HP 64762/3

8086/8088 Emulators PC Interface

User's Guide

64762-97002
Printed in U.S.A.
August, 1990

Edition 4

Certification and Warranty

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 environment 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.

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.

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Hewlett-Packard Company
Logic Systems Division
8245 North Union Boulevard
Colorado Springs, CO 80920, U.S.A.

Printing History

New editions are complete revisions of the manual. The date on the title page changes only when a new edition is published.

A software code may be printed before the date; this indicates the version level of the software product at the time the manual was issued. Many product updates and fixes do not require manual changes, and manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual revisions.

Edition 1	64762-90903, July 1988 E0788
Edition 2	64762-90903, February 1989 E0289
Edition 3	64762-97000, August 1989
Edition 4	**64762-97002, August 1990**

Using this Manual

This manual shows you how to use the HP 64762/3 (8086/88) emulators with the PC Interface.

This manual:

- Shows you how to use emulation commands by executing them on a sample program and describing their results.
- Shows you how to use the emulator in-circuit (connected to a target system).
- Shows you how to configure the emulator for your development needs. Topics include:
  - Restricting the emulator to real-time execution.
  - Selecting a target system clock source.
  - Allowing the target system to insert wait states.

This manual does not:

- Show you how to use every PC Interface command and option. The *Emulator PC Interface Reference* describes commands and options in detail.

Organization

Chapter 1 "Introduction to the 8086/8088 Emulator." This chapter briefly introduces you to emulation concepts and lists the basic features of the 8086/8088 emulator.

Chapter 2 "Getting Started." This chapter shows you how to use emulation commands by executing them on a sample program. This chapter describes the sample program and how to:

- Load programs into the emulator.
- Map, display, and modify memory.
- Display registers.
- Step through programs.
- Run programs.
- Set software breakpoints.
- Search memory for data.
- Use the analyzer.

Chapter 3 "In-Circuit Emulation." This chapter shows you how to install the emulator probe into a target system and how to use "in-circuit" emulation features.

Chapter 4 "Configuring the 8086/8088 Emulator." This chapter shows you how to:

- Restrict the emulator to real-time execution.
- Select a target system clock source.
- Make background cycles visible to the target system.
- Allow the target system to insert wait states.
- Select foreground or background emulation monitors.
- Add custom code to the background monitor.
- Allow DMA accesses to emulation memory.
- Select the internal 8087 numerics coprocessor.

Chapter 5 "Using the Emulator." This chapter describes emulation topics that are not covered in the "Getting Started" chapter (for example, coordinated measurements and storing memory).

Appendix A "Foreground Monitor Description." This appendix describes the foreground monitor program. The foreground monitor is resident in the emulator firmware, but it also comes with the emulator on a floppy disk so that you may customize it, if necessary.

Appendix B "File Format Readers." This appendix shows you how to use files stored in Intel OMF or HP64000 formats.

Index The index allows you to locate topics quickly.

Notes

Contents

3 In-Circuit Emulation

4 Configuring the 8086/88 Emulator

Illustrations

Tables

1

Introduction to the 8086/8088 Emulator

Purpose of the Emulator

The HP 64762/3 8086/8088 emulator replaces the 8086/8088 microprocessor in your target system to help you integrate target system software and hardware. The emulator performs just like the processor that it replaces, but simultaneously gives information about the processor's operation. The emulator gives you control over target system execution. You also can view or modify the contents of processor registers, target system memory, and I/O resources.

Features of the 8086/8088 Emulator

This section introduces you to the features of the emulator.

Supported Microprocessors

The emulator probe has a 40-pin DIP connector. The HP 64762/3 emulators support Intel 8086/8088 microprocessors and other processors that conform to the specifications of the 8086/8088.

Internal 8087 Coprocessor

The HP 64762/3 emulators contain an 8087 numeric data processor. You can enable the internal 8087 with an emulator configuration command. Additional configuration items allow you to:

- Select which RQ/GT pin the internal 8087 will use (if enabled).
- Select the internal 8087 as the 8086/88 INTR input driver.
- Specify the internal interrupt vector (if the internal 8087 drives the 8086/88 INTR input).

