which all display I/O "handshaking" codes are sent by both the user and the 64000 program. The actual address for the display I/O CA is defined in the user program and entered into the 64000 program during the configuration of the emulation CMDFILE. Each I/O interface - display, RS-232, printer, etc. -requires its own unique CA address.

Certain of the I/O codes sent to location CA must also include supplemental information. This supplemental information is generally contained in the locations following CA, i.e., CA+1 through CA+n. The supplemental information must be placed into locations CA+1 through CA+n BEFORE the corresponding control code is placed in CA. If this is not done, the 64000 may respond to the control code in CA before the supplemental data is set into locations CA+1 through CA+n.

The 64000 program responds by opening the display file, and returning a 00 to location CA. If the file cannot be opened, error codes are returned as shown in table 8-2.

After the file is opened, the user program may write on the display as described in the following paragraphs.

Roll To/Write Line 18 (82H)

This command allows writing to be initiated at the bottom of the display. Sequential Roll Up/Write Line 18 commands cause the previously written line 18 to roll to line 17, etc. Thus, writing is always done on the bottom line and the previously written lines are shifted up as each new line 18 is written.

To cause the display to roll up and begin writing on line 18, the user program places the following parameters into location CA+1 through CA+n, and after setting up locations CA+1 through CA+n, then places code 82H into CA.

The line length in bytes is entered into location CA+1. The line length must be a minimum of two bytes and may be a maximum of 80 bytes, in two byte increments. That is, the line must always contain an even number of bytes. If the user writes an odd number of bytes, the 64000 will pad the line with a null.

Locations CA+2 through (CA+2)+n contain the ASCII codes of the characters to be written on line 18. The 64000 responds by storing this data in a display buffer and returning a 00 to location CA. A delay may occur before the program rolls up and writes to line 18. Thus, a program wait may be required. If writing cannot be done, especially if write roll/column is used (roll/column does not use delay), an error code is returned as listed in table 8-2.

After initially rolling up and writing on line 18, subsequent Roll Up/Write Line 18 commands cause the previously written line 18 to roll up to line 17, line 17 to roll to line 16, etc. Although the 64000 responds almost immediately with a 00 in CA, the actual scrolling of a line can take up to 200 msec. The 64000 will accept other commands during this time. Future scrolls are buffered and performed in sequence. Row/Column writes will be performed immediately and may be scrolled if a previous scroll has not been completed.

Select Starting Line/Column (83H)

The user programs may specify the line number and column number at which writing, when indicated, will start. To do this, the user program places the line number (1 through 18) into location CA+1, the column number (1 through 80) into location CA+2, and then places code 83H into location CA.

The 64000 responds by storing the line and column number and returning code 00 to location CA. The line and column numbers are stored until either writing is initiated (code 84H) or the display file is closed.

If the line and column numbers are not accepted by the 64000 program, an error code is returned to location CA as listed in table 8-2.

Figure 8-6 shows the display techniques.

Write From Starting Line/Column (84H)

Before writing can be initiated, a starting line number and column number must be specified by the user program. After this is done, writing may be initiated as follows: the user program initiates writing by placing the record length (i.e., number of characters to be displayed) into location CA+1, the actual display characters (ASCII codes) into locations CA+2 through (CA+2)+n, and then places code 84H into location CA.

The maximum record length is 255 bytes. The display characters must be formatted in ASCII codes. The 64000 program will not accept a display code greater than 0F0H.

The 64000 responds by displaying the record beginning at the starting line and column specified by code 83H. If the record exceeds the length of the starting line, writing continues at column one of the next line, etc.

If the 64000 cannot initiate writing as requested, an error code is returned to location CA as shown in table 8-2.

Close Display File (81H)

The user program closes the display file by placing code 81H into location CA. The 64000 responds by closing the file and returning code 00 to location CA.

If the close file is not accepted, an error code is returned to location CA as shown in table 8-2.

Pressing the inverse video **SIMULATE** key or performing a "reset-reset" will automatically close the display. Closing the display also closes the keyboard.

Keyboard I/O Interface

The operation of the keyboard I/O interface is described in the following four phases:

- User Program Requests Keyboard Read
- 64000 Response to Keyboard Read Request
- 64000 Detects Positive KB Output Command Word
- User's Program Detects 00 in CA

Each of the above phases corresponds to a significant interaction which must be implemented between the user program and the 64000 program for keyboard I/O to occur.

The keyboard I/O interface events are summarized in figure 8-7 and table 8-3.

NOTE

To automatically close the simulated I/O keyboard file and return the keyboard to standard operation, press the **SIMULATE** soft key. If the display file is also open, it too is closed when the soft key is pressed.

Table 8-2. Display I/O Codes

Request Name	User Program Request		64000 Response To: Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
OPEN DISPLAY FILE	CA	80H	CA	00	01 thru 08 & 14
			The 64000 program opens the file and clears the display		09 code >84H or file is open
					10 thru 13: NA
CLOSE DISPLAY FILE	CA	81H	CA	00	01 thru 08 & 14
					09: file is already closed.
					10 thru 13: NA
ROLL TO/ WRITE LINE 18	CA	82H	CA	00	01 thru 08 & 14
	CA+1	Line length in bytes (80 max)	The 64000 program stores this data in a display buffer. A delay may occur before rolling to and writing on line 18 actually occurs.		09: file is not open 10, 11, & 13: NA
	CA+2	Line byte 1*	A program wait may be required. If successive line 18's are written, then the preceeding line 18 is rolled to line 17, 17 to 16, etc.		12: Invalid record length
	(CA+2) +n	Line byte n*			

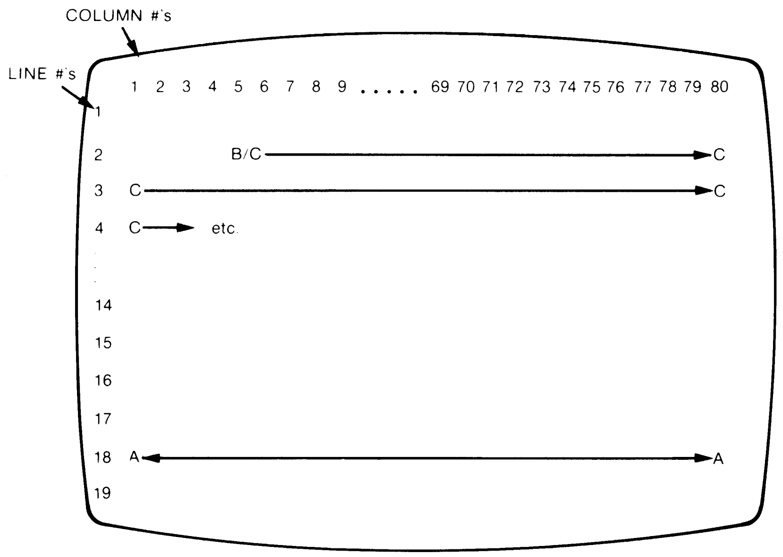
Table 8-2. Display I/O Codes (Cont'd)

Request Name	User Program Request		64000 Response To: Valid User Request		Invalid Request
	Address	**Contents**	**Address**	**Contents**	**Error Code**
SELECT STARTING LINE/ COLUMN	CA	83H	CA	00	01 thru 08 & 14
	CA+1	Line # (1-18)	The 64000 program stores the line and column numbers until a write line/column request is issued or the file is closed.		09: File is not open
	CA+2	Column Number (1-80)			10, 12 & 13: NA 11: Invalid line or column number.
WRITE FROM STARTING LINE/	CA	84H	CA	00	01 thru 08, 13 & 14
	CA+1	Record length in bytes (255 Max)	The 64000 program displays the record starting at line/ column selected by code 83H. If record exceeds one line, writing continues at column 1 of next line,etc. See figure 8-6.		09: file not open. 10 & 12: NA 11: line/column not specified by 83H.
	CA+2 ↓ (CA+2) +n	Record byte#1 ↓ Record byte n*			

*All display characters must be formatted in ASCII code. A code greater than 0F0H will not be accepted by the 64000 program.

NA= Not Applicable.

See table 8-8 for complete error code listing.



64000 DISPLAY

DISPLAY LETTER	MEANING
A	Code 82H automatically causes the display to roll to line 18. Up to 80 characters, in two byte increments, may be written on the line. Sequential Roll To / Write Line 18 commands cause the previous line 18 to roll to line 17, line 17 to roll to line 16, etc.
B/C	B is the point (line 2, column 5) defined by code 83H at which writing will begin. C is the statement which is defined by code 84H and begins at point B. There is no limit on the record length defined by 84H. If the record exceeds the length of line 2, it is continued on line 3 at column 1, etc.

Figure 8-6. Display Techniques

User Program Requests Keyboard Read (80H)

Before any other keyboard operation can be initiated, the user program must request that the KB I/O interface be opened. This is done by first placing the KB-input-command word and the maximum record length specification into the KB I/O buffer as shown in Phase I of figure 8-7. Then, after setting up locations CA+1 through CA+n, code 80H is placed into location CA of the buffer.

NOTE

CA represents the memory location to which all KB I/O codes are sent by both the user program and the 64000 program. The actual address of CA is defined in the user program and entered into the 64000 program during the configuration of the emulation CMDFILE. Each I/O interface - keyboard, RS-232, printer, etc. -requires its own unique interface.

Certain I/O codes sent to location CA must also include supplemental information. This supplemental information is contained in the locations following CA, i.e., CA+1 through CA+n. The supplemental information must be placed into locations CA+1 through CA+n BEFORE the corresponding control code is placed into CA. If this is not done, the 64000 may respond to the control code in CA before the supplemental data is set into locations CA+1 through CA+n.

The KB-input-command word is placed in buffer location CA+1. This word contains either a "–1" or "–2" code. A "–1" code causes the current line not to be cleared on the first character (i.e., the current keyboard characters are appended to any characters already displayed on the same line). A "–2" code causes the current line to be cleared on the first character (i.e., previously displayed characters are erased from the line and only the current keyboard characters are displayed).

The maximum record length specification is placed in buffer location CA+2. This is the maximum record length (i.e., number of keyboard characters) that the user program will accept from the keyboard. The record length specification may specify up to 240 characters (3 lines on the 64000 display). However, the keyboard may transmit more or less characters than this specification. If the number of characters transmitted exceeds the record length specification, the user program is informed of this by an applicable code in the KB-output-command word as described below.

64000 Response to Keyboard Read Request

The 64000 program responds to the KB read request by storing the KB-input-command word and record length specification, and by placing code 82H into location CA as shown in figure 8-7.

The 64000 program sets the KB-output-command word to the same code specified in the KB-input-command word (−1 or −2).

The 64000 then begins monitoring the keyboard until it detects an output command word. The result of this detection is described in the following paragraphs.

64000 Detects Positive KB-Output-Command Word

The keyboard may send either a KB-output-command word by itself or a command word followed by one or more keyboard characters. In either case, when a KB-output-command word is detected, the 64000 program places the word, and if applicable, other data into the KB I/O buffer as shown in figure 8-7 (Phase III). The KB output word, which is always sent, is placed in buffer location CA+1.

The 64000 program places a 00 in location CA to indicate to the user program that either a KB command and/or data is now available.

If keyboard characters are also sent and if a "lost character" was generated then the "lost character" is placed into location CA+2. (How a "lost character" is generated is described later.) Also, when keyboard characters are sent, the actual number of characters in the string (i.e., actual record length) is placed into location CA+3. The keyboard characters themselves (ASCII coded bytes) are placed into locations CA+4 through (CA+4)+n.

The KB output command in location CA+1 may be any one of the codes shown in table 8-4. Two of these codes, 8 and 24, will occur only if the actual record length from the keyboard exceeds the maximum record length specification. If either of these codes is generated, then location CA+2 contains the ASCII code of the surplus or lost character that exceeded the specified record length. A lost character may be generated in either of two ways:

a. When characters are entered as a continuous string and the string exceeds the specified record length. For this case, the first character to exceed the specified record length is placed in "lost character" location CA+2. If typing continues, each individual surplus character is placed into the "lost character" location CA+2 replacing the previous character. Thus, the last "lost character" entered remains in location CA+2.

b. When a character is inserted into a full record. For this case, the character at the end of the already full record is placed into "lost character" location CA+2. If additional characters are inserted, each succeeding end character is placed into CA+2, replacing the previous character.

User's Program Detects 00 in CA

After detecting a 00 in location CA, the user program takes the data from the KB I/O buffer and places either 80H or 81H into location CA. The results of each of these response codes are as follows:

a. 80H Response Code - Read Keyboard I/O

   If the user program responds with code 80H, the KB-input-command word and record length specifications must be supplied by the user program as shown in figure 8-7.

   The 64000 program responds by again reading the keyboard.

b. 81H Response Code - Close KB I/O

   If the user program responds with code 81H, the 64000 program closes the KB I/O interface. This command will also close the display file if it was open.

Disc File I/O Interface

CAUTION

The disc file simulated I/O control codes can be used to access critical system files. Extreme care should be used if any of the following types of files are accessed:

Emulation Command Files (Type 6)

Linker Command Files (Type 7)

Linker Configuration Files (Type 8)

Assembler Configuration Files (Type 10)

Incorrectly accessing these files may destroy them and cause serious system problems!

Table 8-3. Keyboard I/O Interface Codes

			64000 Response To:		
Request Name	**User Program Request**		**Valid User Request**		**Invalid Request**
	Address	**Contents**	**Address**	**Contents**	**Error Code**
OPEN KB INTER FACE	CA	80H		See 82H, below	08, 12, or 14
	CA+1	KB Input Command Word			Other codes do not apply
	CA+2	Max. Record Length Specifi-cation (up to 240 bytes)			
READ IN PROCESS		Initiated by 64000 program in response to 80H above	CA	82H	
			64000 stores KB-input-command word & max. record length spec. It then monitors KB-output-command word until positive word is detected and then responds as follows:		
OUTPUT AVAILABLE		Initiated by 64000 after 82H, above	CA	00	
			CA+1	KB out-put command word	

Table 8-3. Keyboard I/O Interface Codes (Cont'd)

			64000 Response To:		
Request Name	**User Program Request**		**Valid User Request**		**Invalid Request**
	Address	**Contents**	**Address**	**Contents**	**Error Code**
		User program may then respond to 00 with 80H or 81H as shown below.	CA+2	Reserved for Lost Character	
			CA+3	Actual record length (#of KB bytes)	
			CA+4 ↓ (CA+4) +n	KB Byte 0 ↓ KB Byte n	
CLOSE KB I/O	CA	81H	CA	00	08 or 14 Other codes do not apply.

See table 8-8 for complete error code listing.

Table 8-4. Command Word Codes

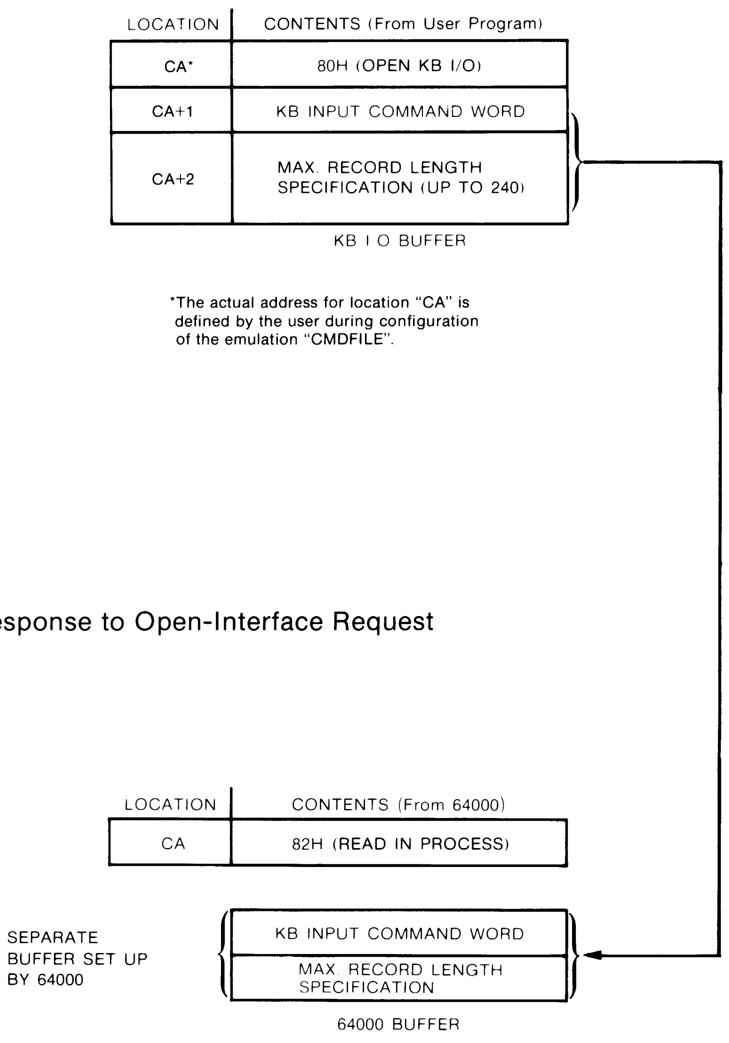
Part A. KB - Input - Command Word

Code	Meaning
-1	Current line not cleared. Characters appended to previously displayed characters.
-2	Current line cleared. Previously displayed characters erased.

Part B. KB - Output - Command Word

Code	Meaning
8	Insert character in full line (lost character placed in CA+2)
9	Tab Key
10	Down arrow key
11	Up arrow key
12	Display next page
13	Carriage return
14	Attempting to move cursor right past last allowed screen location
15	Attempting to move cursor left past first allowed screen location
16	Delete character from full line
17	Shift key
18	Display previous page
19	Roll display down
20	Roll display up
21	Shift right arrow key
22	Shift left arrow key
23	Clear line key
24	Actual record length exceeded record length specification (lost character placed in CA+2)

Phase I - User Requests Interface Opening

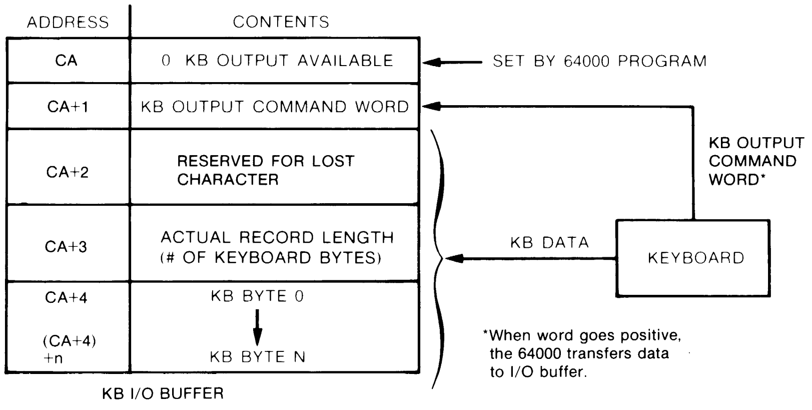


*The actual address for location "CA" is defined by the user during configuration of the emulation "CMDFILE".

Phase II - 64000 Response to Open-Interface Request

Figure 8-7. Keyboard I/O Interface Sequence

Phase III - 64000 Detects Positive KB Output Command Code



Phase IV - The user program may respond with either an 80H code as shown for phase I or an 81H code which closes the simulated keyboard I/O interface.

Figure 8-7. Keyboard I/O Interface Sequence (Cont'd)

The following paragraphs describe the type of files and the events which must be implemented between the user and the 64000 program to either: (1) create a new disc file, or (2) read from, write into, delete, or change the name of an existing file. The file types are described first. Then, the program events are described in the following order:

a. Creating New File

    1) Creating File (80H)

    2) Writing First Record (89H)

    3) Writing Additional Records (89H)

    4) Closing Created File (82H)

b. Accessing Existing File

  1) Opening File (81H)

  2) Selecting Record

    (a) Automatic selection of records 1, 2, 3, ... etc.

    (b) Advance "N" records (84H)

    (c) Backup "N" records (85H)

    (d) Position to record "N" (86H)

    (e) Rewind to record one (88H)

  3) Reading Record (87H)

  4) Writing Record (89H)

  5) Closing Open File (82H)

c. Deleting File (83H)

d. Changing File Name Associated with a CA (8AH)

The predefined file types are listed in table 8-5.

Table 8-6 summarizes the user program requests, the corresponding control codes, and, where applicable, corresponding parameters.

File Types

There are 12 predefined types of files identified by numbers 2 through 13 which may be created by the user program. The names and type numbers of these files are listed in table 8-5. File formats for these files are shown at the end of this chapter.

File type numbers 14 through 255 may be assigned to files defined by the user program, as required. It should be noted, however, that HP may require some unassigned numbers for future use. It is, therefore, recommended that the user leave space for this possibility, starting with number 14.

NOTE

Once created, file types 14 through 255 can only be deleted by using the simulated I/O delete command.

The overall file name is assigned during emulation configuration. Under any one file name, only one each of a file type may be created. For example, a file named USA may only have one each of file types 2 through 255. It cannot have two type 3 files, etc.

Creating New File

Creating File. To create a new file, the user program places the file type number into location CA+1, the disc number into location CA+2, and then places code 80H into location CA. (The disc number is the disc upon which the file will reside.)

NOTE

CA represents the memory location to which all disc file I/O "handshaking" codes are sent by both the user program and the 64000 program. The actual address for the disc files CA is defined in the user program and entered into the 64000 during the configuration of the emulation CMDFILE. Each I/O interface - disc files, display, keyboard, etc. - requires its own unique CA address.

Certain I/O codes sent to location CA must also include supplemental information. This supplemental information is contained in the locations following CA, i.e., CA+1 through CA+n. The supplemental information must be placed into locations CA+1 through CA+n BEFORE the corresponding control code is placed into CA. If this is not done, the 64000 may respond to the control code in CA before the supplemental data is set into locations CA+1 through CA+n.

The 64000 responds by creating the file type requested and returning a 00 to location CA which indicates the file has been created.

If the file cannot be created, an error code as shown in table 8-6 is returned to location CA. (General definitions for the error codes are listed in table 8-8.)

After the file is created, the user program may either write records immediately into it, or close it, and then reopen it and write records into it later.

Writing First Record. After a file is created the first record is written into it as follows. The user program places parameters, as described below, into locations CA+1 through CA+n, and then places code 89H into location CA.

The number of words in the write record is placed into location CA+1. A write record may contain up to a maximum of 128 words (256 bytes). Thus, an even number of bytes (whole words) must always be written. An unused byte in the last word must contain and ASCII blank.

Locations CA+2 through (CA+2)+n contain the words of the write record.

The 64000 responds by automatically writing the records into the file as record number 1. After the record is successfully written, the 64000 returns a 00 to location CA. If the record cannot be written, an error code, as listed in table 8-6, is returned to location CA.

Additional records are written into the file as described in the next paragraph.

Writing Additional Records. If the newly created file is still open (i.e., has never been closed), additional records are written into the file as described for record one with the following difference. Each succeeding record is automatically written with the next corresponding record number. Thus, the second record written becomes record number 2, the third record written becomes record number 3, etc.

Closing Created File. To close the newly created file, the user program places code 82H into location CA. The 64000 responds by closing the file and returning a 00 to location CA. If the file cannot be closed, an error code, as listed in table 8-6, is returned to location CA.

Accessing Existing Files

Opening File. To open an existing file, the user program places the file type number into location CA+1, the disc number into location CA+2, and then places code 81H into location CA.

The 64000 responds by opening the file and returning a 00 to location CA which indicates the file is open. If the file cannot be opened, an error code, as shown in table 8-6, is returned to location CA.

***CAUTION**

When a record is written into a file, it always becomes the last record in the file. Thus, writing a record into any location other than at the end of the file effectively erases all the following records in the file. When accomplishing the following paragraph choose record positions with care!

After the file is opened, the user program may either: (1) immediately read/write* record 1, (2) select any record for reading, or (3) select a position within the file to begin writing*.

Selecting Record. Records are selected in any of the following ways:

a. **Automatic selection of records 1, 2, 3, ..., etc.** When the file is opened, record 1 is automatically selected. Thus, it may be immediately written into, or read from, without first selecting it with and "advance", "position", or "rewind" code. After reading or writing record 1, record 2 is automatically selected and may be read from, or written into. This process can be continued for records 3, 4, 5, ..., etc.

NOTE

Remember, that when a record is written into a file, it becomes the end of the file.

b. **Advance "N" Records.** Records located ahead of the currently selected record (i.e., those records with higher numbers) may be selected as follows. The user program places the number of records into locations CA+1 and CA+2, and then places code 84H into location CA. The number of records is selected with a 15-bit word. The eight least significant bits are located in CA+1. The seven most significant bits are located in CA+2. The most significant bit in CA+2 is not used.

The 64000 responds by advancing the specified number and returning a 00 to location CA. If the record cannot be selected, an error code, as shown in table 8-6, is returned to location CA.

After the record is selected, the user program may then either read from or write into it.

c. **Backup "N" Records.** Records located behind the currently selected record (i.e., those records with smaller numbers then the current record) are selected in a way very similar to "advance "N" records". The only difference is that backup code 85H is placed into location CA. Locations CA+1 and CA+2 contain the number of records as defined in subparagraph b above. The 64000 also responds as described above.

d. **Position to Record "N".** Any record within the file may also be selected without knowing its location relative to the current record. This method is also similar to the "advance" or "backup" methods. The difference is that position code 86H is placed into location CA. Location CA+1 and CA+2 contain the record number as defined in subparagraph b above. The 64000 responds as described above.

e. **Rewind to Record One.** This is a fast way to select record 1. This method differs from the previous selection method in several ways. First, only record 1 can be selected using this method. Second, the user program places code 88H into location CA. Third, there are no entries required in locations CA+1 and CA+2. The 64000 program responds as described in subparagraph b above.

Reading Record. Once a record has been selected by one of the methods described above, it may be read as follows. The user program places the maximum number of 16-bit words it will accept from the record into location CA+1. Up to 128 words may be accepted. (The recommended technique is always set CA+1 to 128. Then, after reading is complete, throw away those words not wanted, if any.) After specifying location CA+1, code 87H is placed into location CA.

If the record is read successfully, the 64000 responds as follows: code 00 is returned to location CA. The actual number of 16-bit words read from the buffer is placed in location CA+1. Location CA+2 through (CA+2)+n contains bytes 0 through n.

If the record cannot be read, an error code, as shown in table 8-6, is returned to location CA.

Writing Record. A new record may be written into an existing file in either one of two ways. It may be added to the end of the file or it may be written over an existing record in the file. However, if an existing record is written over, then the newly written record becomes the last record in the file.

To add a record to the end of the file, the record selected must be one greater than the last record in the file. For example, if a file contains five records, then record 6 must be selected before writing is initiated. (If record 5 is selected, it will be written over by the new record.) After writing record 6, record 7 may be written by issuing another write code, etc.

To write over an existing record, first select the record and then initiate writing. Again, remember that all following records in the file are erased. For example, if a file contains 10 records, and record three is written over, then records four through ten are erased.

Closing Open File. An open file is closed in the same way as described for a newly created file. That is, the user program places code 82H into location CA. The 64000 responds by closing the file and returning a 00 to location CA. If the file cannot be closed, an error code, as listed in table 8-6, is returned to location CA.

Deleting Files

To delete a file, the user program places the file type into location CA+1, the disc number into location CA+2, and then places code 83H into location CA. The 64000 responds by deleting the file. If the file cannot be deleted, an error code is returned to location CA as shown in table 8-6. This delete is similar to a "purge" command in the general operating system. The purged file does go into the recoverable file list.

Changing File Name Assigned to a Particular CA

The file name associated with a given CA location may be changed. This does not rename any files on the disc, but simply changes the name in the emulation command file associated with a given CA. To do this the user must first make sure that the present file associated with the CA of interest is closed.

To change the file name in the emulation configuration file, the user program places the new name record into locations CA+1 through CA+16, and then places code 8AH into location CA. The name record is a fixed length record consisting of eight, 16-bit words. This record contains the record name, USERID, and specifies the length of both of these items.

The name must contain at least one character and may be up to nine characters long. The ID may be up to six characters long. However, the name and ID lengths are specified in a unique way. Also, the words containing these characters must be packed in the name record. Specifying name and character lengths and packing the words are done in the same way as described for the "microprocessor Configuration Record" in the Linker Symbols File description. This discription is located toward the end of this chapter.