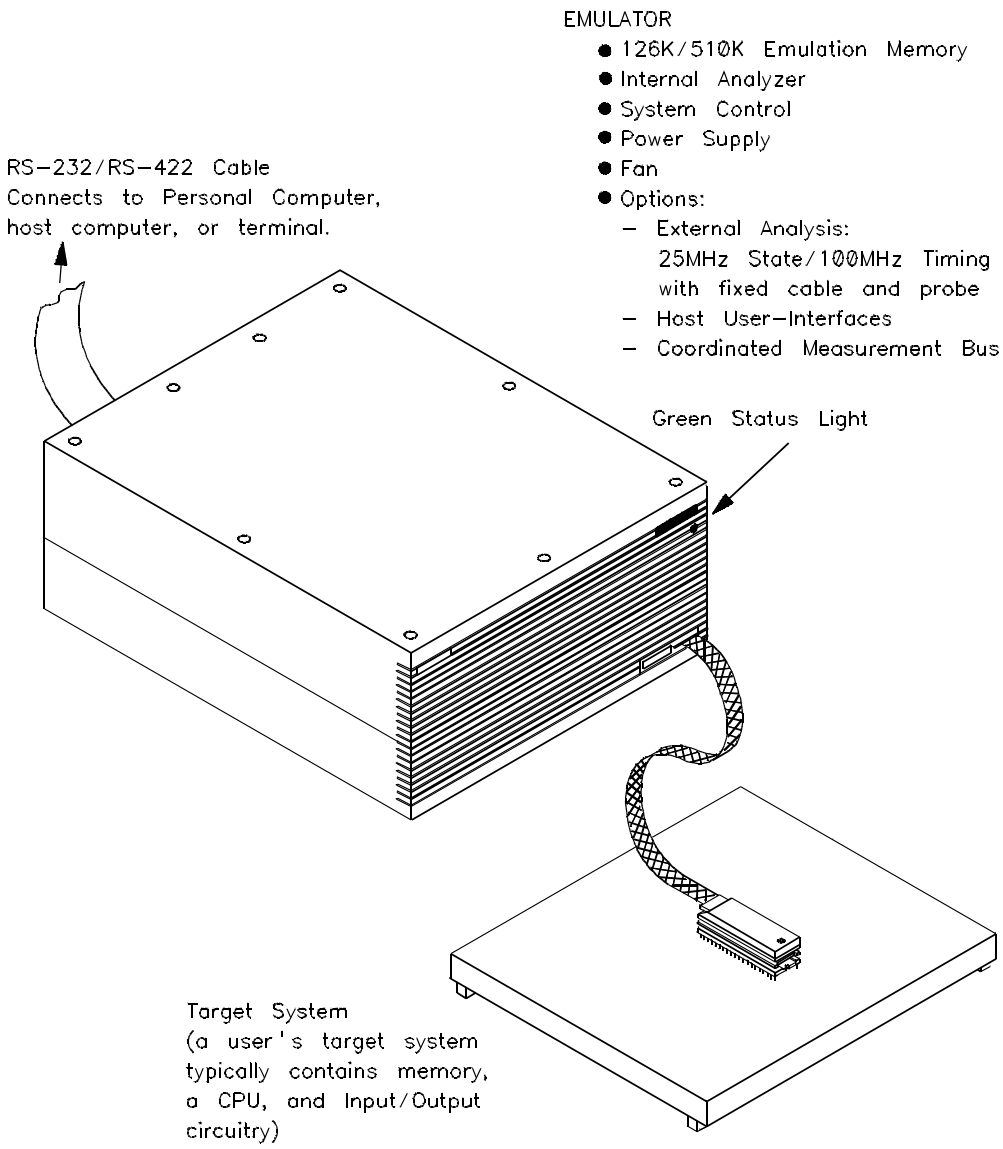


Figure 1-1. The HP 64762/3 Emulator for the 8086/8088

1-2 Introduction

Internal or External Clock Sources

The emulator runs with an internal clock speed of 8 MHz, or with target system clocks from 2-10 MHz.

Emulation Memory

There are either 126K or 510K bytes of emulation memory, depending on the emulator model you have. You can define up to 16 memory ranges (beginning on 1K byte boundaries and at least 1K bytes in length). You can characterize memory ranges as emulation RAM or ROM, target system RAM or ROM, or as guarded memory. The emulator displays an error message for accesses to guarded memory locations. You can configure the emulator so that writes to memory defined as ROM break emulator execution from the user program into the emulation monitor program.