To actually change the name of an existing file, the user must copy the contents of the file under the old file name into the file with the new file name. Either one or both of these files names may be specified by the user program at run time and accessed after "change file name" has been issued to the appropriate CA locations.

Table 8-5. Disc File Type Numbers and Names*

File Type Number	File Name
2	Source
3	Relocatable
4	Absolute
5	Listing
6	Emulation Command
7	Linker Command
8	Trace
9	PROM Absolute
10	Reserved for System Expansion
11	Compiler
12	Assembler Symbols
13	Linker Symbol
**14 thru 255	Types are defined and numbers assigned by user program.

* Formats for selected files are described at the end of this chapter.

** HP may require some unassigned numbers for future use. It is, therefore, recommended that the user leave space for this possibility, starting with number 14.

CAUTION

The disc file simulated I/O codes can be used to access critical system files. Extreme care should be used if any of the following types of files are accessed:

Emulation Command Files (Type 6)

Linker Command Files (Type 7)

Linker Configuration Files (Type 8)

Assembler Configuration Files (Type 10)

Incorrectly accessing these files may destroy them and cause serious system problems!

Table 8-6. Disc File I/O Codes

Request Name	User Program Request		64000 Response To:		
			Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
CREATE FILE	CA	80H	CA	00	01 thru 08, 10
	CA+1	File Type Number			09: file is not open
	CA+2	Disc #			11 thru 14: NA
OPEN FILE	CA	81H	CA	00	01 thru 08, 10
	CA+1	File Type Number			09: File is already open
	CA+2	Disc #			11 thru 14: NA
CLOSE FILE	CA	82H	CA	00	01 thru 08 09: File is already closed 10 thru 14: NA
DELETE FILE	CA	83H	CA	00	01 thru 08, 10
	CA+1	File Type Number			09: File not open

Table 8-6. Disc File I/O Codes (Cont'd)

Request Name	User Program Request		64000 Response To:		
			Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
					11 thru 14: NA
	CA+2	Disc #			
ADVANCE "N" RECORDS	CA	84H	CA	00	01 thru 08
	CA+1	LSB 15-bit* record			09: File not open
	CA+2	MSB number (*bit 16 not used)			10 thru 14: NA
BACKUP "N" RECORDS	CA	85H	CA	00	01 thru 08
	CA+1	LSB 15-bit* record			09: File not open.
	CA+2	MSB number (*bit 16 not used)			10 thru 14: NA
POSITION TO RECORD "N"	CA	86H	CA	00	01 thru 08
	CA+1	LSB 15-bit* record			09: File not open
	CA+2	MSB number (*bit 16 not used)			10 thru 14: NA
READ RECORD	CA	87H	CA	00	01 thru 08
	CA+1	Max. number of words user can accept. (128 words/ 256 bytes max.)	CA+1	Actual # of words read from buffer.	09: File is not open 12

Table 8-6. Disc File I/O Codes (Cont'd)

Request Name	User Program Request		64000 Response To:		
			Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
			CA+2 ↓ (CA+2) +n (*256 bytes/ 128 words is max. record length.)	Read Byte 1 ↓ Read Byte n *	10, 11, 13, 14: NA
REWIND TO RECORD ONE	CA	88H	CA	00	01 thru 08 09: File is not open 10 thru 14: NA
WRITE RECORD	CA	89H	CA	00	01 thru 08, 12
	CA+1	Number of words to be written. (128 words/ 256 bytes maximum.)			09: file is not open. 10, 11, 13, 14: NA
	CA+2 ↓ (CA+2) +n	Write byte 1 ↓ Write byte n			

Table 8-6. Disc File I/O Codes (Cont'd)

Request Name	User Program Request		64000 Response To:		
			Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
CHANGE FILE NAME	CA	8AH	CA	00	01 thru 08 12 & 15
SEE NOTE BELOW		Bits 7-5 specify length of file name in 16-bit words-1. Bits 4 & 3 specify ID length in 16-bit words. Bits 2-0 contain all zeros. (See note below.)			09: File not open 10, 11, 13, 14: NA
	CA+2	First character of file name. Limited to capital letters A thru Z.			
	CA+3	Second and following file name characters may be small or capital letters,			

Table 8-6. Disc File I/O Codes (Cont'd)

Request Name	**User Program Request**		**64000 Response To:**		
			Valid User Request		**Invalid Request**
	Address	**Contents**	**Address**	**Contents**	**Error Code**
		numerals 0 thru 9, underlines, and only if required one blank may be used to fill in last character in last word of name.			
	CA+4 thru CA+n. Where n 10	Up to 9 name characters may be used.			
	CA+ (n+1)	First USERID character.			
	CA+ (n+2) ↓ thru ↓ CA+16	Up to 6 USERID characters may be used. See note below.			

Note: The name and USERID characters must be packed into a fixed length record. This record consists of 8, 16-bit words. Thus, the name record will always require a user buffer consisting of 17 bytes (byte CA through byte CA+16). All unused 16-bit words must be at the end of the record. No intervening unused words or bytes are allowed. If the last byte in the last name and ID word is not required to define the name, then it must contain an ASCII blank. The byte in buffer location CA+1 must be formatted the same as described for the most significant byte of word 16 in the name and user ID word block of the microprocessor configuration record. Refer to the "microprocessor Configuration Record" in the Linker Symbols description for more information.

RS-232 I/O Interface

The following paragraphs describe the events which must be implemented between the user and the 64000 programs for RS-232 I/O to occur.

These events are:

- Open RS-232 File
- Initialize 8251
- Command To 8251
- Status From 8251
- Write To 8251

  Write Single Byte

  Write Record
- Read From 8251

  Read Single Byte

  Read Record
- Updating Read/Write Buffers

The above events, corresponding control codes, and parameters, where applicable, are summarized in table 8-7.

Open RS-232 File (80H)

Before any other RS-232 operation can be initiated, the user program must request that the RS-232 File be opened. This is done by placing code 80H into location CA.

NOTE

CA represents the location to which all RS-232 I/O "handshaking" codes are sent by both the user and the 64000 programs. The actual address for the RS-232 CA is defined in the users program and entered into the 64000 program during the configuration of the emulation CMDFILE. Each I/O interface - RS-232, display, printer, etc.- requires its own unique CA address.

Certain of the I/O codes sent to location CA must also include supplemental information. This supplemental information is contained in the locations following CA, i.e., CA+1 through CA+n. The supplemental information must be placed into locations CA+1 through CA+n BEFORE the corresponding control code is placed in CA. If this is not done, the 64000 may respond to the control code in CA before the supplemental data is set into locations CA+1 through CA+n.

The 64000 responds by opening the RS-232 file and returning a 00 to location CA to indicate that the file is open. If the file cannot be opened, error code 08 or 09 is returned to location CA.

After the file is opened, the 8251 must be initialized as described in the next paragraph.

Initialize 8251 (82H)

In general, 8251 initialization consists of resetting the 8251 and then selecting one of the following three operating modes: (1) asynchronous, (2) synchronous with one sync character, or (3) synchronous with two sync characters. (See figure 8-8.)

For each of the three modes, the user program requests initialization by first setting up buffer locations CA+1 through CA+5 and then placing code 82H into location CA. A command instruction with Internal Reset (IR) bit D6 set is placed into location CA+1. (See figure 8-9.) The contents placed into locations CA+2 through CA+5 depend upon the operating mode selected as described in the following paragraphs.

Asynchronous Mode - For this mode, the asynchronous mode instruction is placed into location CA+2 and a sync option word specifying 0 must be placed into location CA+3. Locations CA+4 and CA+5 contain no meaningful data.

The asynchronous mode instruction is used to select the baud rate*, the character length, the parity parameters, and the number of stop bits. (See figure 8-10.) (*The only baud rates which may be used with the 64000 are the transmitter clock frequency (I X Txc) or 1/16 X Txc. The baud rate factor of 1/64 X Txc cannot be used with the 64000. The basic frequency of Txc is selected by switches on the modem I/O card. Thus, the basic frequency (Txc) may be changed by the I/O card switches.) The user must format this instruction so that the appropriate parameters are specified. 1/16 X Txc must be programmed if the baud rate is to match the baud rate table in the System Overview manual.

The sync option specifies 0 since there are no sync characters for the asynchronous mode.

Synchronous Mode/Single Sync Character - For this mode, the synchronous mode instruction is placed into location CA+2, the sync option word specifying "1" is placed into location CA+3, and the sync character is placed into location CA+4. Location CA+5 contains no meaningful data. (See figure 8-8.)

The synchronous mode instruction is used to select the character length, and the parity and synchronization parameters. (See figure 8-11.) Bit D7 (SCS) of this word must specify a single sync character. The user must format this instruction so that the other appropriate parameters are specified.

The sync option word specifies "1" for a single sync character.

The format of the sync character must be defined by the user.

Synchronous Mode/Double Sync Character - For this mode, the synchronous mode instruction is placed into location CA+2, the sync option word specifying "2" is placed into location CA+3 and sync characters 1 and 2 are placed into locations CA+4 and CA+5, respectively. (See figure 8-8.)

The synchronous mode instruction is used to select the character length, and the parity and synchronization parameters. (See figure 8-11.) Bit D7 (SCS) of this word must specify a double sync character. The user must format this instruction so that the other appropriate parameters are specified.

The sync option word specifies "2" for double sync characters.

The format of both sync characters must be defined by the user.

After the 8251 is initialized, the 64000 returns a 00 to location CA. If it cannot be initialized, error code 08 or 09 is returned as shown in table 8-7.

Command to 8251 (83H)

After the 8251 is initialized (i.e., reset and asynchronous or synchronous operation selected), it must be placed in the appropriate mode - transmit, receive, or combination transmit/receive, etc. To do this, the user program first places the appropriately formatted command word into location CA+1 and then places code 83H into location CA. (The user must format the command word to select the applicable operation as shown in figure 8-9.)

The 64000 responds by supplying the command word to the 8251 and returning a 00 to location CA. If this cannot be done, code 08 or 09 is returned to location CA. (See table 8-7.)

Status from 8251 (84H)

The user may check the status of the 8251 at any time. To do this, code 84H is placed into location CA. The 64000 responds to this status request by returning a 00 to location CA and placing the 8251 status word in location CA+1.

The status word format is shown in figure 8-12.

The status bits D0, D1, and D2 may be cleared or set by the 64000 program when operating in any of the buffered modes. If the user wishes to use these bits to control operation, it is necessary to close the appropriate Tx or Rx buffers before relying on them.

Write to 8251

The user program may write to the 8251 in either of two ways. It may write a byte at a time, or it may set up a write buffer and write data continuously. Both methods are described. (Note: Before attempting to write data, the 8251 must be initialized and the command word, in the appropriate format, sent to the 8251 as described in the previous paragraphs.)

Write Single Byte (86H) - To write a single byte to the 8251, the user program first places the write byte into location CA+1 and then places code 86H into location CA. (See table 8-7.) The 64000 responds by supplying the byte to the 8251 and returning a 00 to location CA. If writing cannot be done, error code 08 or 09 is returned to CA. (See table 8-7.) If more data is to be sent, it is recommended that the user poll the 8251 status to determine if it is ready to receive more transmit data.

Write Record (87H), Update Write Buffer (89H) (see also Update Read/Write Buffer (8DH)) - To write a record to the 8251, the user program must first set up a write buffer and identify the beginning and ending locations in the buffer. (The corresponding 64000 write buffer holds a maximum of 256 bytes.) (See figure 8-13.) It then writes a record into the buffer and identifies the buffer locations into which the first and last bytes of the record are written.

The user program must then request that the record be transferred to the 8251. (See figure 8-14.) This is done by first placing the user write buffers beginning/ending and first/last byte address pointers into locations CA+7 through CA+14 and then placing code 87H into location CA.

The 64000 responds by transferring data from the users write buffer into a 64000 write buffer. (See figure 8-15.) For each byte transferred to the 64000 buffer, the first byte address pointer (in locations CA+11 and CA+12) is incremented by one. Data transfer continues until either all data in the users write buffer is transferred or the 64000 write buffer becomes full. (The 64000 write buffer holds a maximum of 256 bytes, or 128 words.) After a write buffer is set up and if update code 8DH or 89H is used, then the number of bytes actually transmitted by the 8251 is also entered into location CA+6 by the 64000 program. The number of bytes transmitted refers to the number of bytes transmitted from the 64000 buffer.

The user program should periodically examine the first and last address byte pointers (and if using update code 8DH or 89H, the number of bytes transmitted by the 8251 may also be examined) to determine the status of the buffer. (If the first and last byte pointers are equal, all data was transferred to the 64000 buffer.)

If all data was transferred, the user program may either supply another write record, or close the write buffer. If all data was not transferred, the user program may either wait until the remaining data is transferred, add more data to the buffer and update the last byte pointer, or close the write buffer. Each of these options is described in the following paragraphs.

Additional data may be added to, or a new record written into the buffer and the last byte address pointer updated as follows: If the first and last byte address pointers are pointing to the same location, the first new byte goes into the location pointed to by both pointers. If the first and last byte address pointers are not pointing to the same location, then the first new byte goes into the location just ahead of the one pointed to by the last byte address pointer (i.e., last byte address pointer + 1). Then the following bytes are entered into succeeding locations. (See figure 8-15.)

After entering data into the buffer,the user program requests write data transfer. This is done by first placing the updated last byte address pointer into locations CA+13 and CA+14 and then placing code 89H into location CA. (See figure 8-16.)

The 64000 responds by transferring data from the users write buffer to the 64000 write buffer, increments the first byte address pointer for each byte transferred, and if update code 8DH or 89H is being used, the number of bytes sent by the 8251 is also updated.

Once the user program has placed code 8DH or 89H (update buffer) into location CA, the 64000 routinely monitors the last byte address pointer to determine if more data has been loaded into the users write buffer. If the 64000 detects that the last byte address pointer has been incremented, it transfers the data and increments the first byte address pointer to indicate the number of bytes written. It also updates the number of bytes sent by the 8251.

To write another record, the user program updates the last address pointer. The 64000 responds as described above.

To close the buffer, the user program places code 88H in location CA. The 64000 closes the write buffer and returns a 00 to location CA.

Data may be stored in the users write buffer using a "wrap around" method. That is, once the last location in the buffer is filled, the next byte is placed into the first location of the buffer. Thus, it is possible for the last byte address pointer to be pointing to an address that is less than (i.e., ahead of) the first byte address.

If any of the write buffer requests cannot be done, the 64000 returns the appropriate error code to location CA as shown in table 8-7.

Read from 8251

Reading data from the 8251 is similar to writing data to the 8251. The user program may read data in either of two ways. It may read a byte at a time or it may set up a read buffer and read a record at a time. Both methods are described. Note: Before attempting to read data, the 8251 must have been initialized and the command word, in the applicable format, sent to the 8251 as described in the previous paragraphs.

Read Single Byte (85H) - To read a single byte from the 8251, the user program places code 85H into location CA. (See table 8-7.)

The 64000 responds by returning a 00 to location CA and the read byte to location CA+1. If reading cannot be done, error code 08 or 09 is returned to CA.

The 64000 will return whatever character is in the Rx buffer of the 8251. It is recommended that the user check the status of the 8251 to see if Rx RDY is true before performing the single byte read. Any read operation will clear Rx RDY, indicating that the character in the buffer has been read.

Read Record (8AH), Update Read Buffer (8CH) (see also Update Read/Write Buffer (8DH)) - To read a record from the 8251, the user program must first set up a read buffer and identify the beginning and ending locations in the buffer. (See figure 8-17.)

This is done by first placing the address pointers into locations CA+16 through CA+23 and then placing code 8AH into location CA. Locations CA+16 through CA+19 contain the address pointers for the beginning and ending locations of the users read buffer. Locations CA+20 through CA+23 contain the address pointers for the first and last bytes written into the buffer. These pointers are both initially set to point to the first location in the users read buffer. This indicates that the buffer is empty. (The 64000 will force the first data pointer to always point to the beginning of the buffer.)

The 64000 responds by continuously transferring read data from the 8251 to the 64000 read buffer. (See figure 8-19.) The user program must then issue an 8CH or 8DH to transfer the data to the users buffer. For each byte transferred into the users read buffer, the last byte address pointer is incremented by one (see figure 8-18). In addition, when update code 8DH or 8CH is being used, the number of bytes received by the 8251 and transferred into the 64000 is entered into location CA+15.

To determine when and how much read data is available, the user program must monitor the last byte address pointer and the number of bytes received. When it detects that read data is in the buffer, the user program should process the data. If all data expected was received, the user program may then close the read buffer.

Once the user program has placed code 8CH of 8DH into location CA, the 64000 periodically monitors the output of the 8251, transfers data into the user read buffer, and updates the last byte address as required. The user program in turn monitors the last byte address pointer to determine if more data is available. This process continues until the user program closes the read buffer.

If code 8CH or 8DH is being used, and the user issues an 8AH again, the buffer is frozen for the user, yet the 64000 continues to receive data into its buffer.

To close the read buffer, the user program places code 8BH into location CA. The 64000 closes the buffer and returns a 00 to location CA.

Data may be stored in the user's read buffer using a "wrap around" method. That is, once the last location in the buffer is filled, the next byte is placed into the first location of the buffer. Thus, it is possible for the last byte address pointer to be pointing to an address that is less than (i.e., ahead of) the first byte address.

If any of the read buffer requests cannot be done, the 64000 returns the appropriate error code to location CA as shown in table 8-7.

Updating Read/Write Buffers (8DH)

Once the read and write buffers have been set up and opened as described in preceding paragraphs "Write to 8251" and "Read from 8251", the buffers may both be updated by using one code. To do this, the user program places the updated first and last byte address pointers for both the read and write buffers into the corresponding locations in the RS-232 I/O control buffer and then places code 8DH into location CA.

The 64000 responds to the update request as described in the "Write to 8251" and "Read from 8251" paragraphs. However, in addition to setting, monitoring, and updating the first and last byte address pointers, the number of bytes received and transmitted by the 8251 is also set, updated, and monitored. This provides an additional indication of how much data has been sent and received.

Table 8-7. RS-232 I/O Codes

Request Name	User Program Request		64000 Response To: Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
OPEN RS-232 FILE	CA	80H	CA	00	01-07: NA 08 09: File already open. 10-14: NA
CLOSE RS-232 FILE	CA	81H	CA	00	01-07: NA 08 09: File not open. 10-14: NA
INITI-ALIZE 8251	CA	82H	CA	00	Same as 81H, above
	CA+1	Command Instruction			
	CA+2	Mode In-struction			
	CA+3	Sync Op-tion word			
	CA+4	Sync Char-acter,one			
	CA+5	Sync Char-acter,two			

Table 8-7. RS-232 I/O Codes (Cont'd)

Request Name	User Program Request		64000 Response To:		
			Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
COMMAND TO 8251	CA CA+1	83H Command Word	CA	00	Same as 81H, above
STATUS FROM 8251	CA	84H	CA CA+1	00 Status Word	Same as 81H, above
READ SINGLE BYTE FROM 8251	CA	85H	CA CA+1	00 Byte Read	Same as 81H, above
WRITE SINGLE BYTE TO 8251	CA CA+1	86H Write Byte	CA	00	Same as 81H, above
OPEN WRITE BUFFER	CA CA+1 ↓ CA+5	87H Reserved for Initialization buffer	CA The 64000 transfers write data from the users buffer to the 64000 buffer.	87H	

Table 8-7. RS-232 I/O Codes (Cont'd)

Request Name	User Program Request		64000 Response To:		
			Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
	CA+6	#Bytes sent by 8251. Cleared by open (87H).	For each byte transferred to the 64000 buffer, first byte address pointer is incremented by one.		
	CA+7 (lsb) CA+8 (msb)	Updated by 64000 when update code 89H or 8DH is used.			
	CA+9 (lsb)	Buffer Begin Address pointer			
	CA+10 (msb)	Buffer End Address pointer			
	CA+11 (lsb) CA+12 (msb)	First Byte Address pointer			
	CA+13 (lsb) CA+14 (msb)	Last Byte Address Pointer			

Table 8-7. RS-232 I/O Codes (Cont'd)

Request Name	User Program Request		64000 Response To:		
			Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
CLOSE WRITE BUFFER	CA	88H	CA	00	Same as 81H, above.
UPDATE WRITE BUFFER	CA	89H	CA	89H	Same as 81H, above.
	CA+1 CA+5	Reserved for Initialization Buffer	The user updates the last byte address Pointer to indicate how much new write data is in the buffer. The 64000 processes the write data, increments the first byte addr. pointer, and updates # bytes sent by 8251 as required.		
	CA+6	# Bytes sent by 8251.			
	CA+7 CA+10	Not changed by user.			
	CA+11 (lsb) CA+12 (msb)	First Byte Address Pointer			
	CA+13 (lsb) CA+14 (msb)	Updated last byte Address Pointer			

Table 8-7. RS-232 I/O Codes (Cont'd)

Request Name	User Program Request		64000 Response To:		
			Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
OPEN READ BUFFER	CA	8AH	CA	8AH	Same as 81H, above
	CA+1 ↓	Reserved for Initialization and write buffers.	The user sets first and last address pointers to point to buffer beginning address. The 64000 will transfer data from the 8251 to the 64000 buffer. The user must use the commands 8CH or 8DH to transfer the data to the users buffer.		
	CA+14 CA+15 CA+16 (lsb) CA+17 (msb) CA+18 (lsb)	# Bytes received by 8251. Cleared by open (8AH). Updated by 64000 when update code 8CH or 8DH is used.			
	CA+19 (msb)	Buffer Begin Address Pointer.			
	CA+20 (lsb) CA+21 (msb)	Buffer End Address Pointer			
	CA+22 (lsb) CA+23 (msb)	First Byte Address Pointer			
		Last Byte Address Pointer			

Table 8-7. RS-232 I/O Codes (Cont'd)

Request Name	**User Program Request**		**64000 Response To:**		
			Valid User Request		**Invalid Request**
	Address	**Contents**	**Address**	**Contents**	**Error Code**
CLOSE READ BUFFER	CA	8BH	CA	00	Same as 81H above
UPDATE READ BUFFER	CA	8CH	CA	80H	Same as 81H
	CA+1 ↓ CA+14	Reserved for Initialization and write buffers.			
	*CA+15	#Bytes received by 8251.	The 64000 continues to transfer data, increments last byte address pointer, (updates #Bytes received by 8251) as required. User program monitors these parameters to determine how much data is received. (64000 forces first byte address pointer to always point to the beginning of the buffers.)		
	CA+16 CA+19	Not changed by user.			
	CA+20 (lsb) CA+21 (msb)	First Byte Address Pointer.			
	CA+22 (lsb) CA+23 (msb)	Last Byte Address Pointer.			

Table 8-7. RS-232 I/O Codes (Cont'd)

Request Name	User Program Request		64000 Response To:		
			Valid User Request		Invalid Request
	Address	Contents	Address	Contents	Error Code
UPDATE WRITE/ READ BUFFERS	CA	8DH	CA	8DH	Same as 81H above
	CA+1 ↓ CA+5	Reserved for Initialization Buffer	Write and read buffers are both updated as described above.		
	CA+6 ↓ CA+14	Same as shown for update Write Buffer, above.			
	CA+15 ↓ CA+23	Same as shown for update Read Buffer, above.			

ADDRESS	ASYNCHRONOUS MODE - INITIALIZATION FORMAT	SYNCHRONOUS MODE - SINGLE SYNC CHARACTER INITIALIZATION FORMAT	SYNCHRONOUS MODE - DOUBLE SYNC CHARACTER INITIALIZATION FORMAT	ADDRESS
CA	82H - INITIALIZE 8251	82H - INITIALIZE 8251	82H - INITIALIZE 8251	CA
CA+1	COMMAND INSTRUCTION (Internal Reset 8251)	COMMAND INSTRUCTION (Internal Reset 8251)	COMMAND INSTRUCTION (Internal Reset 8251)	CA+1
CA+2	ASYNCHRONOUS MODE INSTRUCTION	SYNCHRONOUS MODE INSTRUCTION	SYNCHRONOUS MODE INSTRUCTION	CA+2
CA+3	SYNC OPTION WORD Ø = No sync characters	SYNC OPTION WORD 1 = 1 sync character	SYNC OPTION WORD 2 = 2 sync characters	CA+3
CA+4	Not Used.	SYNC CHARACTER 1	SYNC CHARACTER 1	CA+4
CA+5	Not Used.	Not Used.	SYNC CHARACTER 2	CA+5
CA+6 ↓ CA+14	RESERVED FOR WRITE CONTROL	RESERVED FOR WRITE CONTROL	RESERVED FOR WRITE CONTROL	CA+6 ↓ CA+14
CA+15 ↓ CA+23	RESERVED FOR READ CONTROL	RESERVED FOR READ CONTROL	RESERVED FOR READ CONTROL	CA+15 ↓ CA+23

Figure 8-8. 8251 Initialization Formats

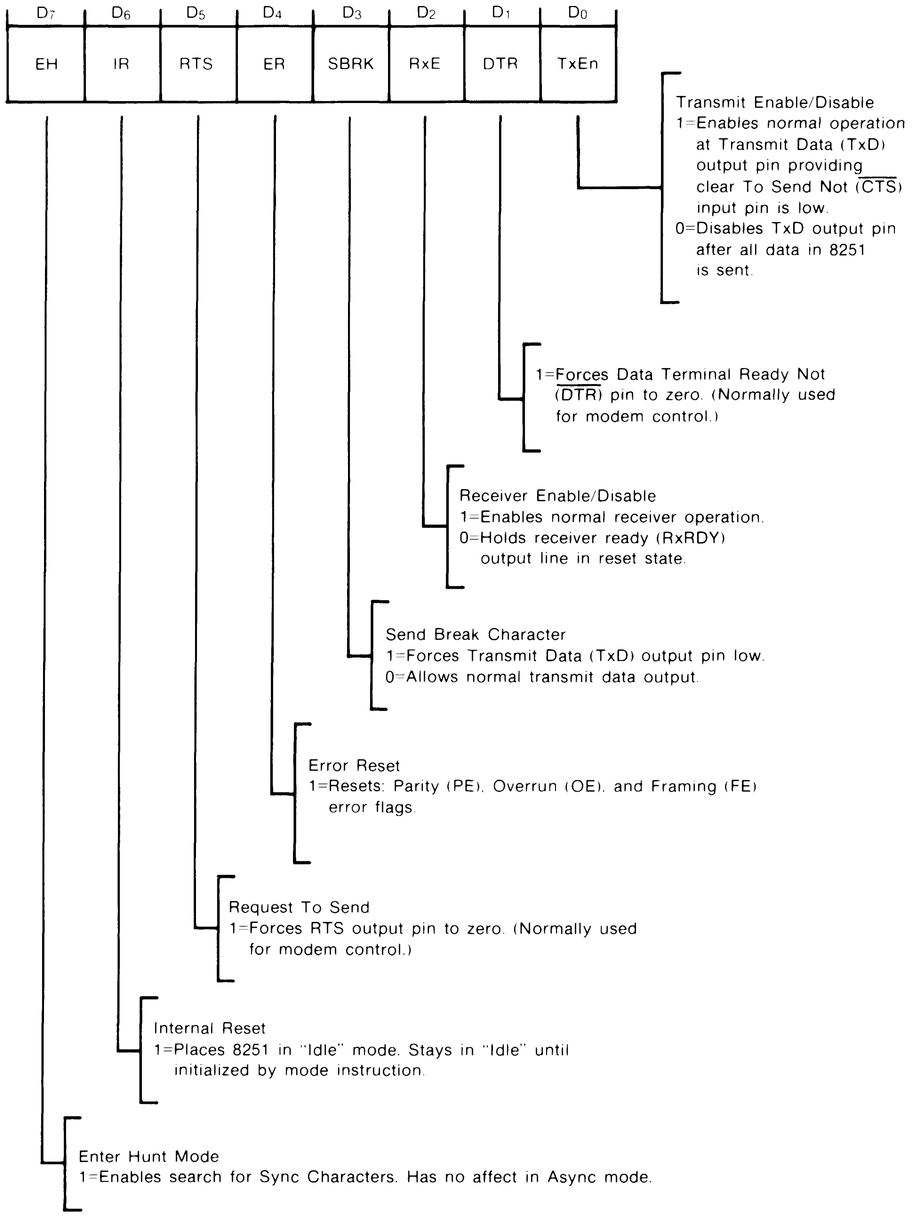

D7 D6 D5 D4 D3 D2 D1 D0
EH IR RTS ER SBRK RxE DTR TxEn
Transmit Enable/Disable
1=Enables normal operation at Transmit Data (TxD) output pin providing clear To Send Not ($\overline{CTS}$) input pin is low.
0=Disables TxD output pin after all data in 8251 is sent.
1=Forces Data Terminal Ready Not ($\overline{DTR}$) pin to zero. (Normally used for modem control.)
Receiver Enable/Disable
1=Enables normal receiver operation.
0=Holds receiver ready (RxRDY) output line in reset state.
Send Break Character
1=Forces Transmit Data (TxD) output pin low.
0=Allows normal transmit data output.
Error Reset
1=Resets: Parity (PE), Overrun (OE), and Framing (FE) error flags.
Request To Send
1=Forces RTS output pin to zero. (Normally used for modem control.)
Internal Reset
1=Places 8251 in "Idle" mode. Stays in "Idle" until initialized by mode instruction.
Enter Hunt Mode
1=Enables search for Sync Characters. Has no affect in Async mode.

Figure 8-9. Command Mode Instruction Format

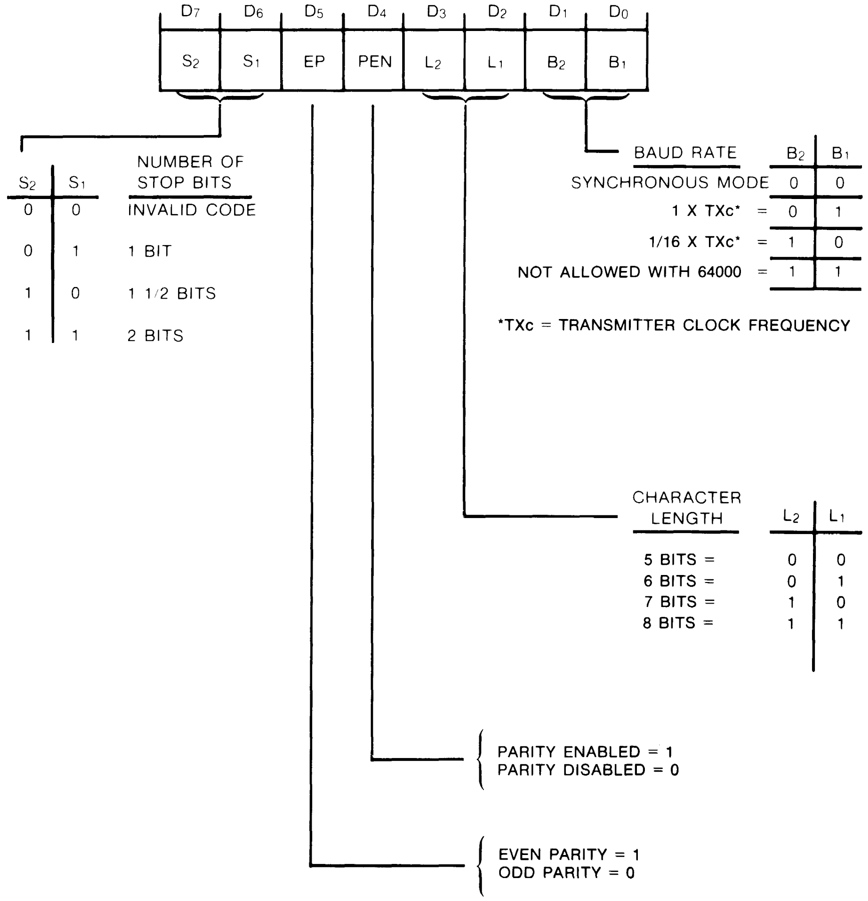


Figure 8-10. Asynchronous Mode Instruction Format

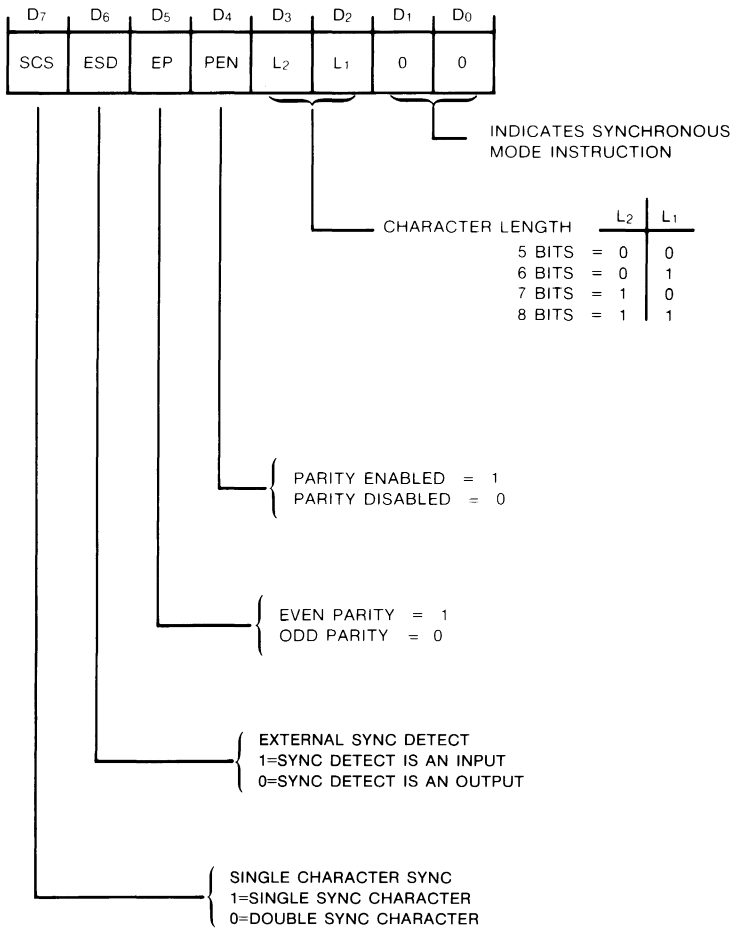


Figure 8-11. Synchronous Mode Instruction Format

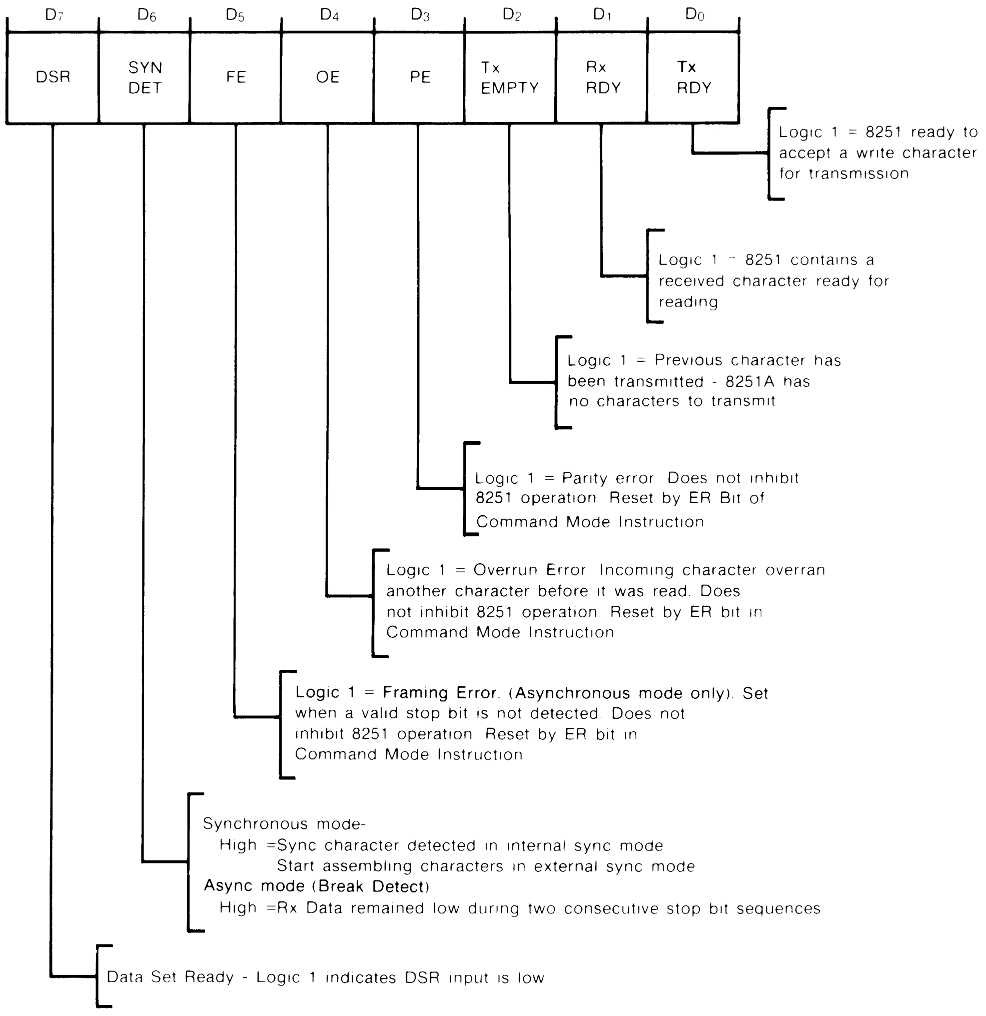

D_7 DSR
D_6 SYN DET
D_5 FE
D_4 OE
D_3 PE
D_2 Tx EMPTY
D_1 Rx RDY
D_0 Tx RDY
Logic 1 = 8251 ready to accept a write character for transmission.
Logic 1 = 8251 contains a received character ready for reading
Logic 1 = Previous character has been transmitted - 8251A has no characters to transmit
Logic 1 = Parity error. Does not inhibit 8251 operation. Reset by ER Bit of Command Mode Instruction
Logic 1 = Overrun Error. Incoming character overran another character before it was read. Does not inhibit 8251 operation. Reset by ER bit in Command Mode Instruction
Logic 1 = Framing Error. (Asynchronous mode only). Set when a valid stop bit is not detected. Does not inhibit 8251 operation. Reset by ER bit in Command Mode Instruction
Synchronous mode-
High =Sync character detected in internal sync mode
Start assembling characters in external sync mode
Async mode (Break Detect)
High =Rx Data remained low during two consecutive stop bit sequences
Data Set Ready - Logic 1 indicates DSR input is low

Figure 8-12. 8251 Status Word Format

Phase I - User Sets Up Write Buffer

User sets up write buffer as follows:

1. Assigns buffer beginning and ending addresses: WBUFBEG and WBUFEND.

2. Writes block of characters into buffer shown as first byte through last byte.

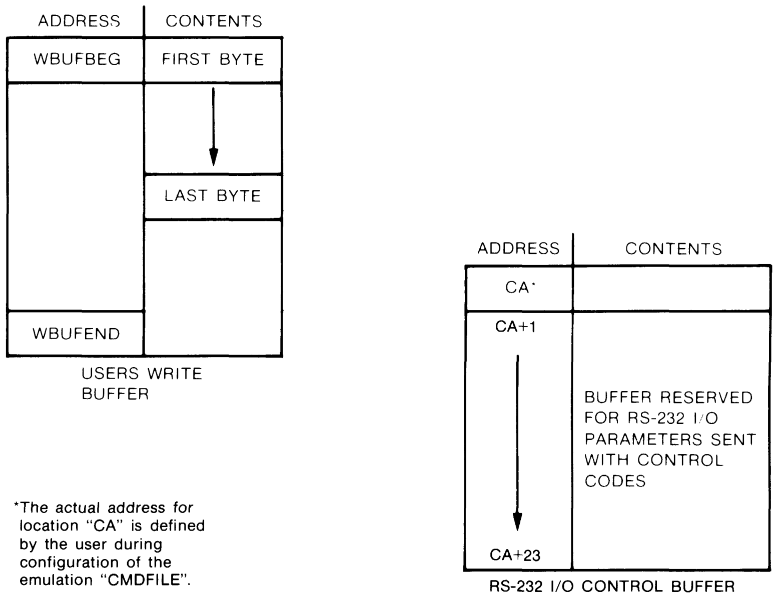

ADDRESS
CONTENTS
WBUFBEG
FIRST BYTE
LAST BYTE
WBUFEND
USERS WRITE BUFFER
*The actual address for location "CA" is defined by the user during configuration of the emulation "CMDFILE".
ADDRESS
CONTENTS
CA*
CA+1
CA+23
BUFFER RESERVED FOR RS-232 I/O PARAMETERS SENT WITH CONTROL CODES
RS-232 I/O CONTROL BUFFER

Figure 8-13. Writing RS-232 Record - Phase I

Phase II - User Requests Write Data Transfer

User initiates RS-232 Write Data Transfer via 64000 by supplying pointers to user write buffer as shown below and then placing code 87H into location CA.

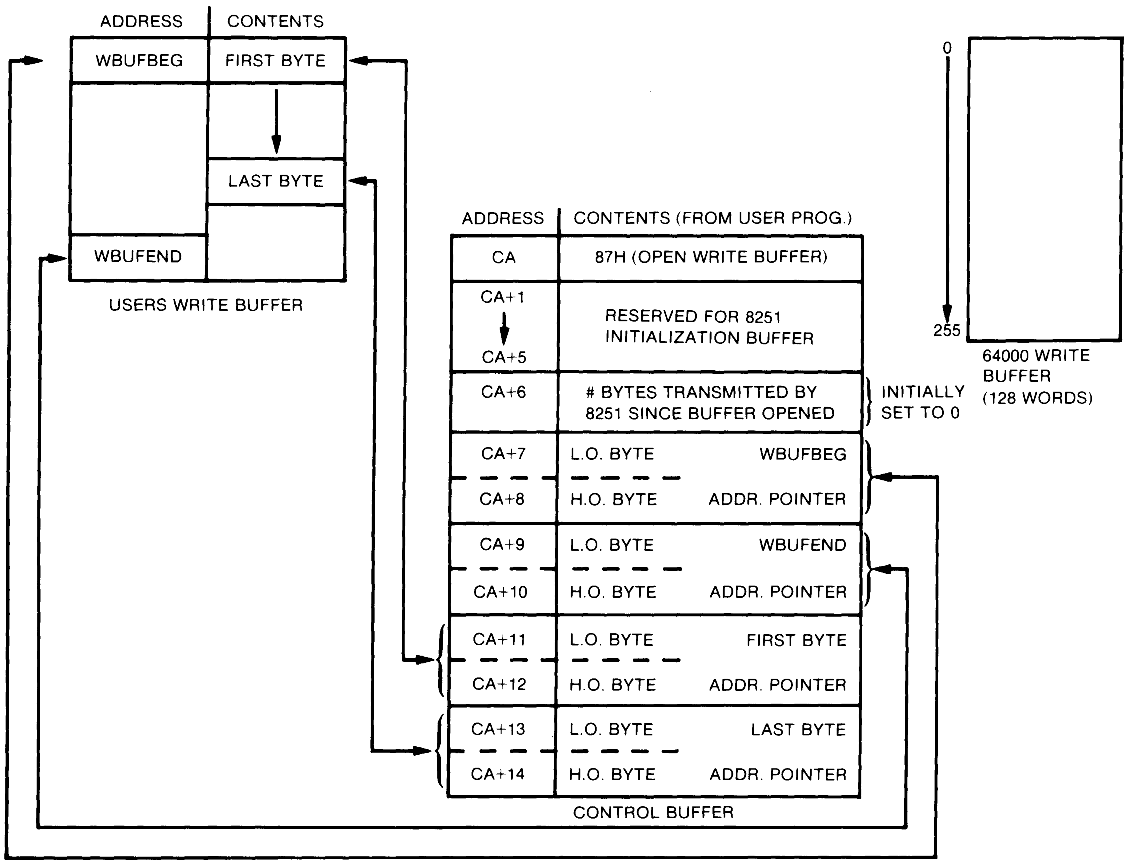


Figure 8-14. Writing RS-232 Record - Phase II

Phase III - 64000 Response To Write Data Request

64000 transfers data from user to 64000 buffer. For each byte transferred, the "FIRST BYTE" address pointer is incremented by one. The user program monitors the first byte address pointer and # bytes transmitted to determine when and how much data is transferred. If all data was transferred, the "FIRST BYTE" and "LAST BYTE" address pointers are equal. If not equal, the difference is the number of bytes yet to be transferred. The user program may either update (see Phase IV) or close the buffer.

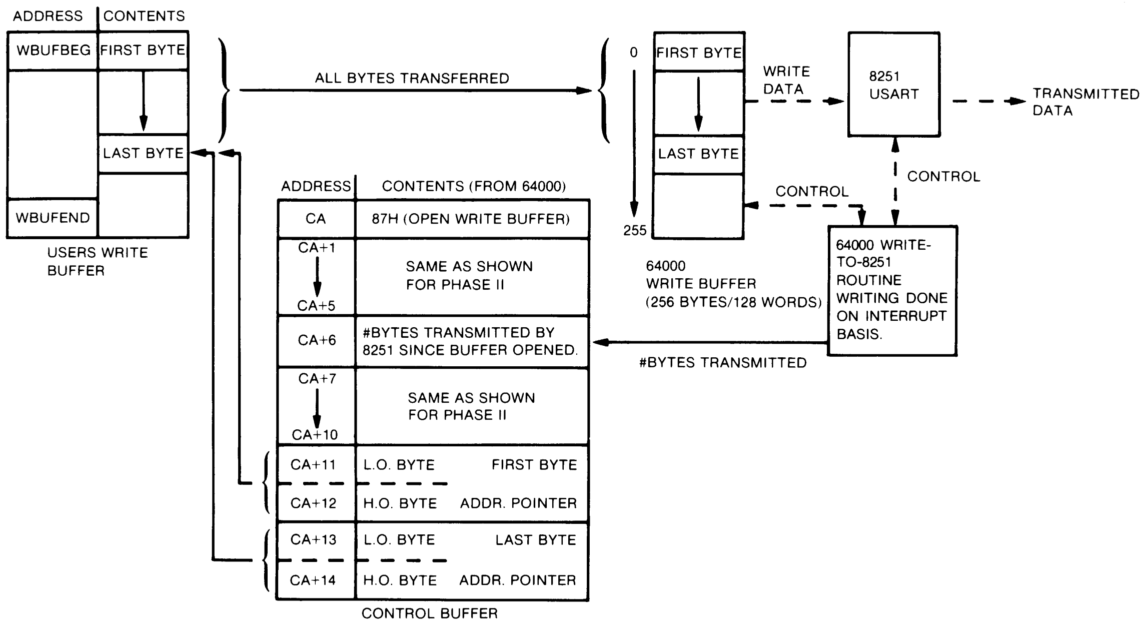


Figure 8-15. Writing RS-232 Record - Phase III

Phase IV - User Updates Write Buffer

The user program places more data into the write buffer and updates the last byte address pointer. User then requests more write data transfer by placing code 89H into CA. The 64000 response is similar to Phase III. Once 89H is in CA, the 64000 periodically monitors last byte address pointer to detect if more data has been placed into buffer by user. The 64000 transfers data and increments first byte address pointer and number of bytes sent by 8251 as required. Process continues until user program closes buffer.

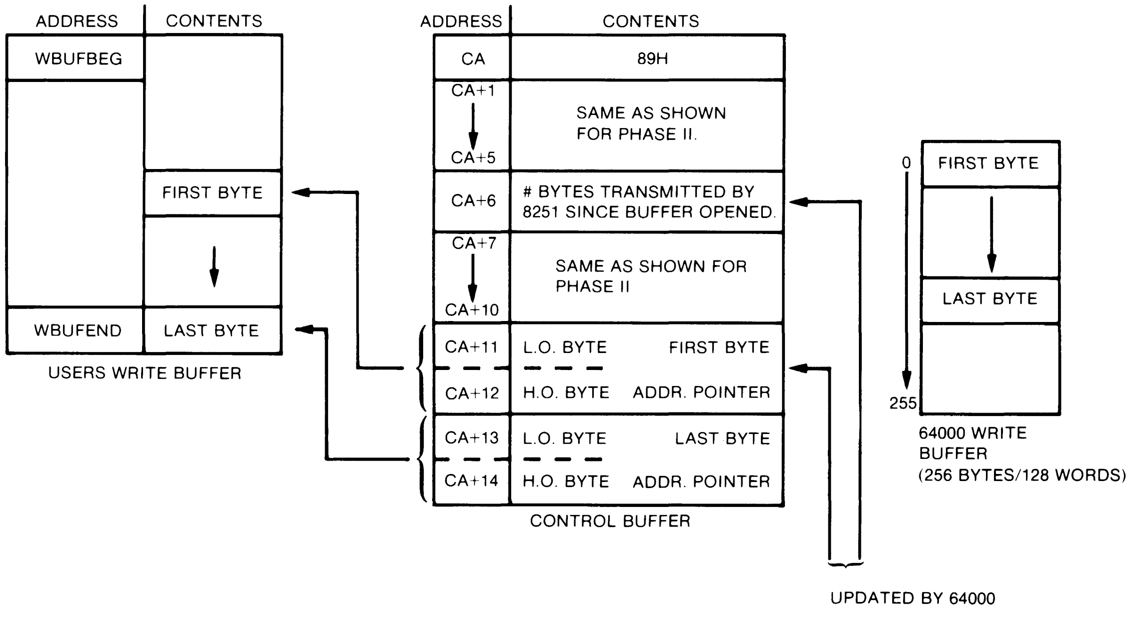


Figure 8-16. Writing RS-232 Record - Phase IV

Phase I - User Sets Up Read Buffer

User sets up read buffer and assigns beginning and ending addresses: RBUFBEG and RBUFEND.

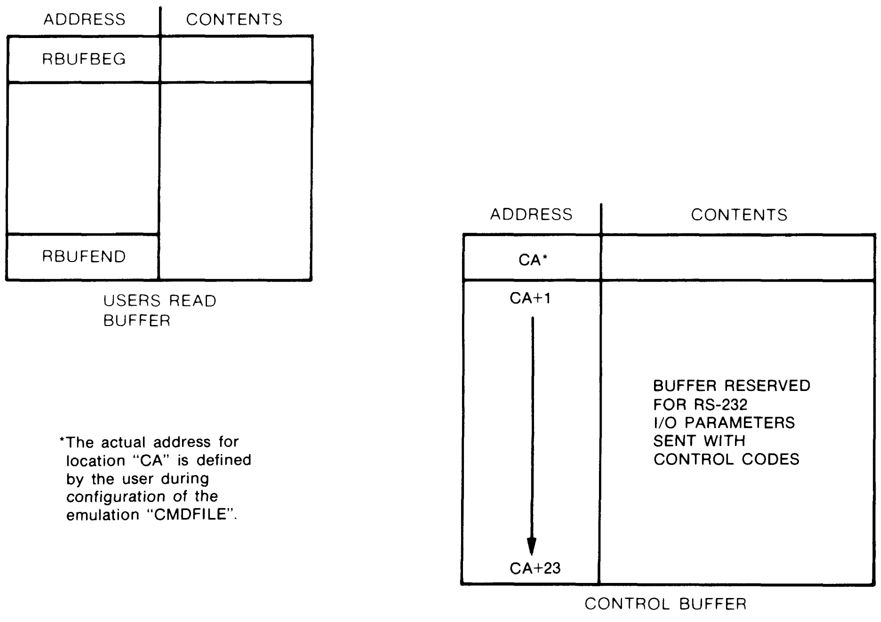

ADDRESS
CONTENTS
RBUFBEG
RBUFEND
USERS READ BUFFER
*The actual address for location "CA" is defined by the user during configuration of the emulation "CMDFILE".
ADDRESS
CONTENTS
CA*
CA+1
CA+23
BUFFER RESERVED FOR RS-232 I/O PARAMETERS SENT WITH CONTROL CODES
CONTROL BUFFER

Figure 8-17. Reading RS-232 Record - Phase I

Phase II - User Requests Read Data Transfer

User enables RS-232 read data transfer via 64000 by:

(1) Supplying pointers to user's read buffer as shown below. (First and last pointers point to same location to indicate buffer is empty.)

(2) Then placing code 8AH into location CA. The 64000 will begin receiving data into it's buffer. The user must do an 8CH or 8DH command to view the received data.

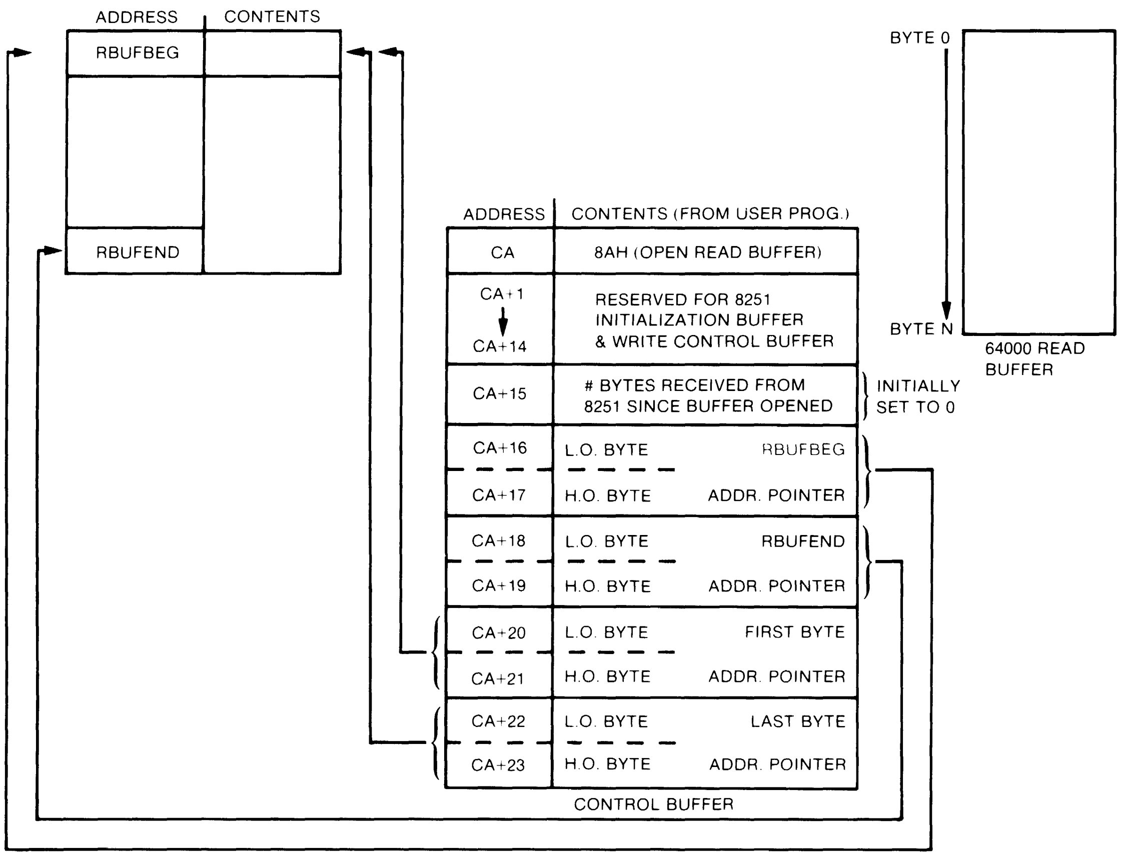


Figure 8-18. Reading RS-232 Record - Phase II

Phase III - 64000 Response to Read Data Request

The 64000 program transfers data from 8251 into user's read buffer via the 64000 read buffer.