External DMA Access to Emulation Memory

You can enable DMA accesses of emulation memory with an emulator configuration command. Target system devices that reside on the local 8086/8088 bus and conform to the 808X MAX mode bus timing (for example, an external 8087) can access emulation memory.

Analysis

The analyzer supplied with the emulator, called the *emulation analyzer*, captures emulator bus cycle states synchronously with the emulation clock.

The optional *external analyzer* allows you to capture data on up to 16 signals external to the emulator. You can configure the external analyzer to make state or timing analysis measurements.

Refer to the *Analyzer PC Interface User's Guide* for a complete list of analyzer features.

Register Display and Modification

You can display or modify the 8086/88 internal register contents, and you can display the contents of the 8087 numeric coprocessor registers. The 8087 register display shows the register stack registers in both hexadecimal and scientific decimal notation.

Single-Step

You can direct the emulation processor to execute a single instruction or many instructions.

Breakpoints

You can set up the emulator/analyzer interaction so that when the analyzer finds a specific state, emulator execution will break out of the user program into the background monitor.

You also can define software breakpoints in your program. The emulator uses the 8086/88 single-byte interrupt facility to provide software breakpoints. When you define a software breakpoint, the emulator places an INT 3 instruction at the specified address. After the INT 3 instruction breaks emulator execution from the user program (into the monitor), the emulator replaces the original opcode.

Reset Support

The emulator can be reset from the emulation system under your control, or your target system can reset the emulation processor.

Configurable Target System Interface

You can configure the emulator to honor target system wait requests when accessing emulation memory. You can configure the emulator so that it presents cycles to, or hides cycles from, the target system when executing in background. You can configure the emulator so that it presents either a FLUSH or a NOP queue status to the target system while in background (and in the maximum mode). You also can configure the emulator to allow external DMA access to emulation memory.

Foreground or Background Emulation Monitor

The emulation monitor is a program executed by the emulation processor. It allows the emulation controller to access target system resources. For example, when you display target system memory, the monitor program executes 8086 instructions to read the target memory locations and send their contents to the emulation controller.

The monitor program can execute in *foreground*, the mode in which the emulator operates as would the target processor. The foreground monitor occupies processor address space and executes as if it were part of the target program.

The monitor program also can execute in *background*, the emulator mode in which foreground operation is suspended so that the emulation processor can be used to access target system resources. The background monitor does not occupy processor address space.

Real-Time Execution

Real-time operation signifies continuous execution of your program without interference from the emulator. (Such interference occurs when

the emulator temporarily breaks into the monitor so that it can access register contents or target system memory or I/O.)

You can restrict the emulator to real-time execution. When the emulator is executing your program under real-time restriction, commands that display/modify registers, or display/modify target system memory or I/O are not allowed.

Notes

2

Getting Started

Introduction

This chapter leads you through a basic tutorial that shows how to use the HP 64762/64763 emulators with the PC Interface.

This chapter:

- Tells you what to do before you can use the emulator in the tutorial examples.
- Describes the sample program used for this chapter's examples.
- Briefly describes how to enter PC Interface commands and how emulator status is displayed.

This chapter shows you how to:

- Start the PC Interface from the MS-DOS prompt.
- Define (map) emulation and target system memory.
- Load programs into emulation and target system memory.
- Enter emulation commands to view execution of the sample program.

Before You Begin

Prerequisites

Before beginning the tutorial presented in this chapter, you must complete the following tasks:

1. Connect the emulator to your computer. The *Hardware Installation and Configuration* manual shows you how to do this.

2. Install the PC Interface software on your computer. Software installation instructions come with the media containing the PC Interface software. The *PC Interface Reference* manual contains additional information on the installation and setup of the PC Interface. If you need more information, refer to the *MS-DOS Applications Installation* manual.

3. You also should read and understand the concepts of emulation presented in the *System Overview* manual. The *System Overview* also covers HP 64700-Series system architecture. Understanding these concepts may help you avoid problems later.

   You should read the *PC Interface Reference* manual to learn general operation of the PC Interface.