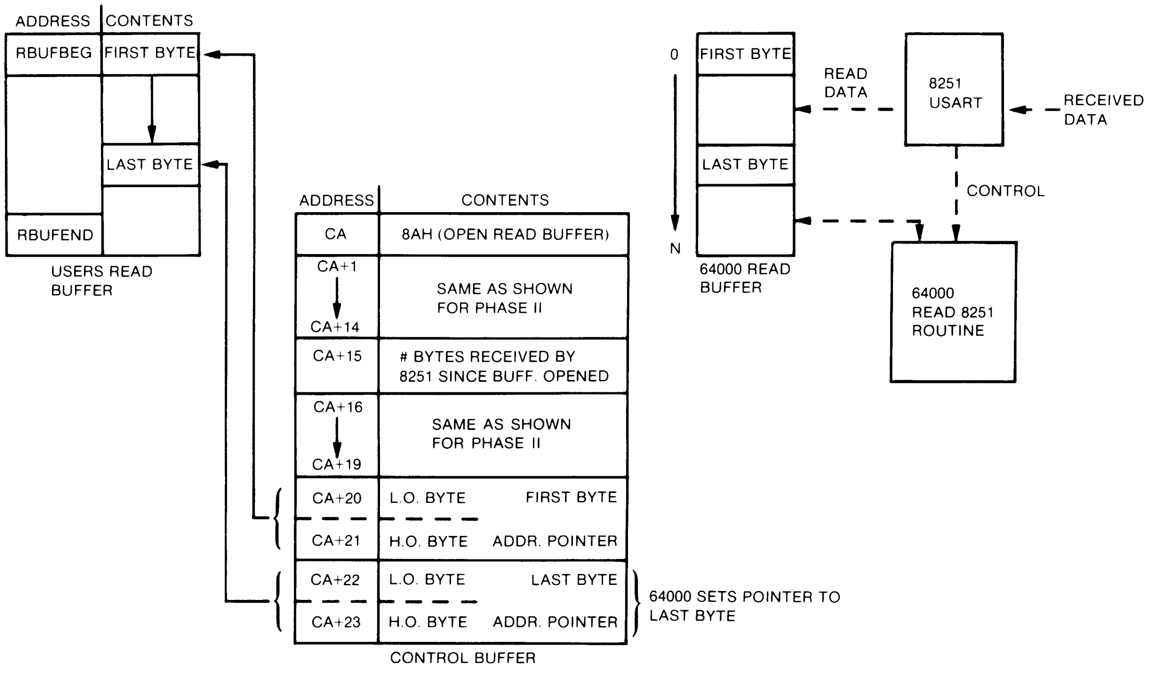


Figure 8-19. Reading RS-232 Record - Phase III

Phase IV - User Updates Read Buffer

The user requests more read data by updating first byte address pointer and by placing code 8CH or 8DH into CA. The 64000 response is similar to that shown for Phase III. Once 8CH or 8DH is in CA, the 64000 periodically transfers data from 8251 into user's read buffer, updates last byte address pointer and # bytes received by 8251 as required. When user program has received all data expected, it may close read buffer by placing 8BH in CA. The 64000 forces first byte address pointer to always point to the beginning of the user buffer.

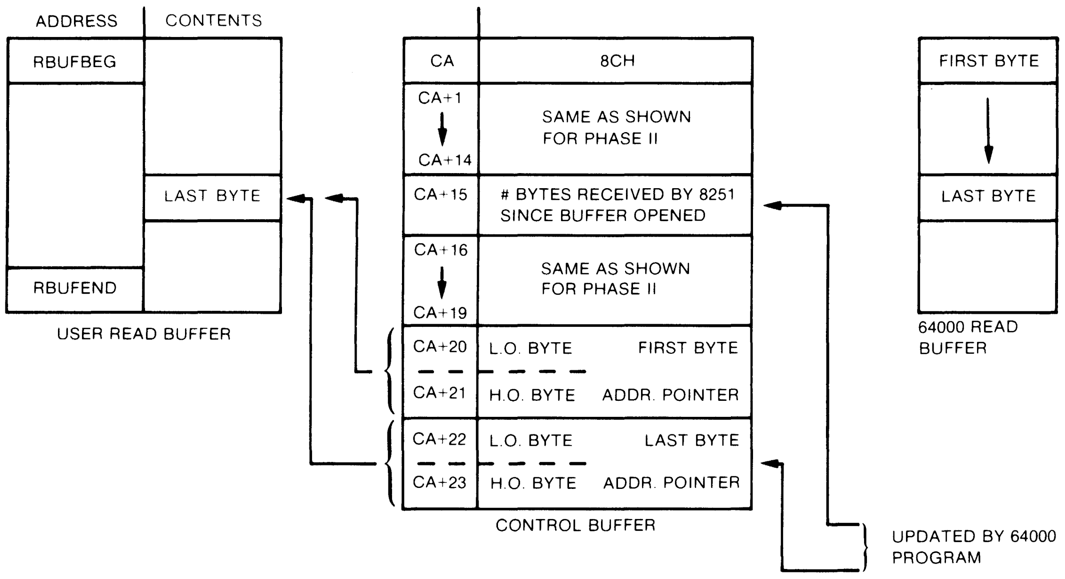


Figure 8-20. Reading RS-232 Record - Phase IV

Simulated I/O Error Codes

The general definitions for the simulated I/O error codes are listed in table 8-8. Where applicable, more specialized definitions of these error codes are listed in individual I/O code tables, 8-1, 8-2, etc.

When a request by the user program cannot be executed, the applicable error code is returned by the 64000 program to location CA.

Table 8-8. Simulated I/O Error Codes-General Definitions

Decimal Code #	**(Hex)**	**Meaning**
00		No error - successful operation
01		End of file
02		Invalid disc
03		File not found
04		File already exists
05		No disc space available
06		No directory space available
07		File is Corrupt (bad linkage)
08		Cannot read/write assigned memory
09		Request not allowed
10	(A)	Invalid file type
11	(B)	Invalid row or column no.
12	(C)	Invalid record length
13	(D)	Invalid display character >OFOH
14	(E)	While in simulated display I/O or simulated keyboard I/O, the 64000 "simulate" soft key was pressed to exit simulate I/O. All open files are closed.
15	(F)	Error in new disc file name when attempting to change a disc file name. First character in file name limited to capital letters A through Z. Second and following characters may contain capital and lower case letters, numerals 0 through 9, underlines, and only if required to fill in the last byte of the last word, a blank is used.

Simulated I/O Sample Programs

The following figures show the listing for source programs which actually use simulated I/O facilities. The programs are real and do work.

Figure #	**Sample Program Type**
8-21	Simulated Display I/O - Sample Program A
8-22	Simulated Display I/O - Sample Program B
8-23	Simulated Keyboard, Display, and One Disc File I/O - Sample Program
8-24	Simulated Keyboard, Display, and Two Disc Files I/O - Sample Program

```
"8080"
*This 8080 program uses the simulated display I/O interface.
*The display is opened, and two messages are written; one to
*      row/column 2,40 and one to row/column 18,20.
*Control address for display is 0000H.
*Program execution should start at 0200H.

                    ORG 200H
USR_ADR EQU 0
START               LXI SP,100H
*OPEN               DISPLAY
                    MVI A,80H
                    STA USR_ADR
                    CALL WAIT
*SET                ROW/COLUMN 2/40
                    MVI A,2
                    STA USR_ADR+1
                    MVI A,40
                    STA USR_ADR+2
                    MVI A,83H
                    STA USR_ADR
                    CALL WAIT
```

Figure 8-21. Simulated Display I/O - Sample Program A

```
*WRITE MESSAGE 1
                    LXI H,MESSAGE_1
                     CALL XFR_MSG
                       CALL WAIT
*SET ROW/COLUMN 18/20
                    MVI A,18
                    STA USR_ADR+1
                    MVI A,20
                    STA USR_ADR+2
                    MVI A,83H
                    STA USR_ADR
                    CALL WAIT
*WRITE MESSAGE 2
                    LXI H,MESSAGE_2
                    CALL XFR_MSG
                    CALL WAIT
*LOOP HERE, LEAVE DISPLAY OPEN
                    JMP $

*WAIT FOR USR_ADR=0 ... IO REQUEST COMPLETED
WAIT
                    LDA USR_ADR
                    CPI 0
                    JM WAIT
                    RET
*TRANSFER MESSAGE FROM C(DE) TO USR_ADR+1
*C(DE(0))=#BYTES
*      C(DE(1))=BYTE 1
*          .
*          .
*          .
*      C(DE(N))=BYTE N
*USR_ADR=84H ... REQUEST TO WRITE ROW/COLUMN
```

Figure 8-21. Simulated Display I/O - Sample Program A (Cont'd)

```
XFR_MSG
                  XCHG
                  LDAX D
                  MOV B,A            ;B=#BYTES
                  INX D
                  LXI H,USR_ADR+1    ;HL=USR_ADR+1
                  INX H
                  MOV M,A
                  INX H
XFR_LOOP
                  LDAX D             ;GET DATA BYTE
                  MOV M,A            ;AND STORE IN USR_ADR BUFFER
                  INX D
                  INX H
                  DCR B              ;AND CHECK FOR COMPLETION
                  MOV A,B
                  CPI 0
                  JNZ XFR_LOOP
                  MVI A,84H          ;SET REQ FOR WRITE ROW/COLUMN
                  STA USR_ADR
                  RET

*MESSAGE 1

MESSAGE_1
                  DB 12              ;#BYTES
                  ASC "Display test"

*MESSAGE 2

MESSAGE_2
                  DB 8               ;#BYTES
                  ASC "End test"
                  END START
```

Figure 8-21. Simulated Display I/O - Sample Program A (Cont'd)

This program scrolls ASCII characters onto the 64000 Display.

```
"8080"
DISP          EQU     0D00H     ;CONTROL ADDRESS FOR
                                ; SIMULATED DISPLAY IO
START         LXI     SP,1000H  ;STACK DOWN FROM 0FFFH
              LXI     H,DISP    ;LOAD H WITH CA
              CALL    CHECK     ;WAIT FOR SERVICE
              MVI     B,0       ;INITIALIZE COUNTER
LOOP          INX     H         ;SET M TO DISP+1
              MVI     M,4       ;SET BYTE COUNT TO FOUR
                                ; CHARACTERS/LINE
              INX     H         ;SET M TO DISP+2
                                ;THEN EACH LINE IS:
              MVI     M,32      ;ASCII BLANK (1ST CHAR)
              INX     H
              INR     B
                                ;THEN:
              MOV     M,B       ;2ND CHAR
              INX     H         ;THEN
              MVI     M,32      ;ANOTHER BLANK (3RD CHAR)
              INX     H
              MVI     M,0       ;AND A "NULL" (4TH CHAR)
              LXI     H,DISP
              MVI     M,82H     ;REQ "SCROLL" 1 LINE
              CALL    CHECK     ;WAIT FOR SERVICE
              MVI     A,127     ;AVOID SPECIAL CHARACTERS
              CMP     B         ;IF B > 127 RESET B TO 0
              JNZ     LOOP
              MVI     B,0
              JMP     LOOP
CHECK         LDA     DISP
              CPI     0         ;WAIT UNTIL CA=0
              RZ
              JMP     CHECK
              END     START
```

Figure 8-22. Simulated Display I/O - Sample Program B

```
"8080"
          ORG 0
* THIS PROGRAM OPENS KEYBOARD AND DISPLAY FILE. THEN, UPON
* CARRIAGE RETURN IT COPIES KEYBOARD DATA TO DISPLAY AND FILE
* FIO.

* OPEN DISPLAY AND KEYBOARD
START
          LXI SP,400H
          MVI A,80H           ;OPEN DISPLAY
          STA DSP_BUF
          STA KEY_BUF
          MVI A,02H           ;DELETE FILE
          STA FILE_BUF+1      ;TYPE 2(SCR)
          MVI A,0             ;DISC#0
          STA FILE_BUF+2
          MVI A,83H
          STA FILE_BUF
WAIT_FILE_D                   ;WAIT FR FILE DELETE
          LDA FILE_BUF
          CPI 0
          JM WAIT_FILE_D
          MVI A,80H           ;CREATE FILE
          STA FILE_BUF
WAIT_FILE_C                   ;WAIT FOR FILE CREATE
          LDA FILE_BUF
          CPI 0
          JM WAIT_FILE_C
NEXT_KEY_DATA
          MVI A,-2            ;NOW SETUP KEYBOARD FOR CMD=-2
          STA KEY_BUF+1
          MVI A,240           ;AND MAX # CHARS
          STA KEY_BUF+2
          LDA K_CMD           ;AND OPEN/READ KEYBOARD
          STA KEY_BUF
* WAIT FOR CR(CMD>=0)
WAIT_FOR_CR
          LDA KEY_BUF
          CPI 0
```

Figure 8-23. Simulated Keyboard, Display, and One Disc File I/O - Sample Program

```
       JM WAIT_FOR_CR
       LXI D,DSP_BUF+1
       CALL XFR_DATA
* WRITE TO DISPLAY
       MVI A,82H
       STA DSP_BUF
WAIT_FOR_DSP
       LDA DSP_BUF
       CPI 0
       JM WAIT_FOR_DSP
* WRITE TO FILE FIO
       LXI D,FILE_BUF+1
       CALL XFR_DATA
       LDA KEY_BUF+3
       INR A
       STC
       CMC
       RAR
       STA FILE2BUF+1        ;SET # WORDS
       MVI A,89H
       STA FILE_BUF
WAIT_FILE_W                  ;WAIT FOR FILE WRITE
       LDA FILE_BUF
       CPI 0
       JM WAIT_FILE_W
       JMP NEXT_KEY_DATA
*
* TRANSFER KEY BOARD DATA TO DISPLAY
XFR_DATA
       LXI H,KEY_BUF+3
       MOV B,M               ;GET # BYTES
       MOV A,B
       STAX D
       INX D
       INX H
```

Figure 8-23. Simulated Keyboard, Display, and One Disc File I/O - Sample Program (Cont'd)

```
XFR_LOOP
     MOV A,M
     STAX D
     INX D
     INX H
     DCR B
     JNZ XFR_LOOP
     MVI A,0
     STAX D
     RET
K_CMD DB 80H
* DISPLAY BUFFER
DSP_BUF EQU 100H
* KEYBOARD BUFFER
KEY_BUF EQU 200H
FILE_BUF EQU 300H
     ORG 100H
     DB 0
     ORG 200H
     DB 0
     ORG 300H
     DB 0
     END START
```

Figure 8-23. Simulated Keyboard, Display, and One Disc File I/O - Sample Program (Cont'd)

```
"8080"
     ORG 0
* THIS PROGRAM OPENS KEYBOARD, DISPLAY AND 2 FILES. THEN UPON
* CARRIAGE RETURN IT COPIES KEYBOARD DATA TO DISPLAY AND TO
* FILES F1 AND F2.

* OPEN DISPLAY AND KEYBOARD
START
     LXI SP,400H       ;STACK 03FFH AND BELOW
     MVI A,80H         ;OPEN DISPLAY
     STA DSP_BUF
     STA K_CMD
     MVI A,2
     STA FB1+1         ;TYPE 2(SOURCE)
     MVI A,0           ;DISC#0
     STA FB1+2         ;BOTH FILES ON DISC 0
     STA FB2+2
     MVI A,83H
     STA FB1
WAIT_FILE_D1           ;WAIT FOR FILE DELETE
     LDA FB1
     CPI 0
     JM WAIT_FILE_D1
     MVI A,80H         ;CREATE FILE
     STA FB1
WAIT_FILE_C1           ;WAIT FOR FILE CREATE
     LDA FB1
     CPI 0
     JM WAIT_FILE_C1
     MVI A,2
     STA FB2+1         ;TYPE 2(SCR)
     MVI A,83H
     STA FB2
WAIT_FILE_D2           ;WAIT FOR FILE DELETE
     LDA FB2
     CPI 0
     JM WAIT_FILE_D2
```

Figure 8-24. Simulated Keyboard, Display, and Two Disc Files I/O Sample Program

```
          MVI A,80H         ;CREATE FILE
          STA FB2
WAIT_FILE_C2                ;WAIT FOR FILE CREATE
          LDA FB2
          CPI 0
          JM WAIT_FILE_C2
NEXT_KEY_DATA
          MVI A,-2          ;NOW SETUP KEYBOARD FOR CMD=-2
          STA KEY_BUF+1
          MVI A,240         ;AND MAX # CHARS
          STA KEY_BUF+2
          LDA K_CMD         ;AND OPEN/READ KEYBOARD
          STA KEY_BUF
* WAIT FOR CR(CMD>=0)
WAIT_FOR_CR
          LDA KEY_BUF
          CPI 0
          JM WAIT_FOR_CR
          LXI D,DSP_BUF+1
          CALL XFR_DATA
* WRITE TO DISPLAY
          MVI A,82H
          STA DSP_BUF
WAIT_FOR_DSP
          LDA DSP_BUF
          CPI 0
          JM WAIT_FOR_DSP
* WRITE TO FILE F1
          LXI D,FB1+1
          CALL XFR_DATA
          LDA KEY_BUF+3
          INR A
          STC
          CMC
          RAR
          STA FB1+1         ;SET # WORDS
          MVI A,89H
          STA FB1
```

Figure 8-24. Simulated Keyboard, Display, and Two Disc Files I/O Sample Program (Cont'd)

```
WAIT_FILE_W1               ;WAIT FOR FILE WRITE
       LDA FB1
       CPI 0
       JM WAIT_FILE_W1
* WRITE TO FILE F2
       LXI D,FB2+1
       CALL XFR_DATA
       LDA KEY_BUF+3
       INR A
       STC
       CMC
       RAR
       STA FB2+1           ;SET # WORDS
       MVI A,89H
       STA FB2
WAIT_FILE_W2               ;WAIT FOR FILE WRITE
       LDA FB2
       CPI 0
       JM WAIT_FILE_W2

       JMP NEXT_KEY_DATA
*
* TRANSFER KEY BOARD DATA TO DISPLAY
XFR_DATA
       LXI H,KEY_BUF+3
       MOV B,M             ;GET # BYTES
       MOV A,B
       STAX D
       INX D
       INX H
XFR_LOOP
       MOV A,M
       STAX D
       INX D
       INX H
       DCR B
```

Figure 8-24. Simulated Keyboard, Display, and Two Disc Files I/O Sample Program (Cont'd)

```
                JNZ XFR_LOOP
                MVI A,0
                STAX D
                RET
K_CMD           DB 80H
                ORG 100H          ;CONTROL ADDRESS FOR DISPLAY
DSP_BUF         DB 0

                ORG 200H          ;CONTROL ADDRESS FOR KEYBOARD
KEY_BUF         DB 0

                ORG 300H          ;CONTROL ADDRESS FOR FILE 1
FB1             DB 0

                ORG 400H          ;CONTROL ADDRESS FOR FILE 2
FB2             DB 0
                END START
```

Figure 8-24. Simulated Keyboard, Display, and Two Disc Files I/O Sample Program (Cont'd)

64000 File Formats

The 64000 file accessable to the user through the simulated disc file I/O interface are described in the following paragraphs.

Assembler Symbols File (File Type 12)

This file contains the symbols and their corresponding values assigned by the assembler. It also indicates the symbol type. Symbols may be either ABS (absolute), or relocatable to the PROG, DATA, or COMN areas. (These terms are all defined in the 64000 Assembler/Linker Reference Manual.)

The assembler symbols file is generated each time a source program containing symbols is assembled into an object file. The file consists of a group of records with each record in turn consisting of up to 128 sixteen-bit words (0-127). Each record must be structured as follows: (See figures 8-25 and 26.)

- Record Identification (ID) Word
- Symbol Definition Blocks (Length variable from two to ten words.)
- Checksum Word

Each of the three items is described in the following paragraphs.

Record ID word - The ID word is always the first word in each record and contains the number "6". (The "6" is used internally and is not to be confused with the file type number which is 12.)

Symbol definition blocks - A symbol definition block consists of the symbol word(s) and the value word(s). (See figure 8-27.)

Symbol word(s) - The ASCII character, or characters, are contained in this word (or words). From one to fifteen ASCII characters may be defined. To specify a single-character symbol, only one symbol word is required. To specify either 14 or 15 ASCII characters, the maximum of eight words is required. (Symbols longer than 15 characters are truncated to 15 characters.)

First symbol word - The first word in each symbol definition block is structured the same. The least significant eight bits (7 thru 0) contain the first ASCII character in the symbol. The most significant eight bits (15 thru 8) always contain the following information:

- **Symbol Length (SL)** - Bits 15, 14, and 13 specify the number of symbol words -1 in this block. (See figure 8-28, Example A.) For example, if the symbol consists of two ASCII characters, which require two symbol words, SL is equal to 1. Examples of symbols made up of one to five characters, which require one and three words respectively, are shown in figure 8-28, examples B and C.

- **Reserved Bits** - Bits 12, 11, and 10 contain 000 and are reserved for use by other program modules.

- **Memory Relocation (Relo)** - Bits 9 and 8 specify how the symbol may be relocated as follows:

Bit 9	**Bit 8**	**Storage Type**
0	0	ABS (Absolute)
0	1	PROG area
1	0	DATA area
1	1	COMN area

Additional symbol words - The second through the eighth symbol words may each contain up to two ASCII characters. However, if in the last symbol word, only one byte is required to define the last symbol character, then the least significant byte in that word must contain an ASCII blank (Code 20H). That is, the two bytes in each symbol word must contain meaningful data, even in the last word.

The symbol words must be packed. Only the words actually required to specify the symbols are to be used. Thus, if five symbol words are required to define a symbol, then only five symbol words must be used.

Value word(s) - Immediately following the last symbol word may be either one or two value words, depending upon the size of the target processors addressable memory. This word ,or words, specifies the value assigned to the symbol by the assembler. If the value can be contained in one 16-bit word, then only one word is to be used. Two 16-bit words are used only if they are both required. When two words are used, the first word contains the least significant 16-bits and the second word contains the most significant 16 bits.

All symbol definition blocks within the assembler symbol file must be structured as defined above.

Checksum Word - The checksum word must be the last word in the assembler symbols file. If the file is completely full, then the checksum word will be the 128th word (word #127).

The checksum word contains the arithmetic sum of the binary values of the preceding words in the file.

User Buffer/Assembler Symbols File Packing Formats

The format relationship between the user buffer when reading from, or writing into, a 64000 Assembler Symbols File is shown in figure 8-27.

Linker Symbols File (File Type 13)

The Linker Symbols File is generated any time program modules are linked together. It consists of the following three types of records (see figure 8-29):

- Microprocessor Configuration Record (one per file)
- Globol Symbols Records
- Program Names Records

Each of these items is described below.

Microprocessor Configuration Record (see figure 8-30) - The microprocessor configuration record is the first record in the Linker Symbols File and only one is allowed per file. This record identifies the microprocessor for which the program modules were configured.

This is a fixed length record and consists of 25 words configured as follows:

- One Record Identification (ID) Word
- 15 Pad Words (contain all zeros)
- Eight Words Allocated To:

  Microprocessor Name (9 characters, maximum)

  Microprocessor ID (6 characters, maximum)
- One Checksum Word

Each of these items is described below.

Record identification (ID) word - The record ID word is always the first word in the record. In this case, it is also the first word in the Linker Symbols File and contains the number "1". This number identifies the record as the microprocessor configuration record. (The "1" is used internally and should not be confused with the file type number which is "13".)

Pad words 1 through 15 - These words are inserted so that word positions 16 through 23 in this name record contain the same data as do the corresponding word positions in the name records of the relocatable files.

Name and user ID word block - An eight word block (words 16 through 23) is allocated to contain the name and user ID words. This is the same ID entered into the 64000 in response to the user ID prompt. This block is always eight words long even if all words are not required to define the microprocessor name and user ID. These eight words are structured as follows:

a. **Word 16** - This is the first word and user ID word. The least significant eight bits (7-0) in this word contain the first ASCII character of the microprocessor name. The most significant eight bits (15-8) always contain the following information:

   - Microprocessor Name Length (MNL) - Bits 15, 14, and 13 specify the number of 16-bit words -1 used in a name. The minimum number of characters that may be used in the name is one that requires one word. Thus, the minimum value for MNL is 0. The maximum number of characters that may be used in the name is nine which requires five words. Thus, the maximum value for MNL is four. (See "Words 17 through 23" below.)

- User ID Length (IDL) - Bits 12 and 11 specify the actual number of 16-bit words required for the user ID. (Note that IDL differs from MNL in that IDL specifies the actual number of words and MNL specifies the number of words -1.) The maximum number of characters that may be used in the user ID is six, which requires three words. Thus, the maximum value for IDL is 3.

- Bits 10-8 contain all zeros.

b. **Words 17 through 23** - These words are used for the remaining name and user ID characters. The name characters are specified first, followed by the user ID characters. However, name and ID characters cannot be mixed within the same word. An unused least significant byte in either a name or ID word must contain an ASCII blank (code 20H). The name and ID words must be packed. That is, the ID words must follow the name words with no intervening unused words. Unused words must be at the end of the block.

Checksum word 24 - The checksum word contains the arithmetic sum of the binary values of the preceeding 24 words in this record.

Global Symbols Records (see figure 8-31) - The Linker Symbols File may contain multiple Global Symbols Records. The first Global Symbol Record follows the microprocessor Configuration Record and all succeeding Global Symbol Records are contiguous.

A Global Symbols Record contains the global symbols and the relocated address values (symbol values) generated when the program modules are linked. Each record may consist of up to 128 sixteen-bit words (words 0-127) structured as follows:

- One Record Identification (ID) Word

- Multiple Global Symbol Definition Blocks

- One Checksum Word

Each of these items is described in the following paragraphs.

Record identification (ID) word - The ID word is always the first word in each record and contains the number "2". (The "2" is used internally and is not to be confused with the file type number which is 13.)

Global symbol definition blocks - A global symbol definition block consists of the symbol word(s) and the value word(s). (See figure 8-31.)

Symbol word(s) - The ASCII character, or characters, are contained in this word (or words). From one to fifteen ASCII characters may be defined. To specify a single-character symbol, only one symbol word is required. To specify either 14 or 15 ASCII characters, the maximum of eight words is required. (Symbols longer than 15 characters are truncated to 15 characters.

First symbol word - The first word in every symbol definition block is structured the same. The least significant eight bits (7 thru 0) contain the first ASCII character in the symbol. The most significant eight bits (15 thru 8) always contain the following information. (See figure 8-32.)

- **Global Symbol Length (GSL)** - Bits 15, 14, and 13 specify the number of symbol words −1 in this block. For example, if the global symbol consists of two ASCII characters, which require two symbol words, GSL is equal to 1. (The second byte in the second word will contain an ASCII blank, i.e. code 20H.)

- **Reserved Bits** - Bits 12, 11, and 10 contain 000 and are reserved for use by other program modules.

- **Memory Relocation (Relo)** - Bits 9 and 8 specify how the symbol may be relocated as follows:

Bit 9	**Bit 8**	**Storage Type**
0	0	ABS (Absolute)
0	1	PROG area
1	0	DATA area
1	1	COMN area

Additional symbol words - The second through the eighth symbol words may each contain up to two ASCII characters. However, if in the last symbol word, only one byte is required to define the last symbol character, then the least significant byte in that word must contain an ASCII blank (code 20H). That is the two bytes in each symbol word must contain meaningful data, even in the last word.

The symbol words must be packed. Only the words actually required to specify the symbols are to be used. Thus, if five symbol words are required to define a symbol, then only five symbol words must be used.

Symbol value words(s) - Immediately following the last symbol word may be either one or two value words depending upon the size of the target processors addressable memory. This word (or words) specifies the value assigned to the symbol by either the assembler (if ABS - absolute) or by the linker. If the value can be contained in one 16-bit word, then only one word is to be used. Two 16-bit words are used only if they are both required. When two words are used, the first word contains the least significant 16-bits and the second word contains the most significant bits.

All global symbol definition blocks within the Linker Symbols File must be structured as defined above.

Checksum word - The checksum word must be the last word in each record. If the record is completely full, then the checksum word will be the 128th word (word #127).

The checksum word contains the arithmetic sum of the binary values of the preceding words in the record.

Program Names Records (see figure 8-33) - The Linker Symbol File may contain multiple Program Names Records. The first Program Names Record follows the last Global Symbols Record. All succeeding Program Names Records are contiguous.

A Program Names Record contains the names of the programs, the corresponding user ID's and the load addresses generated when the program modules are linked. Each record may consist of up to 128 sixteen-bit words (words 0-127) structured as follows:

- One Record Identification (ID) Word
- Multiple Program Name and Addresses Definition Blocks (Fixed length blocks of 14 words each)
- One checksum word

Each of these items are described in the following paragraphs.

Record identification (ID) word - The ID word is always the first word in each record and contains the number "3". (The "3" is used internally and is not to be confused with the file type number which is 13.)

Program name and addresses definition block - This is a fixed length block consisting of 14 sixteen-bit words allocated as follows:

- Eight words reserved for the program name and users ID
- Six words reserved for the linker load addresses (see figure 8-34)

Program name and user ID words - The formatting and packing of these words are done in the same way as described above for the "Microprocessor Configuration Record, Name and ID Word Block".

Load address words - These words contain the load addresses assigned by the linker. If an address is not assigned to a particular area, the address words contain zeros (0000H). The MS 16-bit address word will be used only if required by the target microprocessors addressable memory space.

Checksum word - The checksum word must be the last word in each record. If the record is completely full, then the checksum word will be the 128th word (word #127).

The checksum word contains the arithmetic sum of the binary values of the preceding words in the record.

User Buffer/Linker Symbols File Packing Formats

The format relationship between the user buffer when reading or writing into a 64000 Linker Symbols File is the same as shown for the Assembler Symbols File in figure 8-27.

Source File (File Type 2)

The source file is generated by the programmer from the applicable microprocessor opcodes and assembler pseudo instructions. It consists of a series of ASCII records. (See figures 8-35 and 8-36.)

Each ASCII source record in the file is structured the same. An ASCII source record is of variable length and may contain up to 128 sixteen-bit words. Each 16-bit word contains two 8-bit ASCII bytes. If the last byte in the last word of a record is not used, it must contain an ASCII blank (20H).

The format relationship between the user buffer when reading from or writing into a 64000 source file is also shown in figure 8-36.

Listing File (File Type 5)

The listing file is a copy of a source file. It may be produced when listing to a printer, a display, etc. The format is identical to that described above, and shown in figures 8-35 and 8-36 for the source file.

Absolute File (File Type 4)

Absolute file is generated when the linker produces an absolute image of an object file or files. The absolute file contains two types of records; the first record and the additional records which follow the first record. (See figures 8-37 and 8-38.)

First record - The first record has a fixed length of four 16-bit words. The first word (word 0) specifies the processors data bus width (8, 16, etc.). The second word (word 1) specifies the data width base of the target microprocessor. The data width base is the minimum addressable entity (i.e. group of bits) used by the microprocessor. Normally this will be 8-bits, but not always.

The last two words specify the transfer address value loaded into the target microprocessors program counter. The most significant transfer address word (bits 31 thru 16) is used only if required. If not used it will contain 0000H.

Additional records - All records following record one are formatted the same. Each is a variable length record consisting of up to 128 sixteen-bit words (0-127).

The first word in the record (word 0) specifies the number of data bytes in the record (2 bytes/word). The following two words (words 1 and 2) specify the load address for this record. (The load address is the beginning location for storing this record.) The most significant load address word (bits 31 thru 16) will be used only if required. If not used, bits 31 thru 16 will contain 0000H.

The remaining words in the record (3 thru n) contain the data bytes. If the last byte in the last word of a record is not used for data, it must contain an ASCII blank (code 20H).

The format relationship between the user buffer when reading from or writing into a 64000 absolute file is also shown in figure 8-38.

PROM Absolute File (File Type 9)

The PROM absolute format is similar to that described above and shown in figures 8-37 and 8-38 for the absolute file. The only difference is that in PROM absolute files, the transfer address words in the first record always contain zeros.

Relocatable File (File Type 3)

The relocatable file is produced by the assembler or compiler. It contains information required by the linker to construct an absolute file. This file consists of the following six types of records (see figure 8-39):

- Program Description Record (one per file)
- Global Symbols Record
- Data Record
- External Symbols Record

- Local Symbols Record (optional)
- End Record (one per file)

Each type of record is defined in the following paragraphs.

Program Description Record (see figure 8-40) - The program description record is the first record in the Relocatable File and only one is allowed per file. This record identifies the source program, number of externals, microprocessor, comments, and absolute code definitions.

This is a variable length record (up to 128 words) and is configured as follows:

- One Record Identification (ID) Word
- 14 words allocated to:

  Source Program Name (9 characters, maximum)

  Source Program ID (6 characters, maximum)

  PROG Area Length (2 words, maximum)

  DATA Area Length (2 words, maximum)

  COMN Area Length (2 words, maximum)

- One word allocated to definition of the number of external variables and procedures defined in the module.
- Eight words allocated to:

  Microprocessor Name (9 characters, maximum)

  Microprocessor ID (6 characters, maximum)

- Two words allocated to:

  Date (one word, maximum)

  Time (one word, maximum)

- 11 words allocated to comments

- Up to 88 words allocated to absolute code segment description.

- One checksum word

Each of these items is described as follows:

Record identification (ID) word - The record ID word is always the first word in the record. In this case, it is also the first word in the Relocatable File and contains the number "1". This number identifies the record as the source program description record. (The "1" is used internally and should not be confused with the file type number which is "3".)

Source program name and user ID word block - An eight word block (words 1 thru 8) is allocated to contain the source program name and user ID words. This is the same ID entered into the 64000 in response to the user ID prompt. This block is always eight words long even if all words are not required to define the source program name and user ID. These eight words are constructed as follows:

a. **Word 1** - This is the first word and user ID word. The least significant eight bits (7-0) in this word contain the first ASCII character of the source program name. The most significant eight bits (15-8) always contain the following information:

  - Source Program Name Length (PNL) - Bits 15, 14, and 13 specify the number of 16-bit words −1 used for the name. The minimum number of characters that may be used in the name is one, which requires one word. Thus, the minimum value for PNL is zero. The maximum number of characters that may be used in the name is nine, which requires five words. Thus, the maximum value for PNL is four. (See "Words 2 through 8", below.)

  - User ID Length (IDL) - Bits 12 and 11 specify the actual number of 16-bit words required for the user ID. (Note that IDL differs from PNL in that IDL specifies the actual number of words and PNL specifies the number of words −1.) The maximum number of characters that may be used in the user ID is six, which requires three words. Thus, the maximum value for IDL is 3.

  - Bits 10-8 contain the number of the disc which holds the record.

b. **Words 2 through 8** - These words are used for the remaining name and user ID characters. The name characters are specified first, followed by the user ID characters. However, name and ID characters can not be mixed within the same word. An unused least significant byte in either a name or ID word must contain an ASCII blank (code 20H). The name and ID words must be packed. That is - the ID words must follow the name words with no intervening unused words. Unused words must be at the end of the block.

Length word block - A six word block (words 9 thru 14) is allocated to contain the word lengths of code produced by the assembler or compiler in each of the three relocatable sections; PROG, DATA, and COMN.

Number of externals word - One word (word 15) is allocated to contain the number of external variables and procedures defined in the module. This number can be from 0 to 511.

Microprocessor name and user ID word block - This word block is the same as described for the Linker Symbols File under the "Microprocessor Configuration Record, Name and User ID Word Block".

Date and time word block - Two words (words 24 and 25) are allocated to contain the date and time that the program was assembled or compiled.

Comments word block - A block of eleven words (words 26 thru 36) is allocated for comments. The block contains up to 22 ASCII characters defined by the NAME psuedo in the assembler or compiler. All unused characters must contain ASCII blanks (code 20H).

Absolute code segment word block - A variable length block which contains from 0 to 22 entries of four 16-bit words is allocated for absolute code segments. Each four-word entry defines an absolute code segment declared in the assembler or compiler.

Checksum word - The checksum word must be the last word in each record. If the record is completely full, then the checksum word will be the 128th word. (Word #127.)

The checksum word contains the arithmetic sum of the binary values of the preceding words in the record.

Global Symbols Records (see figures 8-31 and 8-32) - The global symbols record formatting and packing for the Relocatable File is the same as described for the Linker Symbols File under the "Global Symbols Records".

Data Records (see figure 8-41) - The data records contains the relocation area and address of the program as assigned by the linker. It also defines how the absolute codes are produced.

Record identification (ID) word - The ID word is always the first word in each record and contains the number "3". (The "3" is used internally and is not to be confused with the file type number, which is also "3".

Relocation address words - These words contain the relocation address assigned by the linker to this program. The most-significant word is used only when the ID offset equals 3.

Relocation word - The relocation word identifies the relocation destination code as follows: 00=ABS, 01=PROG, 10=DATA, and 11=COMN.

Event selection word - This word contains codes 00, 01, 10, and 11 in bit locations T1 thru T8. Any one of the codes may be contained in any of the locations. As T1 through T8 are read, the event selected by the specific code will be executed. Codes are defined as follows:

Tn=00 -	Produce one byte of absolute code, which is found in the low order byte of the corresponding word.
Tn=01 -	Produce two bytes of absolute code, which is found in the corresponding word.
Tn=10 -	Relocate the address to be found in the second (and optionally, the third) word based on the relocation code in the first word. Then produce an absolute code based on the processor dependent format number in the first word and skeleton, if used.
Tn=11 -	Look up the external symbol whose number is in the first word (which has been previously defined in a type 4 record). Add the displacement and then produce an absolute code based on the format number and skeleton, if used.

Checksum word - The checksum word must be the last word in each record. If the record is completely full, then the checksum will be the 128th word (word #127).

The checksum word contains the arithmetic sum of the binary values of the preceding words in the record.

External Symbols Records (see figure 8-42) - The Relocatable File may contain multiple External Symbols Records.

An External Symbols Record contains the external symbols and the external ID number assigned by the assembler or compiler. Each record may consist of up to 128, sixteen-bit words (words 0-127) structured as follows:

- One Record Identification (ID) Word

- Multiple External Symbol Definition Blocks
- One Checksum Word

Each of these items is described as follows:

Record identification (ID) word - The ID word is always the first word in each record and contains the number "4". (The "4" is used internally and is not to be confused with the file number, which is "3".)

External symbol definition blocks - An external symbol definition block consists of the symbol word(s) and the external ID number. (See figure 8-42.)

Symbol words - The ASCII character, or characters, are contained in this word, or words. From one to fifteen ASCII characters may be defined. To specify a single-character symbol, only one symbol word is required. To specify either 14 or 15 ASCII characters, the maximum of eight words is required. (Symbols longer than 15 characters are truncated to 15 characters.)

First symbol word - The first word in every symbol definition block is structured the same. The least significant 8 bits (7-0) contain the first ASCII character in the symbol. The most significant eight bits (15-8) always contain the following information:

- External Symbol Length (ESL) - Bits 15, 14, and 13 specify the number of symbol words −1 in this block. For example, if the external symbol consists of two ASCII characters, which requires two symbol words, then ESL is equal to 1. (The second byte in the second word will contain an ASCII blank - i.e. code 20H.)

- Reserved Bits - Bits 12, 11, 10, 9, and 8 always contain 00100.

Additional symbol words - The second through the eighth symbol words may each contain up to two ASCII characters. However, if in the last symbol word, only one byte is required to define the last symbol character, then the least significant byte in that word must contain an ASCII blank (code 20H). That is, the two bytes in each symbol word must contain meaningful data, even in the last word.

The symbol words must be packed. Only the words actually required to specify the symbols are to be used. Thus, if five symbol words are required to define a symbol, then only five words are to be used.

External ID number word - The external ID number is assigned by the assembler or compiler. The number can be from 0 to 511.

Checksum word - The checksum word must be the last word in each record. If the record is completely full, then the checksum will be the 128th word (word #127).

The checksum word contains the arithmetic sum of the binary values of the preceding words in the record.

Local Symbols Records (see figures 8-31 and 8-32) - The local symbols records formatting and packing for the Relocatable File is the same as described for the Linker Symbols File under the "Global Symbols Records", except the ID word contains the number "6".

End Record (see figure 8-43) - The end record is the last record in the Relocatable File and only one is allowed per file. The end record contains the relocation code and transfer address. Each record consists of five, 16-bit words structured as follows:

- One Record Identification (ID) Word
- One Relocation Word
- Two Transfer Address Words
- One Checksum Word

Each of these items are described as follows:

Record identification (ID) word - The ID word is always the first word in each record and contains the number "5". (The "5" is used internally and is not to be confused with the file number, which is "3".)

Relocation word - The relocation word identifies the relocation destination code, as follows: 00=ABS, 01=PROG, 10=DATA, and 11=COMN.

Transfer address words - The transfer address words contain the address where control will be transferred to when the program is run.

Checksum word - The checksum word must be the last word in each record. The checksum word contains the arithmetic sum of the binary values of the preceding words in the record.

User Buffer/Relocatable File Packing Formats

The format relationship between the user buffer when reading from, or writing into, a 64000 Relocatable File is the same as shown for the Assembler Symbols File in figure 8-27.

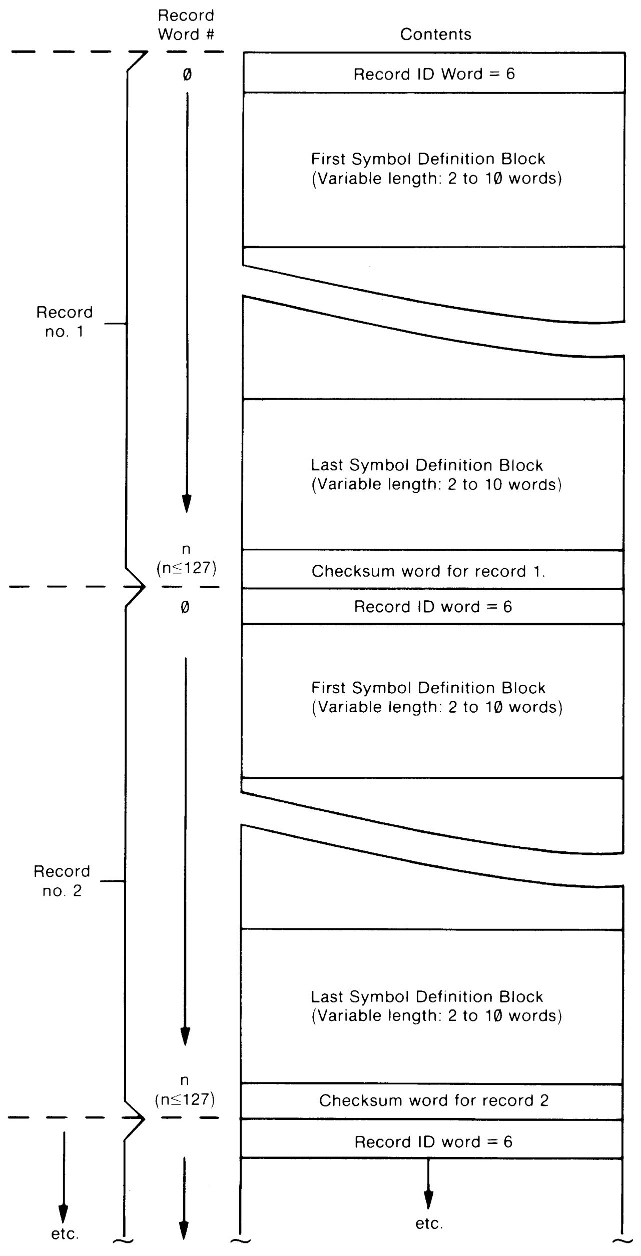
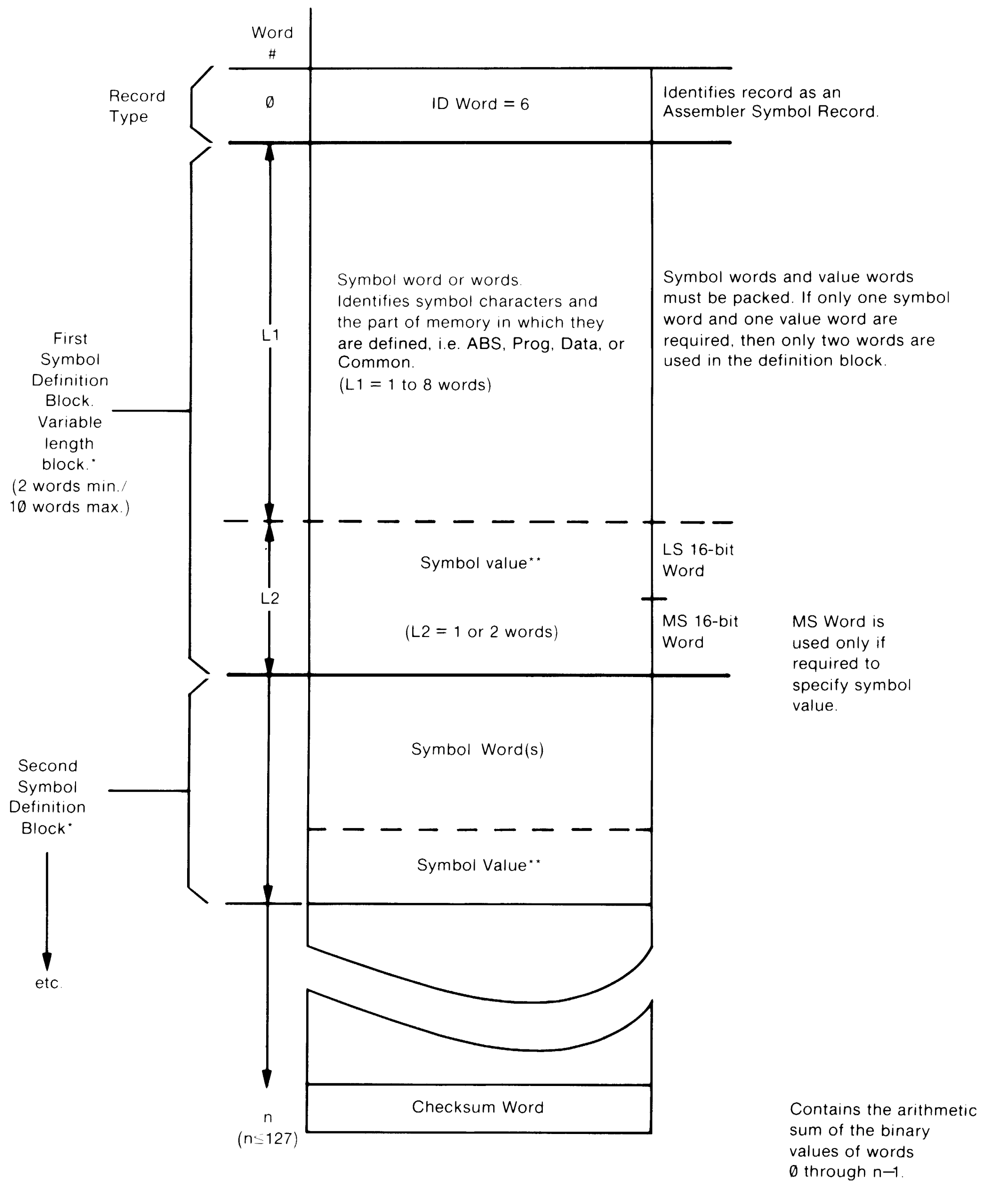

Record Word #
Contents
Ø
Record ID Word = 6
First Symbol Definition Block (Variable length: 2 to 1Ø words)
Record no. 1
Last Symbol Definition Block (Variable length: 2 to 10 words)
n (n≤127)
Checksum word for record 1.
Ø
Record ID word = 6
First Symbol Definition Block (Variable length: 2 to 1Ø words)
Record no. 2
Last Symbol Definition Block (Variable length: 2 to 1Ø words)
n (n≤127)
Checksum word for record 2
Record ID word = 6
etc.
etc.

Figure 8-25. Assembler Symbol File Overall Structure

ASSEMBLER SYMBOL RECORD STRUCTURE


Word #
Record Type
0
ID Word = 6
Identifies record as an Assembler Symbol Record.
First Symbol Definition Block. Variable length block.* (2 words min./ 10 words max.)
L1
Symbol word or words. Identifies symbol characters and the part of memory in which they are defined, i.e. ABS, Prog, Data, or Common. (L1 = 1 to 8 words)
Symbol words and value words must be packed. If only one symbol word and one value word are required, then only two words are used in the definition block.
L2
Symbol value**
(L2 = 1 or 2 words)
LS 16-bit Word
MS 16-bit Word
MS Word is used only if required to specify symbol value.
Second Symbol Definition Block*
Symbol Word(s)
Symbol Value**
etc.
n (n≤127)
Checksum Word
Contains the arithmetic sum of the binary values of words 0 through n–1.

Notes

*For block structure details, see "Assembler-Symbol Record/User Buffer Format Details".

**Symbol value as assigned by assembler. If a relocatable value it will be relocated by the linker.

Figure 8-26. Assembler Symbol Record Structure

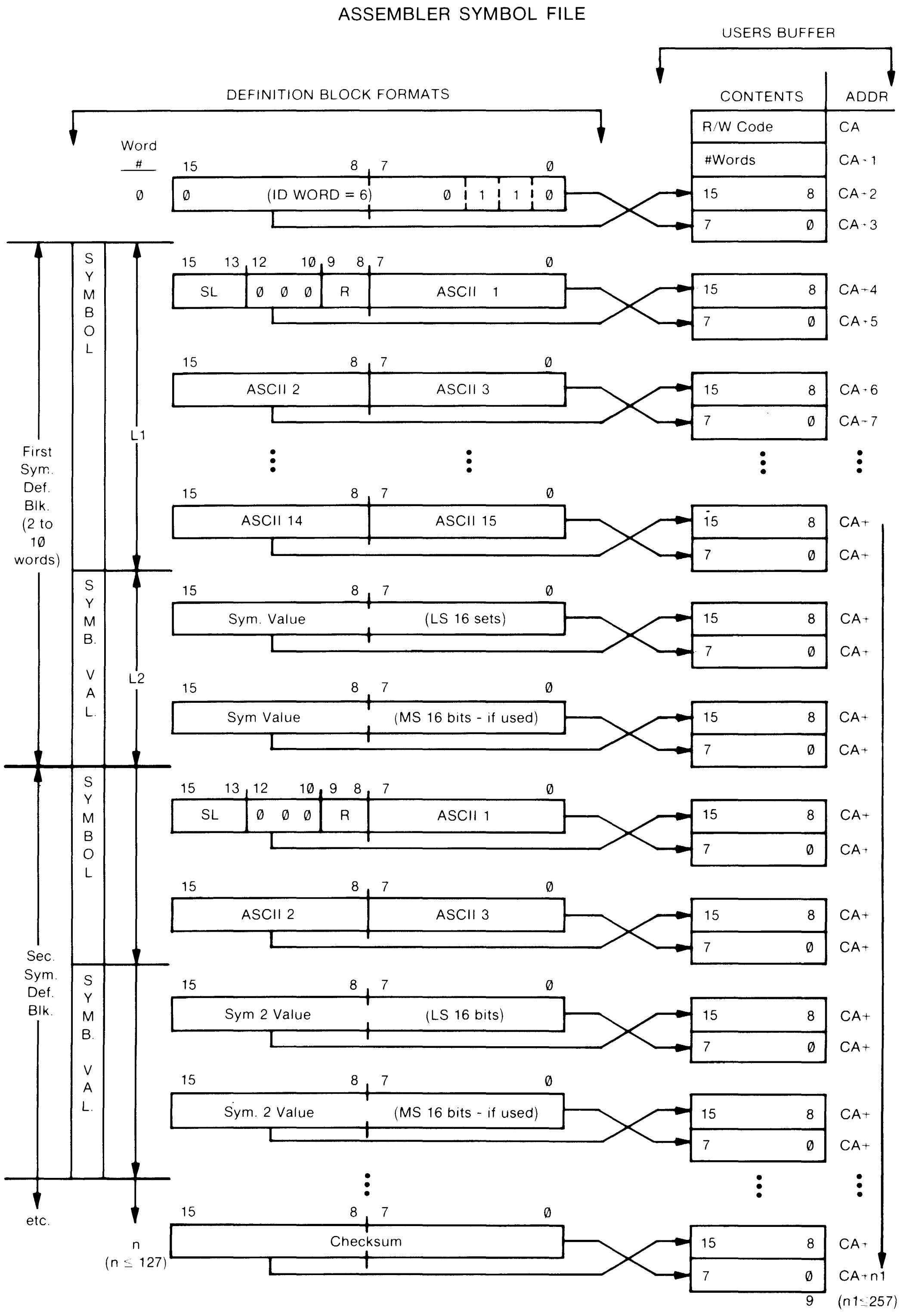
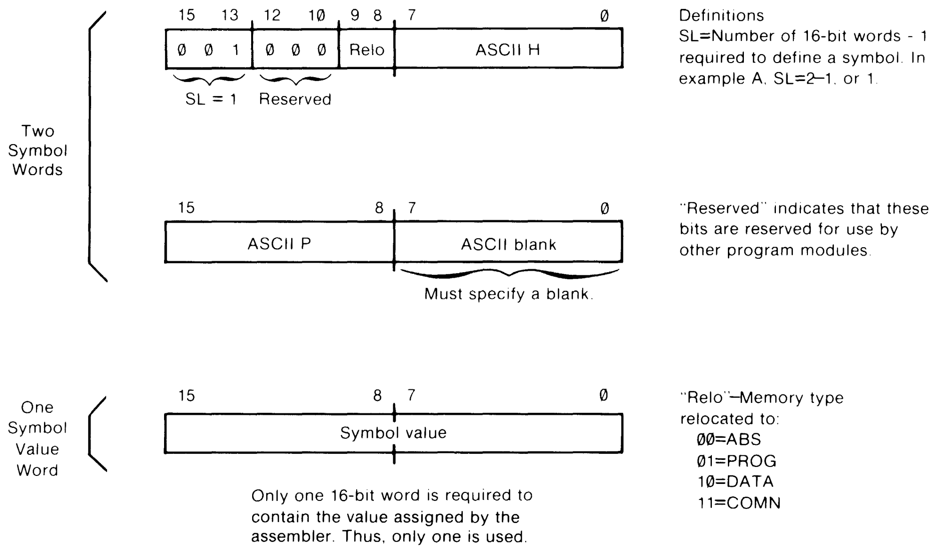


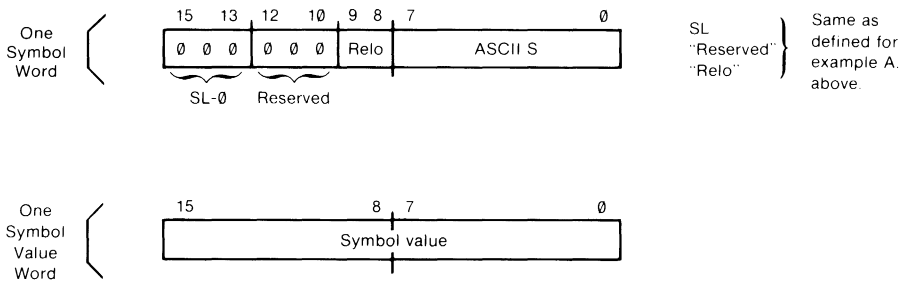
Figure 8-27. Assembler Symbol Record/User Buffer Format Details

EXAMPLE A. SYMBOL = HP



Two Symbol Words
15 13 12 10 9 8 7 0
Ø Ø 1
Ø Ø Ø
Relo
ASCII H
SL = 1
Reserved
15 8 7 0
ASCII P
ASCII blank
Must specify a blank.
One Symbol Value Word
15 8 7 0
Symbol value
Only one 16-bit word is required to contain the value assigned by the assembler. Thus, only one is used.
Definitions
SL=Number of 16-bit words - 1 required to define a symbol. In example A, SL=2–1, or 1.
"Reserved" indicates that these bits are reserved for use by other program modules.
"Relo"–Memory type relocated to:
ØØ=ABS
Ø1=PROG
1Ø=DATA
11=COMN

EXAMPLE B. SYMBOL = S



Again, only one 16-bit word is required to contain the symbol value. Thus, only one is used.