A Look at the Sample Program

Figure 2-1 lists the sample program used in this chapter. The program is a primitive command interpreter.

Data Declarations

The area ORGed at 500H defines the messages used by the program to respond to various command inputs. These messages are labeled **Msg_A**, **Msg_B**, and **Msg_I**.

Initialization

The program instructions from the **Init** label to the **Read_Cmd** label perform initialization. The segment registers are loaded and the stack pointer is set up.

```
HEWLETT-PACKARD: 8086 Assembler
FILE: C:\DIR86\CMD_RDR.S
LOCATION OBJECT CODE LINE     SOURCE LINE

                      1 "8086"
                      2                 GLB     Msgs,Init,Cmd_Input,Msg_Dest
                      3                 ORG     500H
    0500              4 Msgs
    0500 436F6D6D61   5 Msg_A           DB      "Command A entered "
    0505 6E64204120
    050A 656E746572
    050F 656420
    0512 456E746572   6 Msg_B           DB      "Entered B command "
    0517 6564204220
    051C 636F6D6D61
    0521 6E6420
    0524 496E76616C   7 Msg_I           DB      "Invalid Command "
    0529 696420436F
    052E 6D6D616E64
    0533 20
    0534              8 End_Msgs
                      9
                     10
                     11
                     12
                     13                 ORG     400H
                     14                 ASSUME  DS:ORG,ES:ORG
                     15 ************************************************
                     16 * The following instructions initialize segment
                     17 * registers and set up the stack pointer.
                     18 ************************************************
    0400 B80000      19 Init            MOV     AX,SEG Msg_A
    0403 8ED8        20                 MOV     DS,AX
    0405 B80000      21                 MOV     AX,SEG Cmd_Input
    0408 8EC0        22                 MOV     ES,AX
    040A 8ED0        23                 MOV     SS,AX
    040C BCF906      24                 MOV     SP,OFFSET Stk
                     25 ************************************************
                     26 * Clear previous command.
                     27 ************************************************
    040F 26C6060006  28 Read_Cmd        MOV     Cmd_Input,#0
    0414 0090
                     29 ************************************************
                     30 * Read command input byte.  If no command has been
                     31 * entered, continue to scan for command input.
                     32 ************************************************
    0416 26A00006    33 Scan            MOV     AL,Cmd_Input
    041A 3C00        34                 CMP     AL,#0
    041C 74F8        35                 JE      Scan
                     36 ************************************************
                     37 * A command has been entered.  Check if it is
                     38 * command A, command B, or invalid.
                     39 ************************************************
    041E 3C41        40 Exe_Cmd         CMP     AL,#41H
    0420 7407        41                 JE      Cmd_A
    0422 3C42        42                 CMP     AL,#42H
```

Figure 2-1. Sample Program Listing

```
0424 740C          43                JE      Cmd_B
0426 E91200        44                JMP     Cmd_I
                   45 ************************************************
                   46 * Command A is entered.  CX = the number of bytes in
                   47 * message A.  SI = location of the message.  Jump to
                   48 * the routine which writes the messages.
                   49 ************************************************
0429 B91200        50 Cmd_A           MOV     CX,#Msg_B-Msg_A
042C BE0005        51                MOV     SI,OFFSET Msg_A
042F E90F00        52                JMP     Write_Msg
                   53 ************************************************
                   54 * Command B is entered.
                   55 ************************************************
0432 B91200        56 Cmd_B           MOV     CX,#Msg_I-Msg_B
0435 BE1205        57                MOV     SI,OFFSET Msg_B
0438 E90600        58                JMP     Write_Msg
                   59 ************************************************
                   60 * An invalid command is entered.
                   61 ************************************************
043B B91000        62 Cmd_I           MOV    CX,#End_Msgs-Msg_I
043E BE2405        63                MOV     SI,OFFSET Msg_I
                   64 ************************************************
                   65 * Message is written to the destination.
                   66 ************************************************
0441 8D3E0106      67 Write_Msg       LEA     DI,Msg_Dest
0445 F3A4          68             REP MOVSB
                   69 ************************************************
                   70 * The rest of the destination area is filled
                   71 * with zeros.
                   72 ************************************************
0447 C60500        73 Fill_Dest       MOV     BYTE PTR [DI],#0
044A 47            74                INC     DI
044B 81FF2106      75                CMP    DI,#Msg_Dest+20H
044F 75F6          76                JNE     Fill_Dest
                   77 ************************************************
                   78 * Go back and scan for next command.
                   79 ************************************************
0451 EBBC          80                JMP     Read_Cmd
                   81
                   82                ORG     600H
                   83 ************************************************
                   84 * Command input byte.
                   85 ************************************************
0600               86 Cmd_Input       DBS     1
                   87 ************************************************
                   88 * Destination of the command messages.
                   89 ************************************************
0601               90 Msg_Dest        DDS     3EH
06F9               91 Stk             DWS     1       ; Stack area.
                   92                END     Init

Errors=    0
```