Figure 8-28. Assembler Symbol Record/Symbol Definition Block Examples

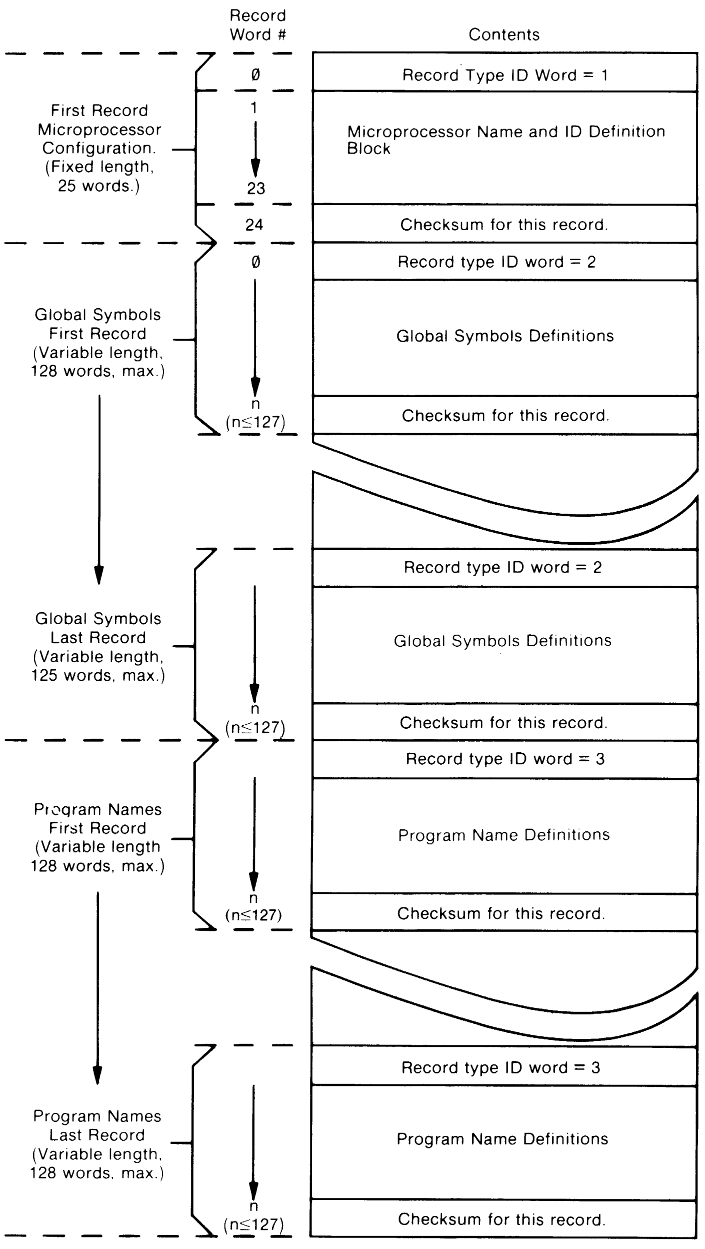
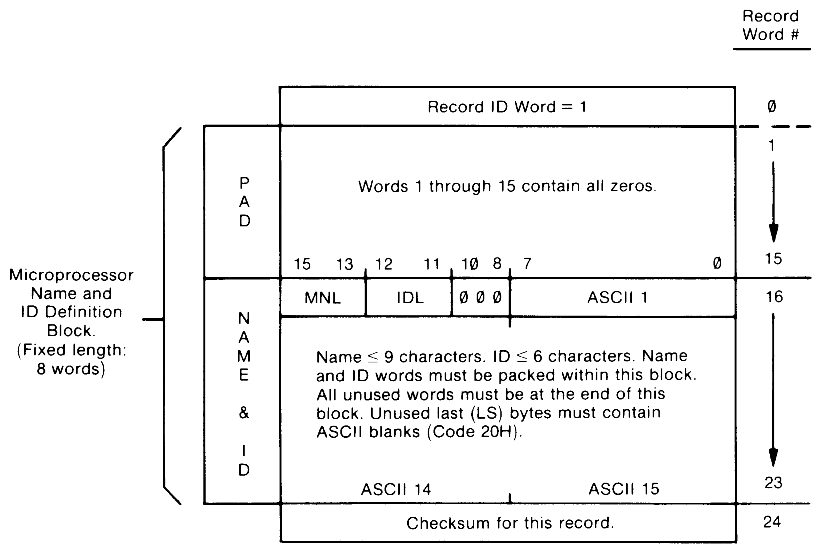

Record Word #
Contents
Ø
Record Type ID Word = 1
1
Microprocessor Name and ID Definition Block
23
24
Checksum for this record.
First Record Microprocessor Configuration. (Fixed length, 25 words.)
Ø
Record type ID word = 2
Global Symbols Definitions
n (n≤127)
Checksum for this record.
Global Symbols First Record (Variable length, 128 words, max.)
Record type ID word = 2
Global Symbols Definitions
n (n≤127)
Checksum for this record.
Global Symbols Last Record (Variable length, 125 words, max.)
Record type ID word = 3
Program Name Definitions
n (n≤127)
Checksum for this record.
Program Names First Record (Variable length 128 words, max.)
Record type ID word = 3
Program Name Definitions
n (n≤127)
Checksum for this record.
Program Names Last Record (Variable length, 128 words, max.)

Figure 8-29. Linker Symbol File Overall Structure

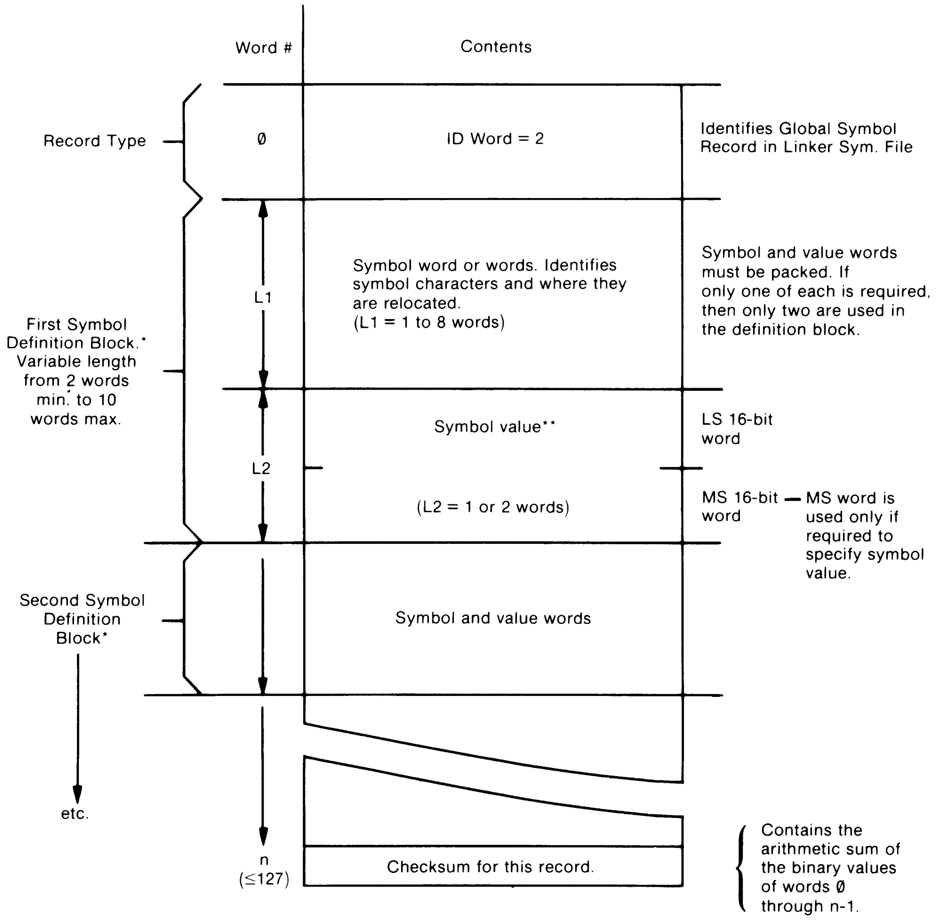

Record Word #
Record ID Word = 1
Ø
1
P A D
Words 1 through 15 contain all zeros.
15 13 12 11 1Ø 8 7 Ø
15
Microprocessor Name and ID Definition Block. (Fixed length: 8 words)
N A M E & I D
MNL
IDL
Ø Ø Ø
ASCII 1
16
Name ≤ 9 characters. ID ≤ 6 characters. Name and ID words must be packed within this block. All unused words must be at the end of this block. Unused last (LS) bytes must contain ASCII blanks (Code 20H).
ASCII 14
ASCII 15
23
Checksum for this record.
24

Notes

1. Words 1 through 15 are added so that word positions 16-23 in this name record contain the same data as do the corresponding word positions in the name records of the relocatable files.
2. MNL = Number of 16-bit words−1 required to define the microprocessor name. At least one character in the "ASCII 1" byte is required. Thus, with a one character name, MNL = 0. If all nine characters are used (5 words), MNL = 4.
3. IDL = Actual number of 16-bit words required to define the user ID. If one word is used, IDL = 1. If all three words are used, IDL = 3.
4. Bits 1Ø, 9, and 8 always contain ØØØ.
5. ASCII bytes 1-15 contain the name and ID characters. These words must be packed. That is the ID words must follow the name words. Unused words must be at the end of the block. An unused byte in either a name or ID word must contain an ASCII blank (Code 20H).
6. The checksum contains the arithmetic sum of the binary values of words Ø through 23.

Figure 8-30. Microprocessor Configuration Record Structure

GLOBAL SYMBOLS RECORD

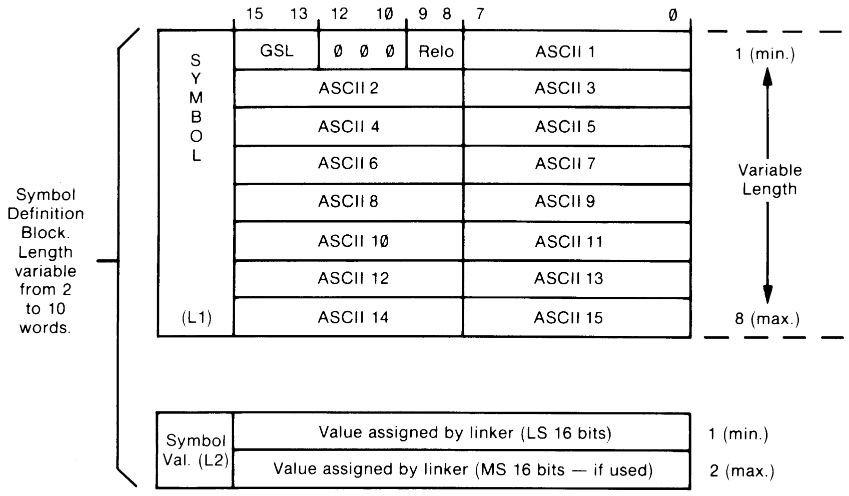

Word #
Contents
Record Type
0
ID Word = 2
Identifies Global Symbol Record in Linker Sym. File
First Symbol Definition Block.* Variable length from 2 words min. to 10 words max.
L1
Symbol word or words. Identifies symbol characters and where they are relocated. (L1 = 1 to 8 words)
Symbol and value words must be packed. If only one of each is required, then only two are used in the definition block.
L2
Symbol value**
(L2 = 1 or 2 words)
LS 16-bit word
MS 16-bit word
MS word is used only if required to specify symbol value.
Second Symbol Definition Block*
Symbol and value words
etc.
n (≤127)
Checksum for this record.
Contains the arithmetic sum of the binary values of words 0 through n-1.

Notes

*For block structure details see "Global Symbols Definition Block Diagram."

**Symbol value assigned by assembler. If relocatable value (not ABS), it will be relocated by the linker.

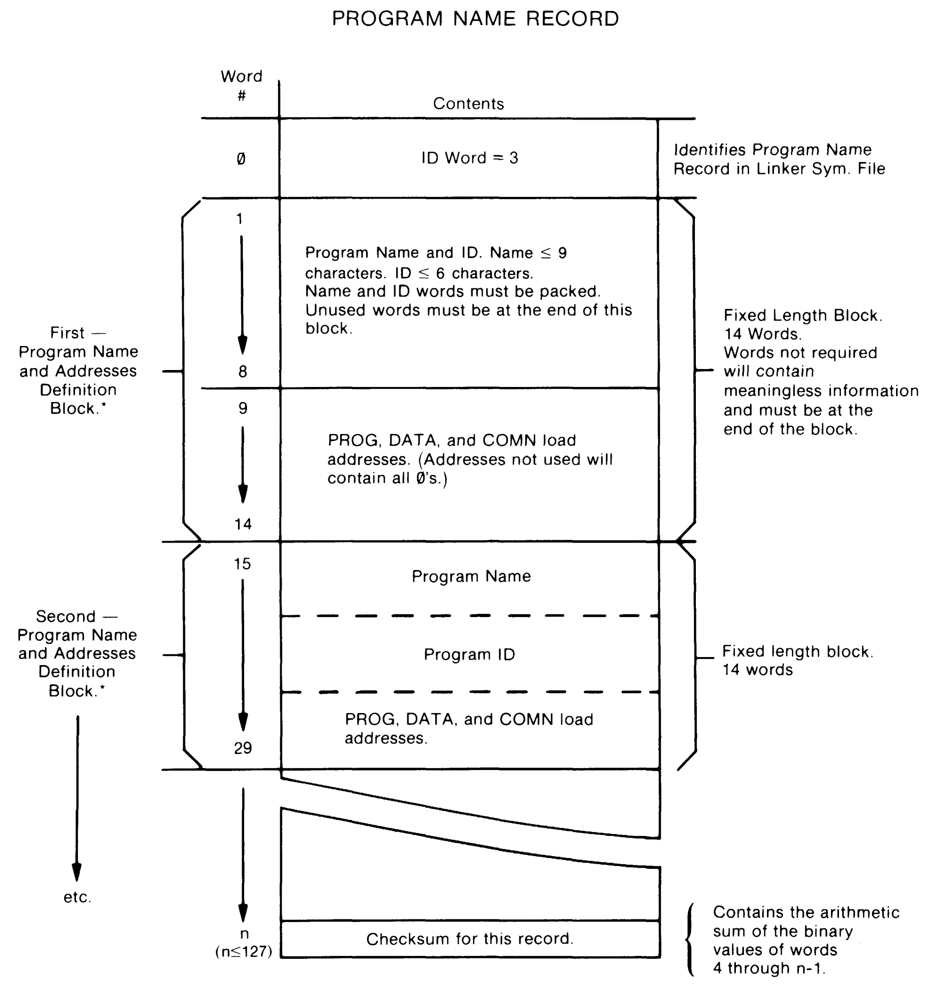
Figure 8-31. Global Symbol Record Structure



Notes

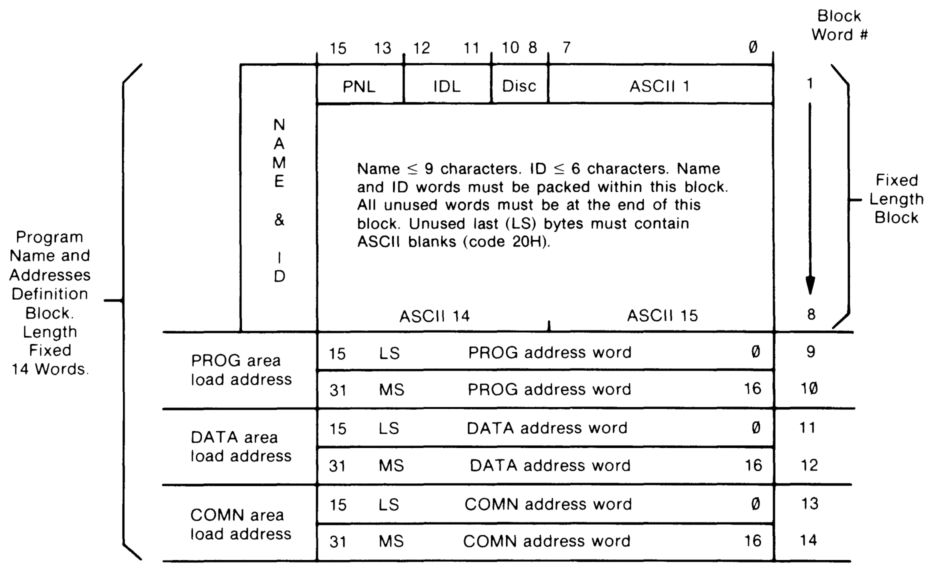
1. GSL = Number of 16-bit words −1 required to define a global symbol. At least one character is required in the "ASCII 1" byte. Thus, with a one character name, NL = 0. If all 15 characters are used (8 words), NL = 7.
2. Bits 12, 11, 10 are reserved for use by other program modules and always contain 000.
3. "Relo" contains the binary code for area relocated to as follows: 00 = ABS, 01 = PROG, 10 = DATA, and 11 = COMN.
4. The bytes labeled ASCII 1-15 are the maximum number of bytes available to define the symbol. Only the actual number of 16-bit words required to define the symbol will exist. However, if the first byte (MSB) is used, then the second byte (LSB) must contain an ASCII blank (code 20 H).
5. The symbol value is assigned by the assembler. If a relocatable value it will be relocated by the linker.

Figure 8-32. Global Symbol Definition Block


PROGRAM NAME RECORD
Word #
Contents
0
ID Word = 3
Identifies Program Name Record in Linker Sym. File
1
8
Program Name and ID. Name ≤ 9 characters. ID ≤ 6 characters. Name and ID words must be packed. Unused words must be at the end of this block.
9
14
PROG, DATA, and COMN load addresses. (Addresses not used will contain all 0's.)
First — Program Name and Addresses Definition Block.*
Fixed Length Block. 14 Words. Words not required will contain meaningless information and must be at the end of the block.
15
29
Program Name
Program ID
PROG, DATA, and COMN load addresses.
Second — Program Name and Addresses Definition Block.*
Fixed length block. 14 words
etc.
n (n≤127)
Checksum for this record.
Contains the arithmetic sum of the binary values of words 4 through n-1.

Notes
*For block structure details, see figure 8-34.

Figure 8-33. Program Name Record Structure


Block Word #
15 13 12 11 10 8 7 Ø
PNL
IDL
Disc
ASCII 1
1
N A M E & I D
Name ≤ 9 characters. ID ≤ 6 characters. Name and ID words must be packed within this block. All unused words must be at the end of this block. Unused last (LS) bytes must contain ASCII blanks (code 20H).
Fixed Length Block
Program Name and Addresses Definition Block. Length Fixed 14 Words.
ASCII 14
ASCII 15
8
PROG area load address
15 LS PROG address word Ø
9
31 MS PROG address word 16
1Ø
DATA area load address
15 LS DATA address word Ø
11
31 MS DATA address word 16
12
COMN area load address
15 LS COMN address word Ø
13
31 MS COMN address word 16
14

Notes

1. PNL = Number of 16-bit words—1 required to define the program name. At least one character in the "ASCII 1" byte is required. Thus, with a one character name, PNL = Ø. If all nine characters are used (5 words), PNL = 4.

2. IDL = Actual number of 16-bit words required to define the user ID. If one word is used, IDL = 1. If all three words are used, IDL = 3.

3. DISC = The identifying number of the disc upon which the program resides.

4. ASCII bytes 1-15 contain the name and ID characters. These words must be packed. That is - the ID words must follow the name words. Unused words must be at the end of the block. An unused byte in either a name or ID word must contain an ASCII blank (Code 20H).

5. Load Address Words - The load address words contain the load address assigned by the linker to this program. Unused address words contain all zeros.

Figure 8-34. Program Name and Address Definition Block Format

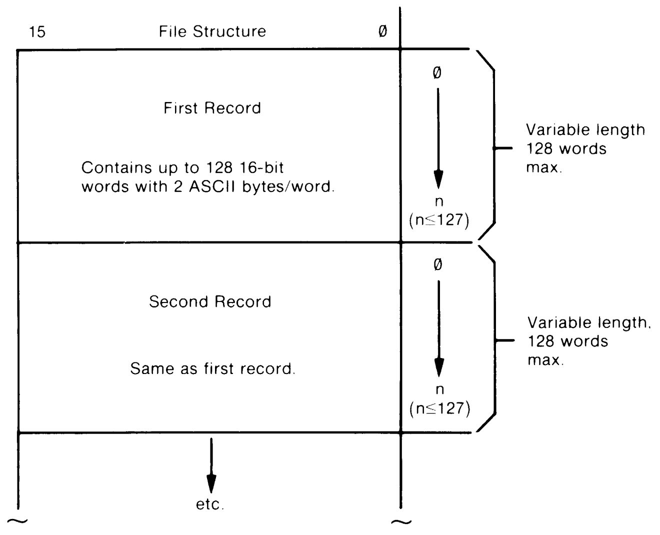

15
File Structure
0
First Record
Contains up to 128 16-bit words with 2 ASCII bytes/word.
0
n
(n≤127)
Variable length 128 words max.
Second Record
Same as first record.
0
n
(n≤127)
Variable length. 128 words max.
etc.

Figure 8-35. Source and Listing Files - Overall Structure

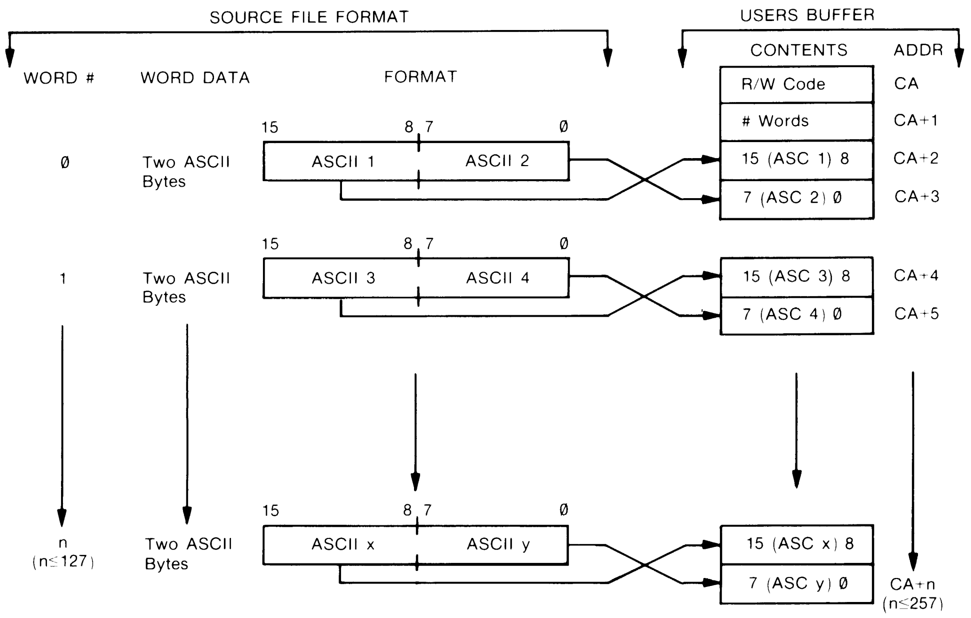


Figure 8-36. Source and Listing File Format

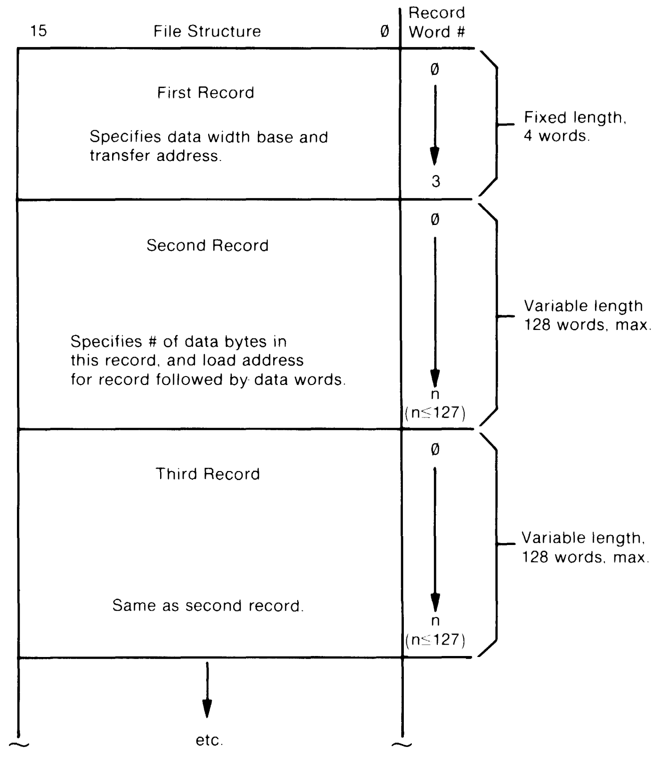
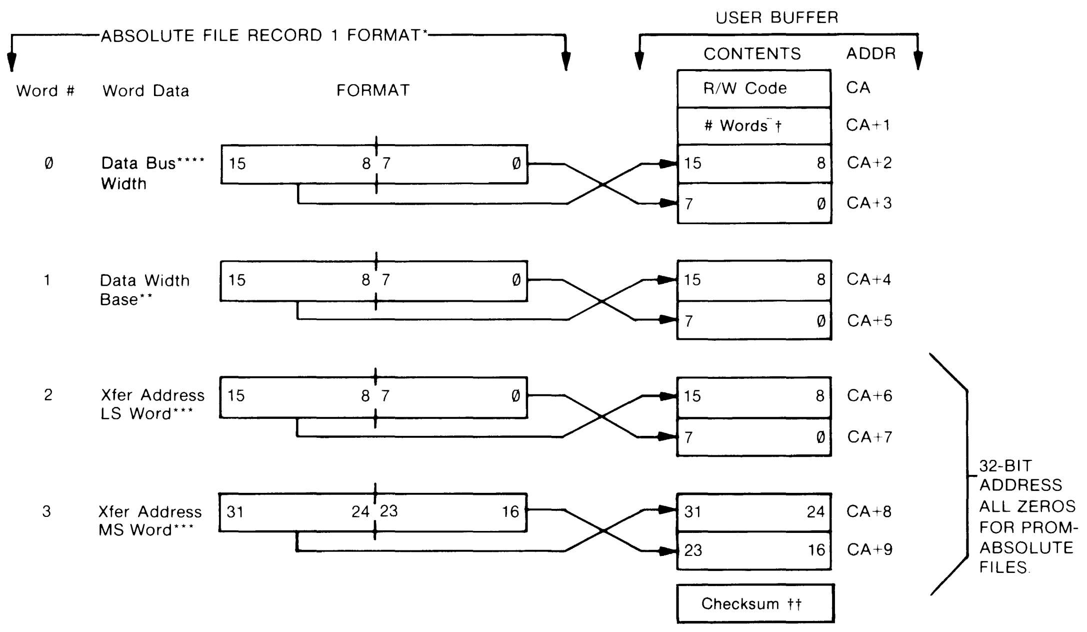

15
File Structure
Ø
Record Word #
First Record
Specifies data width base and transfer address.
Ø
3
Fixed length, 4 words.
Second Record
Specifies # of data bytes in this record, and load address for record followed by data words.
Ø
n
(n≤127)
Variable length 128 words, max.
Third Record
Same as second record.
Ø
n
(n≤127)
Variable length, 128 words, max.
etc.

Figure 8-37. Absolute and PROM Absolute File - Overall Structure

ILLUSTRATION A.
RECORD 1 FORMAT ONLY.
(Format for all Other Records Shown on Illustration B)



Notes

*Record 1 must precede all other records in an absolute file and it must always be formatted as shown. (Always four words.)

**The Data Width Base is the minimum addressable entity (i.e., group of bits) used by the microprocessor. Normally this will be 8 bits but not always.

***The transfer address is the value loaded into the microprocessor program counter. This value is all zeros for PROM Absolute files.

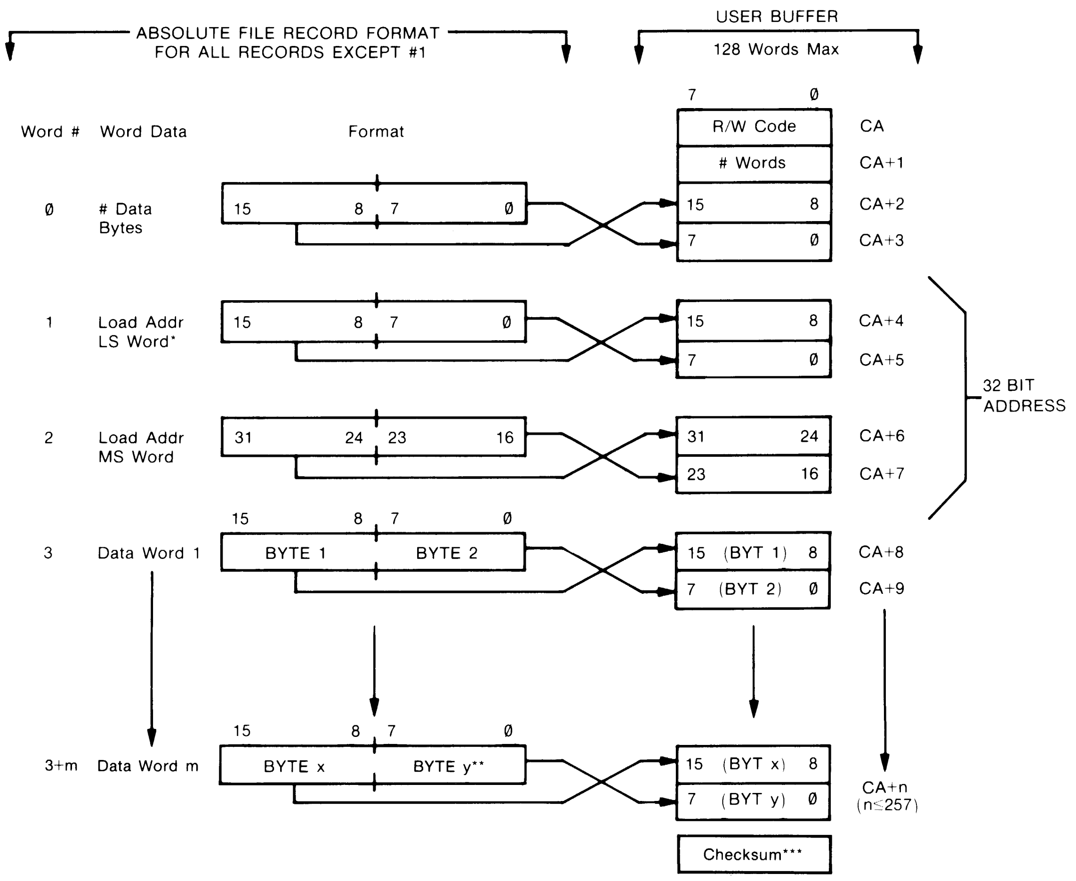
****Width of processor data bus (i.e., 8, 16 etc.)