Figure 2-1. Sample Program Listing (Cont'd)

Reading Input

The instruction at the **Read_Cmd** label clears any random data or previous commands from the **Cmd_Input** byte. The **Scan** loop continually reads the **Cmd_Input** byte to look for a command (a value other than 0H).

Processing Commands

When a command is entered, the instructions from **Exe_Cmd** to **Cmd_A** determine whether the command was "A," "B," or an invalid command.

If the command input byte is "A" (ASCII 41H), execution transfers to the instructions at **Cmd_A**.

If the command input byte is "B" (ASCII 42H), execution transfers to the instructions at **Cmd_B**.

If the command input byte is neither "A" or "B," an invalid command was entered, and execution transfers to the instructions at **Cmd_I**.

The instructions at **Cmd_A**, **Cmd_B**, and **Cmd_I** each load register CX with the length of the message to be displayed, and register SI with the message's starting location. Then, execution transfers to **Write_Msg**, which writes the appropriate message to the destination location, **Msg_Dest**.

After the message is written, the instructions at **Fill_Dest** fill the remaining destination locations with zeros. (The entire destination area is 20H bytes long.) Then, the program jumps back to read the next command.

The Destination Area

The area ORGed at 600H declares memory storage for the command input byte, the destination area, and the stack area.

Sample Program Location

The sample program is installed in the directory \hp64700\demo\64762. You can copy the files cmd_rdr.s and cmd_rdr.k to another directory from the demo directory if you wish. A batch file (make.bat) is provided to automate assembly and linking of

the file, or you can enter the commands listed in the following paragraphs.

Assembling the Sample Program

The sample program is written for and assembled with the HP 64853 8086/88 Series Cross Assembler/Linker. The following command was used to assemble the sample program.

```
C>asm -o cmd_rdr.s > cmd_rdr.o
<RETURN>
```

The assembler creates the assembler listing (cmd_rdr.o) and two other files. The "cmd_rdr.r" file is the relocatable file. Relocatable files are linked together to form the absolute file, which is loaded into the emulator. The "cmd_rdr.a" file is the assembler symbol file. It contains information on the local symbols in the sample program.

Linking the Sample Program

The linker command file (cmd_rdr.k) shown in figure 2-2 and the following linker command were used to generate the absolute file that is loaded into the emulator.

```
C>lnk -o cmd_rdr.k > cmd_rdr.map
<RETURN>
```

Three files are created. First is the linker load map listing (cmd_rdr.map). The "cmd_rdr.x" file is the file that contains the absolute code to be loaded into the emulator. The "cmd_rdr.l" file is the linker symbol file. It contains information on the global symbols in the sample program and the relocatable files that were combined to form the absolute file.

```
segment
object files  cmd_rdr.R
library files
load addresses  00000400H 00000500H 00000600H
absolute file name  cmd_rdr.X
```

Figure 2-2. Linker Command File for Sample Program

Starting the 8086 PC Interface

If you have set up the emulator device table and the **HPTABLES** shell environment variable as shown in the *PC Interface Reference*, you can start up the 8086 PC Interface. Enter the following command at the MS-DOS prompt:

```
C> pci808x <emulname>
```

In the command above, **pci808x** is the command to start the PC Interface; "<emulname>" is the logical emulator name given in the emulator device table. If this command is successful, you will see the display shown in figure 2-3. Otherwise, you will see an error message and return to the MS-DOS prompt.