† Total number of words in record excluding checksum and number of words, (i.e. n-2), always equal to 4 for record 1.

†† The checksum is the module 256 sum of bytes CA+2 through CA+9.

Figure 8-38. Absolute and PROM Absolute File Formats

ILLUSTRATION B.
FORMAT FOR ALL RECORDS EXCEPT RECORD 1
(See Illustration A for Record 1 Format)



Note

*The load address is the address of the first location into which this record is stored.

**This last byte will be a pad byte if the record contains an odd number of bytes. This is required to fill up the word boundary.

***The checksum is the module 256 sum of bytes CA+2 through N-1.

Figure 8-38. Absolute and PROM Absolute File Formats (Cont'd)

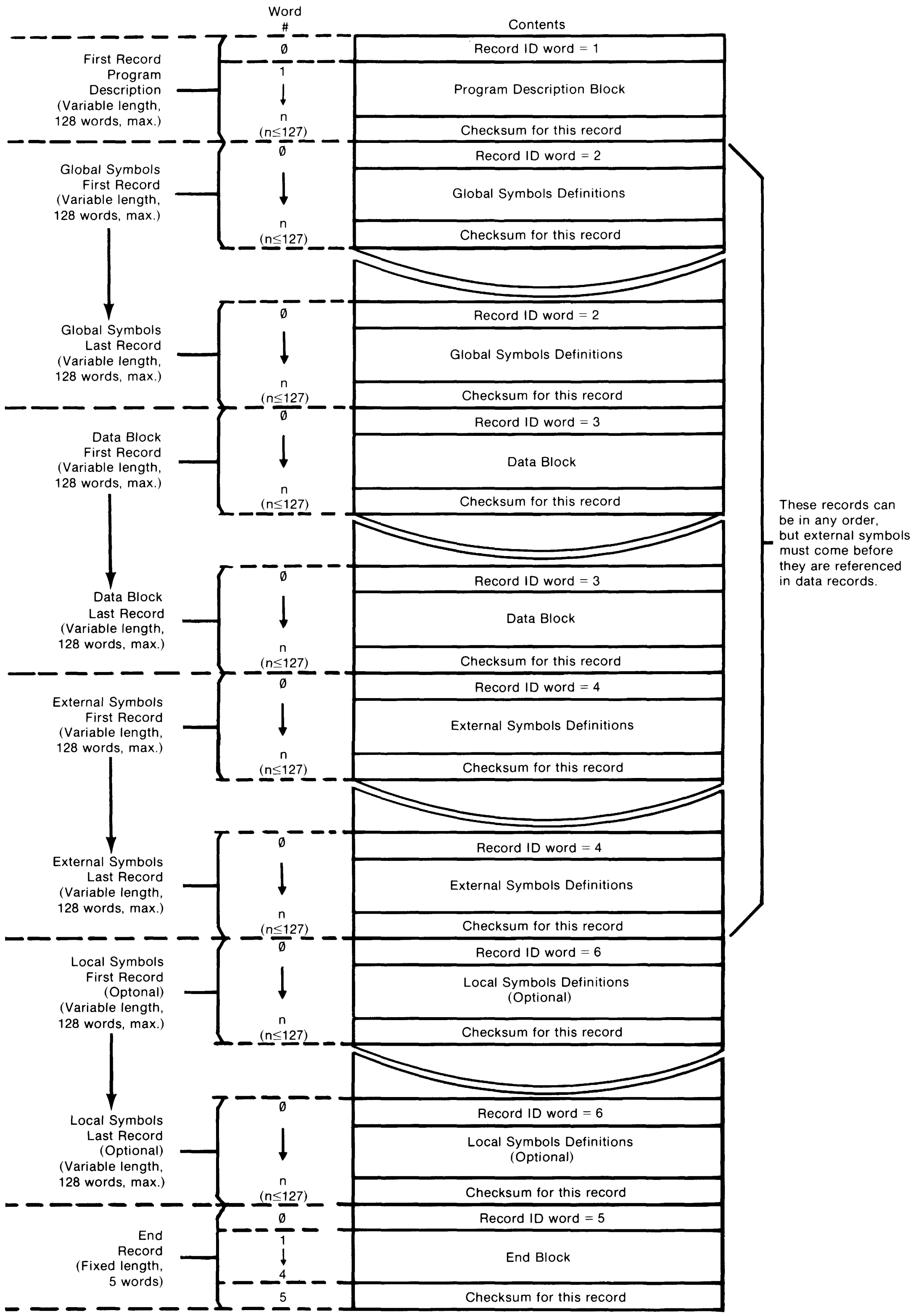
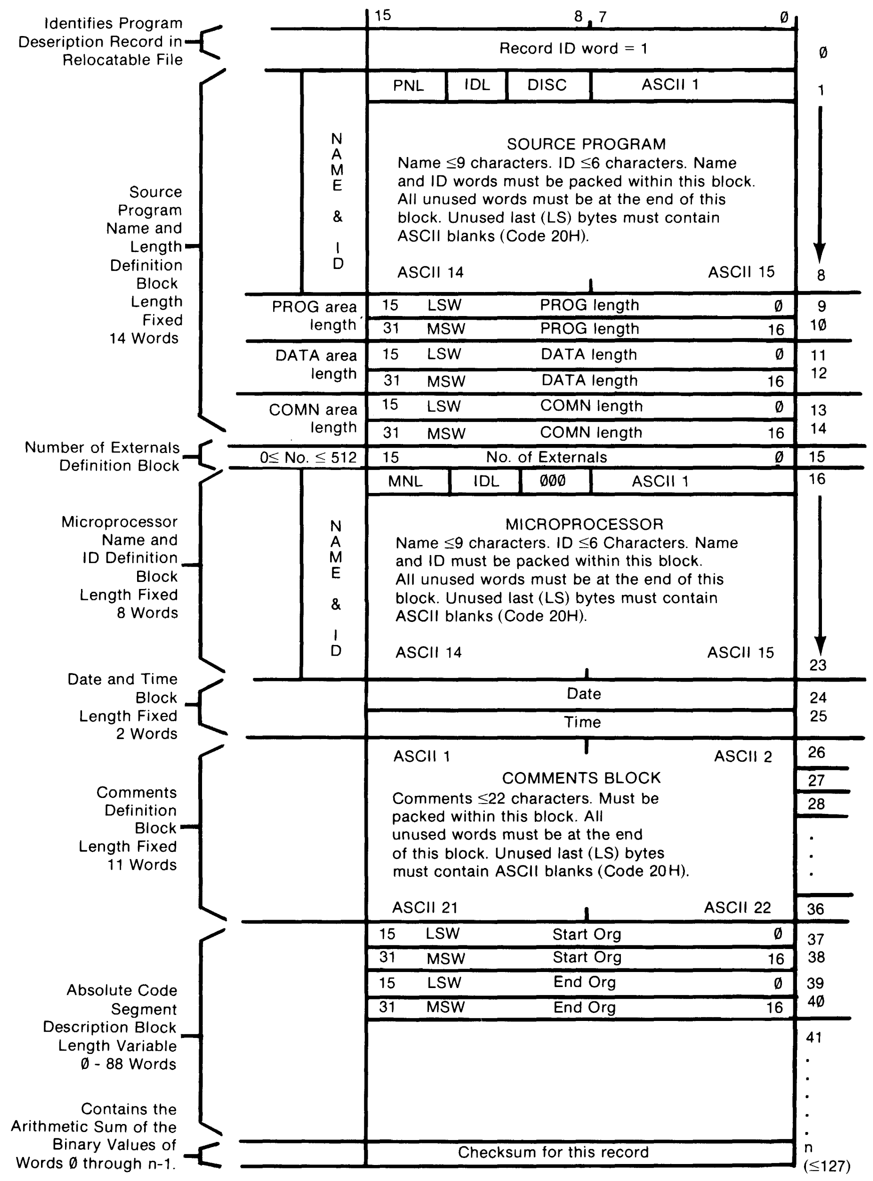

Word #
Contents
First Record Program Description (Variable length, 128 words, max.)
0
1
n (n≤127)
Record ID word = 1
Program Description Block
Checksum for this record
Global Symbols First Record (Variable length, 128 words, max.)
Record ID word = 2
Global Symbols Definitions
Checksum for this record
Global Symbols Last Record (Variable length, 128 words, max.)
Record ID word = 2
Global Symbols Definitions
Checksum for this record
Data Block First Record (Variable length, 128 words, max.)
Record ID word = 3
Data Block
Checksum for this record
Data Block Last Record (Variable length, 128 words, max.)
Record ID word = 3
Data Block
Checksum for this record
External Symbols First Record (Variable length, 128 words, max.)
Record ID word = 4
External Symbols Definitions
Checksum for this record
External Symbols Last Record (Variable length, 128 words, max.)
Record ID word = 4
External Symbols Definitions
Checksum for this record
These records can be in any order, but external symbols must come before they are referenced in data records.
Local Symbols First Record (Optonal) (Variable length, 128 words, max.)
Record ID word = 6
Local Symbols Definitions (Optional)
Checksum for this record
Local Symbols Last Record (Optional) (Variable length, 128 words, max.)
Record ID word = 6
Local Symbols Definitions (Optional)
Checksum for this record
End Record (Fixed length, 5 words)
0
1
4
5
Record ID word = 5
End Block
Checksum for this record

Figure 8-39. Relocatable File Overall Format



Name ≤9 characters. ID ≤6 characters. Name and ID words must be packed within this block. All unused words must be at the end of this block. Unused last (LS) bytes must contain ASCII blanks (Code 20H).

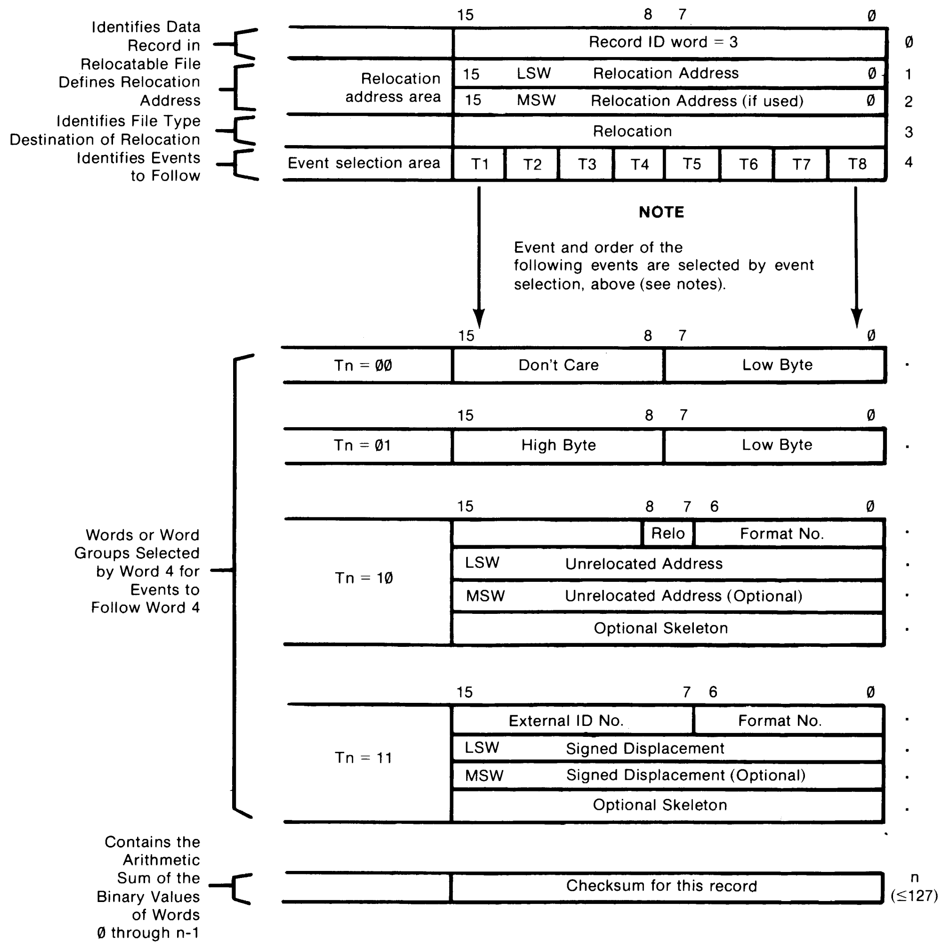
Name ≤9 characters. ID ≤6 Characters. Name and ID must be packed within this block. All unused words must be at the end of this block. Unused last (LS) bytes must contain ASCII blanks (Code 20H).

Comments ≤22 characters. Must be packed within this block. All unused words must be at the end of this block. Unused last (LS) bytes must contain ASCII blanks (Code 20H).

NOTES:

1. PNL and MNL = Number of 16-bit words-1 required to define program or microprocessor name. At least one character in the "ASCII 1" byte is required. Thus, with a one character name, PNL or MNL = Ø. If all nine characters are used (5 words) PNL or MNL = 4.
2. IDL = Actual number of 16-bit words required to define the user ID. If one word is used, IDL = 1. If all three words are used IDL = 3.
3. Disc (in program name segment) - The identifying number of the disc upon which the program resides.
4. Bits 1Ø, 9, and 8 in microprocessor name seqment always contain ØØØ.
5. ASCII bytes 1-15 contain the name and ID characters. These words must be packed. That is; the ID words must follow the name words. Unused words must be at the end of the block. An unused byte in either a name or ID word must contain an ASCII blank (Code 20H).
6. Length bytes or words - Contains the number of bytes or words (processor dependent) of code produced by the assembler or compiler in each of the three relocatable sections; PROG, DATA, COMN.
7. Number of externals - Contains the number of external variables and procedures defined in the module.
8. Comments - Contains up to 22 ASCII characters defined by the NAME psuedo in the assembler or compiler. All unused characters must contain ASCII blanks (Code 20H).
9. Absolute code segment description - Contains Ø to 22 entries of four 16-bit words. Each four word entry defines an absolute code segment declared in the assembler or compiler.

Figure 8-40. Relocatable File Program Description Definition Block


Identifies Data Record in Relocatable File
Defines Relocation Address
Identifies File Type Destination of Relocation
Identifies Events to Follow
15
8
7
Ø
Record ID word = 3
Relocation address area
15 LSW Relocation Address Ø
15 MSW Relocation Address (if used) Ø
Relocation
Event selection area
T1 T2 T3 T4 T5 T6 T7 T8
Ø
1
2
3
4
NOTE
Event and order of the following events are selected by event selection, above (see notes).
Words or Word Groups Selected by Word 4 for Events to Follow Word 4
Tn = ØØ
Don't Care
Low Byte
Tn = Ø1
High Byte
Low Byte
Tn = 1Ø
Relo
Format No.
LSW Unrelocated Address
MSW Unrelocated Address (Optional)
Optional Skeleton
Tn = 11
External ID No.
Format No.
LSW Signed Displacement
MSW Signed Displacement (Optional)
Optional Skeleton
Contains the Arithmetic Sum of the Binary Values of Words Ø through n-1
Checksum for this record
n (≤127)

NOTES:

1. Relocation Address Words - The relocation address words contain the relocation address assigned by the linker to this program. The MSW is used only when the ID offset = 3.

2. Relocation contains the binary code for area relocated to as follows: ØØ = ABS, Ø1 = PROG, 1Ø = DATA, and 11 = COMN.

3. Event Selection Area - Selects events to follow. T1 through T8 may contain any one of codes ØØ, Ø1, 1Ø, or 11. Codes are defined as follows: ØØ = one byte absolute with no modifications, Ø1 = two bytes absolute with no modifications, 1Ø = relocatable reference, and 11 = external reference. As T1 through T8 are read, the event selected by the specific code will be executed.

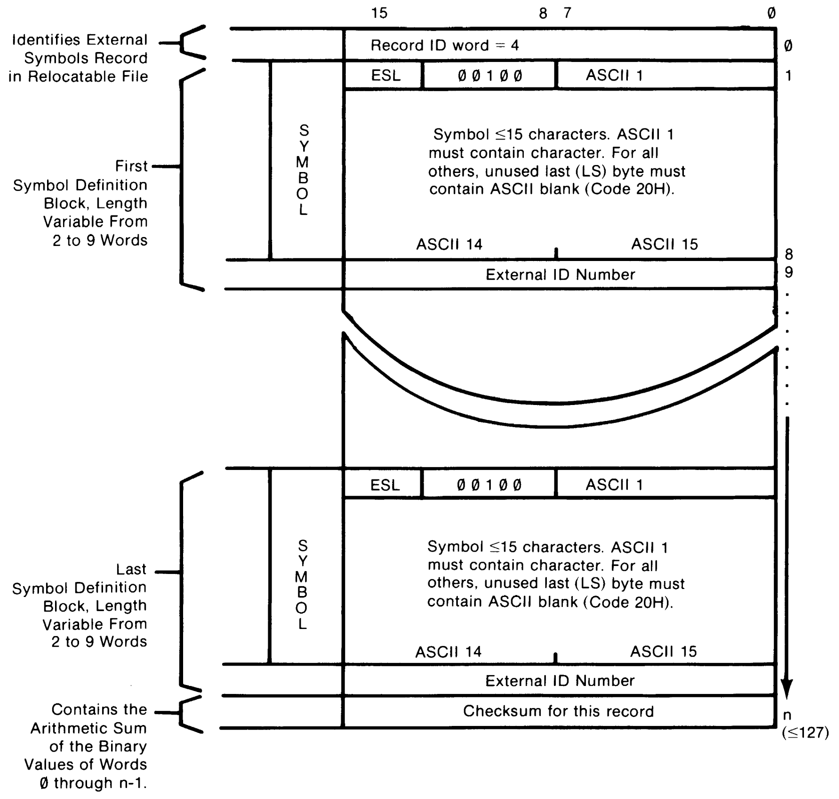
4. Tn = ØØ - Produce one byte of absolute code, which is found in the low order byte of the corresponding word.

5. Tn = Ø1 - Produce two bytes of absolute code, which is found in the corresponding word.

6. Tn = 1Ø - relocate the address to be found in the second word (and optionally, the third word) based on the relocation code in the first word. Then produce an absolute code based on the processor dependent format number in the first word and skeleton, if present.

7. Tn = 11 - look up the external symbol whose number is in the first word (which has been previously defined in a type 4 record). Add the displacement and then produce an absolute code based on format number and optional skeleton.

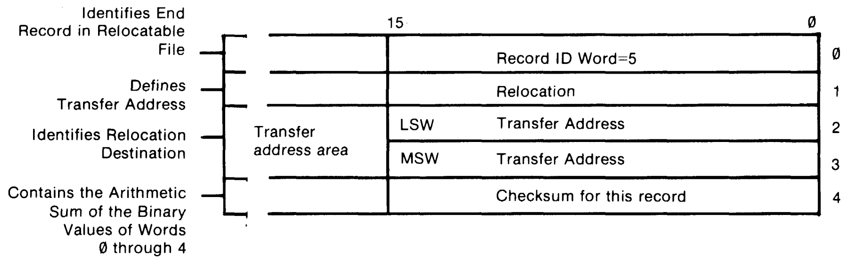
Figure 8-41. Relocatable File Data Definition Block


15
8 7
Ø
Identifies External Symbols Record in Relocatable File
Record ID word = 4
Ø
ESL
ØØ1ØØ
ASCII 1
1
First Symbol Definition Block, Length Variable From 2 to 9 Words
SYMBOL
Symbol ≤15 characters. ASCII 1 must contain character. For all others, unused last (LS) byte must contain ASCII blank (Code 20H).
ASCII 14
ASCII 15
8
External ID Number
9
Last Symbol Definition Block, Length Variable From 2 to 9 Words
ESL
ØØ1ØØ
ASCII 1
SYMBOL
Symbol ≤15 characters. ASCII 1 must contain character. For all others, unused last (LS) byte must contain ASCII blank (Code 20H).
ASCII 14
ASCII 15
External ID Number
Contains the Arithmetic Sum of the Binary Values of Words Ø through n-1.
Checksum for this record
n (≤127)

NOTES:

1. ESL = Number of 16-bit words required to define an external symbol. At least one character in the ASCII 1 byte is required. Thus, with a one character definition, ESL = 0. If all 15 characters are used (8 words) ESL = 7.
2. Bits 8 through 12 always contain ØØ1ØØ.
3. The bytes labeled ASCII 1-15 are the maximum number of bytes available to define the symbol. Only the actual number of 16-bit words required to define the symbol will exist. However, if the first byte (MSB) is used, then the second byte (LSB) must contain an ASCII blank (Code 20H).
4. External ID Number is assigned by the assemnbler or compiler. ID number is ≤511.

Figure 8-42. Relocatable File External Symbols Definition Block


Identifies End Record in Relocatable File
Defines Transfer Address
Identifies Relocation Destination
Contains the Arithmetic Sum of the Binary Values of Words Ø through 4
Transfer address area
15
Ø
Record ID Word=5
Relocation
LSW
Transfer Address
MSW
Transfer Address
Checksum for this record
Ø
1
2
3
4

Notes:

1. Relocation-contains the binary code for area relocated to as follows: ØØ=ABS, Ø1=PROG, 1Ø=DATA, 11=COMN, 1ØØ=No transfer address.

2. Transfer Address Words-Contains the address where control will be transferred to when the program is run. Only one module in a program may have a transfer address, and it is defined in the END label psuedo in the assembler or the presence of the main program block in a PASCAL module.

Figure 8-43. Relocatable File End Definition Block

Appendix A
Syntactical Variable Definitions

The syntactical variables used throughout this manual are described in this appendix.

<ABSFILE>

The <ABSFILE> variable is the file identifier of a absolute file. It has the same format requirements as the <FILE> variable which is described later in this appendix.

<ADDRESS> (or <ADDR>)

The <ADDRESS> variable defines a bit pattern of up to 16 bits which specifies a particular location in mapped memory. That bit pattern can be represented by a binary, octal, hexadecimal, or decimal number; a local or global symbol; or a mathematical combination of any two numbers or symbols. <ADDRESS> has the same format requirements as the <VALUE> variable which is described later in this appendix.

<CMDFILE>

The <CMDFILE> variable is the file identifier for an existing emulation configuration command file. This type of command file contains the answers entered in response to an emulation configuration question sequence and can be used to initialize the 64000 for an emulation session. <CMDFILE> has the same format requirements as the <FILE> variable which is described later in this appendix.

<EVENT>

The <EVENT> variable specifies a particular state or set of states on the emulation buses and/or the contents of the internal registers of the emulation processor.

An event can consist of any single state specified alone or one of the states logically "AND"ed with one or all of the other optional states shown in brackets, (**[]**). Within a series, any one or more of the bracketed states can be omitted but when specified they must be in the order shown. Note that the address can be specified as a range (<ADDR> thru <ADDR>) and that more than one register state can be specified. Register states are the only states which can be specified in multiple in the same event.

The <EVENT> variable consists of the following parameters:

$$\textbf{[}\text{address}\left\{\begin{array}{l}\left\{\begin{array}{c}<\;>\\=\end{array}\right\}\ \text{<ADDRESS>}\\ \text{range = <ADDR> thru <ADDR>}\end{array}\right\}\textbf{][}\text{data}\left\{\begin{array}{c}<\;>\\=\end{array}\right\}\ \text{<ADDRESS>}\textbf{]}$$

$$\textbf{[}\text{status}\left\{\begin{array}{c}<\;>\\=\end{array}\right\}\left\{\begin{array}{l}\text{<OPERATION>}\\ \text{<VALUE>}\end{array}\right\}\textbf{][}\text{register (X)} = \text{<VALUE>}\textbf{]}$$

[register (X) = <VALUE>**]**...etc.

where:

<ADDRESS> and <ADDR>	are entered as described earlier in this appendix.
<OPERATION>	refers to any of the operations performed by the microprocessor being emulated. A typical operation might be a memory read or write, an I/O read or write, or the execution of any opcode. The operations available for the emulated processor are listed as softkeys.
<VALUE>	specifies a state on the emulation bus or the contents of a register. The format requirements for <VALUE> are described later in this appendix.
address	is a key word which is used to identify a state or set of states on the emulation address bus.
data	is a key word which is used to specify a state on the emulation data bus. The range of valid data values depends upon the word size of the emulation microprocessor.
status	is a key word which is used to specify an operation on the emulation buses.
register (X)	is a key word which is used to specify the contents of one of the registers of the emulated processor. The names of the internal registers are represented by (X) and are displayed as softkeys.

A shorthand syntax may be used when specifying an <EVENT>. The words "address", "data", and "status" can be omitted as long as commas are used to separate the fields which determine the <EVENT>. For example, "address = 810H data = 0FFH status = write" could be entered as follows: 810H,0FFH,write. Likewise, "address = 900H status = read" could be entered as "900H,,read". Notice that when a particular field has no entry, commas must still be used to distinguish the fields. The first comma specifies the end of the address field and the second comma specifies the end of the data entry.

There is no shorthand syntax which replaces "register (X) = <VALUE>"; therefore, the longhand format must be used to specify the contents of a register.

<FILE>

The <FILE> variable is used to identify files generated or accessed by the development system commands. <FILE> consists of the following parameters:

<FILE NAME>**[**:<USERID>**][**:<DISC#>**]**

where:

<FILE NAME>	is the identifier given to a particular file. <FILE NAME> must begin with an upper case alphabetic character and can have a total length of nine characters. After the first character, any upper or lower case alphanumeric character or an underscore can be used. If more than nine characters are specified, the name is truncated to the first nine characters.
<USERID>	is the identifier assumed by a particular system user. <USERID> must begin with an upper case alphabetic character and can have a total length of six characters. The characters following the first character can be any upper or lower case alphanumeric characters, including the underscore. If more than six characters are specified, the userid is truncated to the first six characters. If a userid is not entered, the current userid is used as the default.
<DISC#>	specifies the memory disc on which the file is stored. <DISC#> can be any digit from 0 to 9, but it must correspond to the HP-IB Unit Address assigned to one of the discs installed in the system. The default is disc 0.

<POSITION>

<POSITION> is used as part of the trace command to determine whether the "states" placed in the trace buffer occur "before", "after", or "about" the specified trigger. <POSITION> can be one of the following parameters: **before** , **after** , or **about** .

<QUALIFIER>

The <QUALIFIER> variable is a series of <TERM>s connected by logical OR's which is used to specify the conditions for terminating a run or for triggering or terminating a trace. A <QUALIFIER> is entered according to the following syntax:

<TERM>**[**or <TERM>**][**or <TERM>**]**...etc.

where:

<TERM>	represents an <EVENT> which must occur a specified number of times. Refer to the definitions for <TERM> and <EVENT> which are included in this appendix.
or	is the logical OR condition.

<STATES>

The <STATES> variable specifies a particular state or set of states on the emulation buses. <STATES> can consist of any single state specified alone or one of the states logically "AND"ed with one or all of the other optional states shown in brackets, (**[ ]**). Within a series, any one or more of the bracketed states can be omitted but when specified, they must be in the order shown. Note that the address can be specified as a range (<ADDR> thru <ADDR>). <STATES> are entered according to the following syntax:

$$\textbf{[}\text{address}\left\{\begin{matrix}\left\{\begin{matrix}<\quad>\\=\end{matrix}\right\}\text{<ADDRESS>}\\ \text{range}\left\{\begin{matrix}<\quad>\\=\end{matrix}\right\}\text{<ADDR> thru <ADDR>}\end{matrix}\right\}\textbf{][}\text{data}\left\{\begin{matrix}<\quad>\\=\end{matrix}\right\}\text{<VALUE>}\textbf{]}$$

$$\textbf{[}\text{status}\left\{\begin{matrix}<\quad>\\=\end{matrix}\right\}\left\{\begin{matrix}\text{<OPERATION>}\\ \text{<VALUE>}\end{matrix}\right\}\textbf{]}$$

where:

<ADDRESS> and <ADDR>	are entered as described earlier in this appendix.
<OPERATION>	refers to any of the operations performed by the microprocessor being emulated. A typical operation might be a memory read or write, an I/O read or write, or the execution of any opcode. The operations available for the emulated processor are listed as softkeys.
<VALUE>	specifies a state on the emulation bus or the contents of a register. The format requirements for <VALUE> are described later in this appendix.
address	is a key word which is used to identify a state or set of states on the emulation address bus.
data	is a key word which is used to specify a state on the emulation data bus. The range of valid data values depends upon the word size of the emulation microprocessor.
status	is a key word which is used to specify an operation on the emulation buses.

A shorthand syntax may be used when entering the information required by the <STATES> variable. The words "address", "data", and "status" can be omitted as long as commas are used to separate the fields which contain the entries for each state. For example, "address = 810H data = 0FFH status = write" could be entered as follows: 810H,0FFH,write. Likewise, "address = 900H status = read" could be entered as "900H,,read" using the shorthand syntax. Notice that when a particular field has no entry, commas must still be used to distinguish the fields. The first comma specifies the end of the address field and the second comma specifies the end of the data entry.

<TERM>

The <TERM> variable represents a particular state or set of states on the emulation buses and/or the contents of the internal registers of the emulation processor. The state or states must occur the specified number of times before the requirements of the run or trace command in which the <TERM> appears are fulfilled.

<TERM> consists of the following parameters:

<EVENT>**[**occurs <#TIMES>**]**

where:

<EVENT>	represents a particular state or set of states on the emulation buses and/or the contents of the internal registers of the emulation processor. For more information, refer to the description of <EVENT> which is included in this appendix.
<#TIMES>	represents a decimal integer in the range of 1 to 65535.

<VALUE>

<VALUE> is a syntactical variable that allows specification of symbols (labels), numbers, and math operators (+, −, /, *). A <VALUE> is specified using to the following syntax:

<NUMBER>	+	<NUMBER>
<LOCAL SYMBOL>**[**:<FILE>**]**	-	<LOCAL SYMBOL>**[**:<FILE>**]**
<GLOBAL SYMBOL>**[**:<FILE>**]**	/	<GLOBAL SYMBOL>**[**:<FILE>**]**
	*	

where:

<NUMBER>	is an alphanumeric representation of a 16 bit pattern of ones, zeros, and don't cares (X's). The bit pattern can be represented in binary, octal, hexadecimal, or decimal where binary is indicated by a "B", octal by a "Q", hex by an "H", and decimal by a "D". Decimal is the default value.

Examples:

10101011XXXXXXXB
145XXXQ
2563
START + 5

<LOCAL SYMBOL> and <GLOBAL SYMBOL>

The <LOCAL SYMBOL> variable represents the name of a symbol which can only be used by the program module in which it is defined. The <GLOBAL SYMBOL> variable represents the name of a symbol which can be called by program modules other than the one in which it is defined. The global symbol must be declared as such by a GLB statement in the source file.

<LOCAL SYMBOL> and <GLOBAL SYMBOL> can consist of up to 15 upper or lower case alphanumeric characters including the underline symbol. In each case, however, the first character must be an upper case alphabetic character.

<FILE>

specifies the file in which the local or global symbol is defined. If no <FILE> is specified, the global symbol table associated with the absolute program file loaded by the emulator is searched for the <LOCAL SYMBOL> name. If the symbol name is not found in the global symbol table, a search is made of the last referenced local symbol table. If the symbol name is not found in the local symbol table, an error message is displayed on the status line. For more information, refer to the description of <FILE> which is included in this appendix.

Appendix B
Emulation Command Descriptions

The commands available in the emulation mode are described in this appendix. For more information on a particular command, refer to its description in the appropriate chapter.

SYNTAX

copy <ADDRESS>**[**thru<ADDRESS>**]** to <FILE>

Default Value

none

Examples:

copy 10A0H to TEMP1

copy 800H thru 20FFH to TEMP2

copy EXEC thru DONE to TEMP3

FUNCTION

The copy command is used to store the contents of specific memory locations in an absolute file on a disc without altering the contents of memory. Either a single memory location (<ADDRESS>) or a series of locations (<ADDRESS> thru <ADDRESS>) can be specified for transfer.

SYNTAX

$$\textbf{count} \begin{bmatrix} \text{time} \\ \text{<STATES>} \end{bmatrix}$$

Default Value

If neither time nor a set of conditions for <STATES> is entered, the 64000 defaults to counting time.

Examples:

count time

count 810H,,30H register h = 14H

count address = 900H data = 74H

FUNCTION

The count command is used to measure the elapsed time between the states of a trace or to determine the number of times a specified <STATE> occurs within a trace.

SYNTAX

```
           { count [absolute
                    relative]
           
             global
           
             local <FILE>
           
display      memory[<ADDRESS>][dynamic][mnemonic][offset_by<ADDRESS>]
           
             registers [dynamic][offset_by <ADDRESS>]
           
             run_spec
           
             trace [absolute][unpacked][offset_by <ADDRESS>]
           
             trc_spec }
```

Default Values

dynamic	- If dynamic is omitted, the display is static and is not updated as the contents of the registers change except when registers are modified using the modify command.
offset_by	- If offset_by is omitted, the actual program counter values of addresses are displayed.
absolute	- If absolute is omitted, the data is displayed in opcode mnemonic form.
unpacked	- If unpacked is omitted, the operators and operands are displayed together on a single line. For example, STA 8200H would be displayed as:

STA 8200H

Examples:

display count relative

display local_symbols_in TEMP1

display memory 800H

display memory dynamic mnemonic

display registers dynamic offset_by 0A10H

display trace unpacked offset_by 900H

display trace_specification

FUNCTION

The display command initiates the display of the time or state counts, local or global symbols, the contents of registers or memory, the current run or trace specification, or the contents of the trace buffer.

edit_cnfg

SYNTAX

edit_cnfg

Default Value

none

Example:

edit_configuration

FUNCTION

The edit_configuration command allows the question_answer sequence of the current emulation configuration to be reviewed and edited.

emulate

SYNTAX

$$\textbf{emulate}\left[\begin{array}{l}\text{<CMDFILE>[load <ABSFILE>][options }\left\{\begin{array}{l}\text{edit}\\ \text{continue}\end{array}\right\}\text{]}\\ \text{load <ABSFILE>}\end{array}\right]$$

Default Value

<CMDFILE>	- If no command file is specified, the system will run through the emulation configuration question-answer sequence and a new command file may be created.

Examples:

emulate

emulate CMDF1

emulate CMDF1 options edit

emulate load JR8085

FUNCTION

The emulate command intiates an emulation session using either an existing emulation command file or the emulation configuration question-answer sequence. It can also initiate the loading of an absolute file into either emulation or user memory.

SYNTAX

end

Default Value

none

Example:

end_emulation

FUNCTION

The end_emulation command teminates the current emulation session and returns the 64000 operating system to the monitior mode.

Execution of program code by the emulation processor and the execution of a trace command are unaffected by the end command.

SYNTAX

$$\textbf{list} \quad \left\{ \begin{array}{l} \text{printer} \\ \text{file} \end{array} \right\}$$

Default Value

none

Examples:

list_display_to printer

list_display_to TEMP1

FUNCTION

The list command produces a copy of the information currently displayed on the CRT. The copy can be either a listing file stored in the 64000 memory or a hardcopy produced by the printer.

SYNTAX

load <FILE>

Default Value

none

Example:

load_memory JR8085

FUNCTION

The load command transfers absolute code from the 64000 system memory into user RAM or emulation memory. The destination of the the absolute code is determined by the memory configuration map which was set up during emulation configuration.

SYNTAX

```
        ⎧ memory <ADDRESS> to <VALUE>[,<VALUE>]...  ⎫
modify  ⎨ <ADDRESS> thru <ADDRESS> to <VALUE>       ⎬
        ⎩ register (X) to <VALUE>[(X) to <VALUE>]...⎭
```

Default Value

none

Examples:

modify memory 800H to 10H

modify memory 910H to 0CH,56H,36H

modify memory 0A10H thru 0AFFH to 00

modify register a to 39H

modify register h to 0AH l to 50H a to 18H

FUNCTION

The modify command is used to modify either the contents of memory or the contents of the processor registers.

SYNTAX

restart

Default Value

none

Example:

restart_processor

FUNCTION

The restart command causes the emulation processor to go through a processor dependent power_up sequence.

SYNTAX

$$\textbf{run [}\text{from <ADDRESS>}\textbf{][}\text{until} \left\{ \begin{array}{l} \text{trc\_cmplt} \\ \text{<ADDRESS>} \\ \text{<TERM>}\textbf{[}\text{or <TERM>}\textbf{]}\text{...} \end{array} \right\} \textbf{]}$$

Default Values

from <ADDRESS>	- If the from <ADDRESS> option is omitted, the emulator will begin program execution at the current address specified by the processor's next program counter.
until	- If the until option is omitted, the processor will continually execute program instructions.

Examples:

run

run from 810H

run from 810H **until** address = 84AH data = 60H

FUNCTION

The run command controls the execution of the emulation program.

SYNTAX

step [<# STEPS>**][**from <ADDRESS>**]**

Default Values

<# STEPS>	- If no value is entered for number of times, only one instruction is executed each time the RETURN key is pressed. However, multiple instructions can be executed by holding down the RETURN key.
from <ADDRESS>	- If the from <ADDRESS> option is omitted, stepping begins at the next program counter address.

Examples:

step

step from 810H

step 20 **from** 0A40H

FUNCTION

The step command allows program instructions to be sequentially analyzed by causing the emulation processor to execute a specified number of instructions. The contents of the processor registers may be displayed after each instruction is executed and the contents of memory can be displayed upon completion of the step command.

SYNTAX

$$\textbf{stop} \quad \left\{ \begin{array}{l} \text{run} \\ \text{trace} \end{array} \right\}$$

Default Value

none

Examples:

stop run

stop trace

FUNCTION

The stop command terminates either the current program run or trace command.

SYNTAX

```
trace    [[sequence <TERM>[ { or | then } <TERM>]...

         [restart_on <TERM> [or <TERM>]...]

         [then <TERM>[ { or | then } <TERM>]...[restart_on <TERM>

         [or <TERM>]...]]...] (trigger) <POSITION><TERM>

         [or <TERM>]...][(trc_)only <EVENT>[or <EVENT>]...]

         [continue]
```

Default Value

If the pretrigger and trigger specifications are omitted, the trigger <POSITION> defaults to after and the trigger <TERM> defaults to "don't cares".

Examples:

trace

trace continuous

trace only 810H or 900H,10H

trace after address = 54H continuous

trace in_sequence ,21H occurs 5 then 800H or 801H
restart on data = 25H trigger about 10A0H

trace in_sequence 610H or 654H occurs 3 then 680H
trigger after 685H trace_only data = 0FFH

FUNCTION

The trace command is used to analyze and display a particular portion of a program run. Each state of the program run is examined and if that state helps fulfill the requirements of the trace specificaton, it is stored in a trace buffer which can store up to 256 states. When the trace specification is satisfied, the contents of the trace buffer is displayed on the CRT.

Appendix C
Example System Hardware/Software

The Example System

Most of the examples in this manual have been taken from a system of hardware and software that can be obtained and exercised by the user of the Model 64000 development system. The hardware is the HP Model 5036A Microprocessor Laboratory in which an optional zero insertion force socket has replaced the normal microprocessor socket. This socket allows ready access to the system by the processor probe of the Model 64000 emulation and analysis system. The software is a collection of three software modules which are entitled "EXEC", "DISP", and "KYBRD". "DISP" and "KYBRD" service the seven segment display and the keyboard of the 5036A, respectively. The "EXEC" routine controls the interface between the service routines.

Example System Design

The example system is designed to be easily understood and to provide a continuity in the examples of this manual. The examples will help you learn quickly the capabilities of the Model 64000 emulation and analysis system. The system is also useful in learning other aspects of the 64000. For example, the three program modules may be entered in the editor, assembled and linked, and emulated in the emulation and analysis system. The programs may then be relocated and programmed into I2716 EPROMs. Then the EPROMs may be used to replace the operating system EPROM in the microprocessor lab and the program can be executed without the use of the emulation equipment.

Where to get Information

Information concerning the HP Model 5036A Microprocessor Lab may be found in any 1979 or later Hewlett-Packard catalog. Contact your local Hewlett-Packard representative for further information.

The software is simple in concept and function. The EXEC routine presets a buffer in RAM to six digit positions of an underline character. It then "calls" the keyboard service routine (called "KYBRD") to scan the hex keypad of the Model 5036A for any input characters. When a valid character is found, it is put into the buffer and all the previous entries are shifted to higher positions. The EXEC routine next calls the "DISP" routine to display a single scan of the buffer contents. When the display scan is completed, the scan keyboard, scan display cycle continues until the processor is halted.

Commented listings of these program modules are included in this appendix and may be manually entered into the Model 64000 using the editor. When entered and executed in a Model 5036A with an I8085 emulation and analysis system, you will be able to duplicate the examples of this manual with your own system.

```
´8080´
****************************************************************
*
*
*   THIS PROGRAM IS THE "EXECUTIVE" DRIVE FOR THE HP64000
*   DEMONSTRATION TARGET SYSTEM SOFTWARE.  THIS ALGORITHM
*   READS NUMERIC HEX INPUT FROM THE KEYBOARD AND DISPLAYS
*   THE DATA IN THE SEVEN SEGMENT DISPLAY. ALL CONTROL KEYS
*   EXCEPT RESET ARE IGNORED.  AS NEW DATA IS ENTERED, THE DIS-
*   PLAY SHIFTS LEFT AND THE NEW ENTRY IS DISPLAYED IN THE LEAST
*   SIGNIFICANT DIGIT POSITION.
*
*   TWO EXTERNAL PROGRAMS ARE REQUIRED FOR OPERATION.
*       (1) KYBRD......A ROUTINE THAT SERVICES THE KEYBOARD
*       (2) DISP.......A ROUTINE THAT SERVICES THE SEVEN
*                      SEGMENT DISPLAY.
*
*   THIS PROGRAM PERFORMS THE FOLLOWING FUNCTIONS:
*       (1)  INITIALIZE THE STACK POINTER
*       (2)  CALL THE KEYBOARD SERVICE ROUTINE
*       (3)  CHECKS FOR NON CHARACTER BETWEEN VALID INPUT
*            CHARACTERS
*       (4)  SHIFT DISPLAY BUFFER (0800H TO 805H)
*       (5)  CALL THE DISPLAY SERVICE ROUTINE TO DISPLAY
*            CONTENTS OF THE DISPLAY BUFFER
*       (6)  REPEATS STEPS 1 THROUGH 5 ABOVE
*
*
*
****************************************************************
         ORG 0810H
         EXT KYBRD,DISP
EXEC     LXI SP,0C00H         LOAD THE STACK POINTER TO C00H
         LXI H,805H           INITIALIZE THE H,L REGISTERS
         MVI A,0F7H           INITIALIZE UNDERLINE DISPLAY FOR BUFFERS
LP1      MOV M,A              GET DATA
         DCR L                DECREMENT BUFFER POINTER
         JP LP1               POINTER AT END OF BUFFER? NO, RETURN TO LP1
LP2      CALL KYBRD           YES, CALL KEYBOARD ROUTINE
         JNC XX               DATA INPUT? YES, GO TO XX
         XRA A                NO, CLEAR THE ACCUM
         CMA                  COMPLEMENT THE ACCUM
         STA 0806H            STORE FFH IN LAST FOUND DATA LOCATION
         JMP LIGHT            GO TO LIGHT TO CALL DISPLAY ROUTINE
XX       LXI H,0806H          LOAD H,L WITH LAST FOUND DATA ADDRESS
         CMP M                COMPARE THIS INPUT WITH LAST
         JZ LIGHT             SAME? YES, IGNORE INPUT, GO TO LIGHT
         MOV M,A              NO, STORE CURRENT DATA IN LAST FOUND DATA
         PUSH PSW             STORE CURRENT INPUT DATA
         LXI H,0805H          INITIALIZE REGISTERS FOR BUFFER SHIFT
         LXI D,0804H
GO1      LDAX D               SHIFT BUFFER TO THE LEFT, GET DATA FROM MEM
         MOV M,A              STORE IN THE NEXT HIGHER ADDRESS LOCATION
         DCR E                DECREMENT THE BUFFER POINTERS
         DCR L
         JNZ GO1              DONE? NO, RETURN TO GO1
         POP PSW              YES, RESTORE INPUT DATA TO ACCUM AND
         MOV M,A              STORE IN LOWEST BUFFER LOCATION
LIGHT    CALL DISP            CALL THE DISPLAY SERVICE ROUTINE
         JMP LP2              GO TO THE BEGINNING, LP2
         END
```

```
****************************************************************
*
*
*   THIS IS A SUBROUTINE THAT IS CALLED BY THE EXEC DRIVE OF THE
*   HP64000 TARGET SYSTEM SOFTWARE.  "KYBRD" SERVICES THE KEYBOARD
*   AND RETURNS TO THE EXECUTIVE ROUTINE WITH THE INPUT DATA (IN HEX)
*   IN THE ACCUMULATOR.  IN THE EVENT THAT NO KEY HAS BEEN PRESSED
*   DURING KEYBOARD SCAN, THE ROUTINE RETURNS TO "EXEC" WITH THE
*   CARRY BIT SET.
*
*   THIS ROUTINE PERFORMS THE FOLLOWING FUNCTIONS
*          (1)  LOAD THE SCAN REGISTER
*          (2)  READ THE KEYBOARD INPUT REGISTER
*          (3)  TEST FOR AN ACTIVE KEY
*          (4)  IF A KEY IS ACTIVATED, DEBOUNCE THE KEY BY REQUIRING
*               THE SAME KEY TO BE READ 128 TIMES IN SUCCESSION WITHOUT
*               MISSING ONE TIME
*          (5)  IF DEBOUNCE IS SUCCESSFUL, CONVERT THE INPUT DATA TO A
*               HEX CHARACTER
*          (6)  RETURN TO THE EXECUTIVE WITH HEX DATA IN THE ACCUMULATOR
*               AND THE CARRY BIT RESET
*               IF THE DEBOUNCE IS NOT SUCCESSFUL, CONTINUE THE SCAN
*          (7)  IF NO KEY STROKE IS FOUND, RETURN TO "EXEC" WITH THE
*               CARRY BIT SET
*
*
*
****************************************************************
        ORG 900H
        GLB KYBRD
KYBRD   PUSH H              SAVE REGISTERS
        PUSH D
        PUSH B
        LXI B,00FEH         INITIALIZE B,C TO SCAN BYTES
READ    MOV A,C             LOAD SCAN REGISTER
        STA 2800H
        LDA 1800H           READ KEYBOARD REGISTER
        CPI 0FFH            IS A KEY ACTIVE? YES, CALL DEBOUNCE ROUTINE
        CNZ DBNC
        JC FOUND            DEBOUNCE SUCCESSFUL? YES, GO FOUND
CONT    INR B               NO, OR NO INPUT, INCREMENT SCAN COUNTER
        MOV A,C             ROTATE SCAN BYTE
        RLC
        MOV C,A             RESTORE SCAN BYTE
        JC READ             SCAN DONE? NO, RETURN TO READ
        STC                 SET CARRY BIT
        JMP QUIT            JUMP TO QUIT
FOUND   CPI 0F7H            CHARACTER FOUND = F7H? YES, CONTINUE SCAN
        JZ CONT                 AT CONT
        CMA                 NO, COMPLEMENT THE ACCUMULATOR
        CPI 4H              ACCUM=4H? NO, SKIP NEXT INSTRUCTION
        JNZ FNDA
        DCR A               DECREMENT THE ACCUMULATOR 4=>3
FNDA    MOV D,A             STORE DATA IN D
        MOV A,B             GET SCAN COUNT
        ADD B               COMPUTE 3 TIMES SCAN COUNT
        ADD B
        ADD D               ADD INPUT DATA
```

```
         CPI 0AH          IS RESULT GREATER THAN AH? YES, GO TO HEX
         JP HEX           NO, CONTINUE
         CPI 7H           IS RESULT = 0H?  YES, GO TO ZERO
         JZ ZERO          NO CONTINUE
         JMP CONT         DATA IS NOT VALID, CONTINUE SCAN AT CONT
ZERO     XRA A            DATA IS ZERO, CLEAR ACCUMULATOR
         ANA A            CLEAR CARRY FLAG
         JMP QUIT         GO TO QUIT
HEX      SUI 9H           DATA IS HEX, SUBTRACT 9H TO CONVERT
         ANA A            CLEAR CARRY FLAG
         JMP QUIT         GO TO QUIT
QUIT     POP B            PREPARE TO RETURN TO CALLING ROUTINE
         POP D            RESTORE REGISTERS
         POP H
         RET              RETURN
DBNC     PUSH D           DEBOUNCE ROUTINE, SAVE THE D REGISTER PAIR
         MOV D,A          STORE KEY FOUND IN D
         MVI E,80H        INITIALIZE E TO 80H
DBLP     LDA 1800H        READ INPUT REGISTER
         CMP D            IS IT THE SAME AS D (PREVIOUS DATA)
         JNZ BAD          NO, GO TO BAD
         DCR E            YES, DECREMENT ITERATION COUNTER
         JNZ DBLP         COUNTER=0H? NO, RETURN TO DBLP
         STC              YES, SET CARRY FLAG
DDN      POP D            RESTORE D,E REGISTER
         RET              RETURN TO DISP ROUTINE
BAD      ANA A            DATA IS NOT THE SAME, KEY BOUNCED,
         JMP DDN          CLEAR CARRY FLAG, RETURN TO DDN
         END
```

```
´8080´
*******************************************************************
*
*
*
*   THIS PROGRAM SERVICES THE SEVEN SEGMENT DISPLAY FOR A SINGLE SCAN.
*   IT IS CALLED BY THE 64000S TARGET SYSTEM SOFTWARE "EXECUTIVE"
*   ROUTINE.  SINCE THIS ROUTINE SCANS ONE TIME.  IT MUST BE CALLED
*   CONTINUALLY WHEN DATA IS TO BE DISPLAYED.  DATA CONTAINED IN
*   THE DISPLAY BUFFER (800H TO 805H) IN HEX, IS CONVERTED TO SEVEN
*   SEGMENT DATA AND LOADED INTO THE DISPLAY REGISTER.  IF THE DATA
*   IN THE DISPLAY BUFFER IS GREATER THAN AH, NO CONVERSION IS MADE,
*   AND THE DATA IS DISPLAYED AS FOUND IN THE BUFFER.
*
*   THIS PROGRAM PERFORMS THE FOLLOWING FUNCTIONS:
*       (1)  READ THE DISPLAY BUFFER
*       (2)  CONVERT THE DATA TO SEVEN SEGMENT
*       (3)  LOAD THE SCAN REGISTER
*       (4)  LOAD THE DISPLAY REGISTER
*       (5)  WAIT DURING EACH SEGMENT
*       (6)  DO THE ABOVE FOR SIX DIGITS
*       (7)  CONVERSION FROM HEX TO SEVEN SEGMENT IS DONE BY TABLE LOOK UP
*
*
*
*******************************************************************
         ORG 0A00H
         GLB DISP
DISP     PUSH H               SAVE REGISTERS
         PUSH B
         PUSH D
         LXI H,800H           INITIALIZE BUFFER POINTER
         MVI C,1H             INITIALIZE SCAN BUFFER
LP1      MOV A,M              GET DATA
         CALL CNVT            CONVERT DATA TO SEVEN SEGMENT
         STA 3800H            LOAD DISPLAY BUFFER
         MOV A,C
         STA 2800H            LOAD SCAN REGISTER
         MVI A,0FFH           WAIT DECREMENT FROM 256
A1       DCR A
         JNZ A1
         INX H                INCREMENT BUFFER POINTER
         MOV A,C
         RLC                  ROTATE SCAN BYTE LEFT
         MOV C,A
         JNC LP1              SCAN COMPLETE? NO, GO TO LP1
         XRA A                YES, CLEAR ACCUMULATOR
         CMA                  COMPLEMENT ACCUMULATOR
         STA 3800H            LOAD FF INTO DISPLAY REGISTER TO TURN OFF DISPLAY
         POP D                RESTORE REGISTERS
         POP B
         POP H
         RET                  RETURN TO CALLING SUBROUTINE
CNVT     PUSH H               CONVERSION ROUTINE, SAVE H,L
         LXI H,CONV           LOAD H,L WITH ADDRESS OF CONVERSION TABLE
         CPI 10H              IS ACCUMULATOR GREATER THAN AH?
         JNC X                YES, GO TO X
         ADD L                NO, ADD REGISTER L TO ACCUMULATOR
         MOV L,A           RETURN THE SUM TO L
         MOV A,M           GET CONVERSION BYTE
X        POP H             RESTORE H,L
         RET               RETURN TO DISP PROGRAM
CONV     DB 0C0H           CONVERSION TABLE FOR 0
         DB 0F9H                                1
         DB 0A4H                                2
         DB 0B0H                                3
         DB 099H                                4
         DB 092H                                5
         DB 082H                                6
         DB 0F8H                                7
         DB 080H                                8
         DB 098H                                9
         DB 088H                                A
         DB 083H                                B
         DB 0C6H                                C
         DB 0A1H                                D
         DB 086H                                E
         DB 08EH                                F
         END
```

Appendix D
Emulator Electrical Properties

The emulation equipment, when connected to a target system, will respond similarly to the microprocessor it emulates. The timing of the processor signals at the probe closely approximates the timing of the microprocessor normally inserted in the same plug. Voltage and current requirements for the drive and receive circuitry of the emulator are generally equivalent to LS TTL specifications. The capacitive loading of the emulation probe is equivalent to the LS TTL gate capacitance plus the capacitance of the probe cable, which is approximately 20 pF.

The clocks supplied to microprocessor chips generally come in one of three forms. Clocks are supplied at MOS levels as high as 15 volts, at TTL levels, or are generated by an on-board oscillator in conjunction with an external crystal. The emulator processors of the 64000 allow all of these methods for driving the emulator clock. Any external clock or crystal frequency within the range specified for a specific emulator may be used. The frequency of the external clock is fixed and varies with the oscillator type.

The loading values and specifications for the clock options are as follows:

CLOCK, High-level

Levels: as specified by processor

$$C = C_{spec} + C_{cable}$$

CLOCK, TTL

Levels: TTL as specified by processor

$$C = C_{TTLS\ gate} + C_{cable}$$

CLOCK, Crystal Inputs

The emulator processor accepts frequency determining elements as specified by the processor manufacturer unless otherwise noted.

Generally, an additional delay of one TTLS gate is associated with each input and output of the emulator, with respect to the corresponding delay specification of the actual processor. It should be noted that because of reduced capacitive loading on the internal emulator processor, and because the fastest available processors, are used, the emulator may actually be faster than the standard and midrange versions of the actual processor.

When emulating processors that have an internal dynamic RAM refresh or DMA capability, the emulator pod will continue to execute these functions when emulation is not in process.

Some critical emulator signals may exhibit characteristics different than explained above. The operating manual supplement for your emulator should be referenced for details on these signals.

NOTE

The emulation pod presents greater drive capability and slightly greater capacitive loading to the target system than the processor being replaced. Consequently, it is conceivable that a user's system, which operates under emulation, may not operate properly when driven by a microprocessor IC (or vice-versa). Noise margins and signal levels in marginally overloaded designs may not cause problems when driven by emulation but may be fatal to system operation under normal microprocessor drive conditions. Be sure that your design does not exceed the loading specification for the actual processor, and that it can provide sufficient drive for the emulator inputs.

64980-90996

HEWLETT
PACKARD

PRINTED IN U.S.A.