Selecting PC Interface Commands

You can select command options by either using the left and right arrow keys to highlight the option. Then, press the **Enter** key. Or, you can simply type the first letter of that option. If you select the wrong option, you can press the **ESC** key to retrace the command tree.

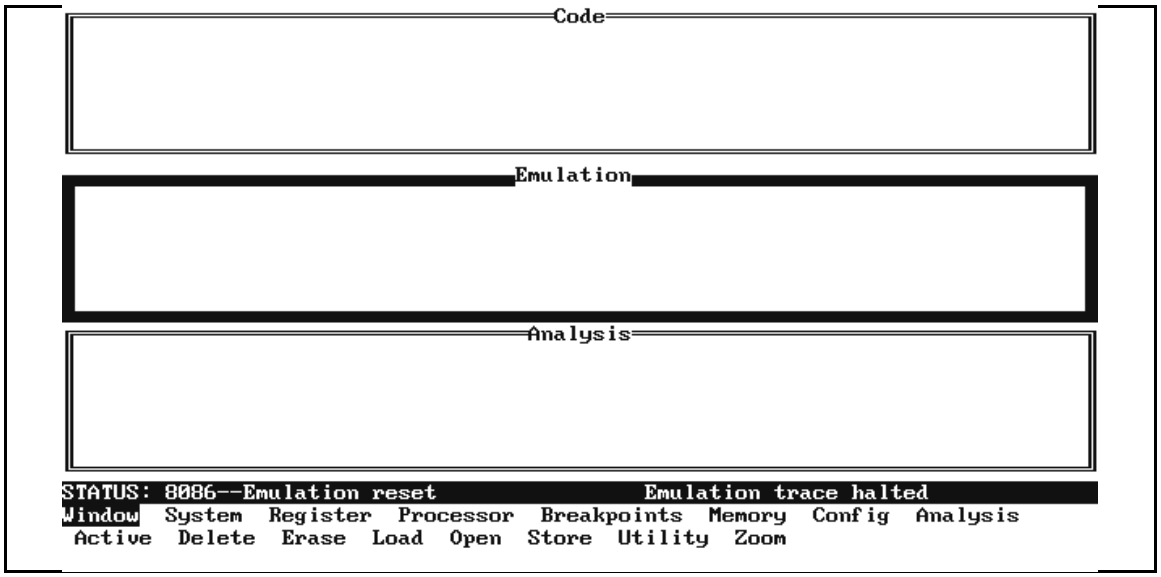


Figure 2-3. PC Interface Display

When a command or option is highlighted, the bottom line of the display shows the next level of options or a short message describing the current option.

Emulator Status

The line above the command options displays the emulator's status. The PC Interface periodically checks the status and updates the status line.

Mapping Memory

Depending on the emulator model number, emulation memory has either 128K or 512K bytes, mappable in 1K byte blocks. The monitor occupies 2K bytes, leaving 126K or 510K bytes of emulation memory for your programs. The emulation memory system does not need wait states.

The memory mapper allows you to characterize memory locations. You can specify that a certain range of memory is present in the target system, or that you will be using emulation memory for that address range. You also can specify whether the target system memory is ROM or RAM, and you can specify that emulation memory be treated as ROM or RAM.

Blocks of memory also can be characterized as guarded memory. Guarded memory accesses will generate "break to monitor" requests. Writes to ROM will generate "break to monitor" requests if the "Enable breaks on writes to ROM?" configuration item is enabled (see the "Configuring the Emulator" chapter).

Which Memory Locations Should Be Mapped?

Typically, assemblers generate relocatable files and linkers combine relocatable files to form the absolute file. The linker load map listing will show what locations your program will occupy in memory. Figure 2-4 shows a linker load map listing for the sample program.

```
HP 64000+ Linker                                        Fri Aug 17 09:34:03 1990
Page   1

FILE/PROG NAME               PROGRAM    DATA       COMMON     ABSOLUTE
-------------------------------------------------------------------------
cmd_rdr.R
    C:\HP64700\86\CMD_RDR.S
                                                                00000500-00000533
                                                                00000400-00000452
                                                                00000600-000006FA

next address

XFER address = 00000400  Defined by cmd_rdr.R
Current working directory = C:\HP64700\86
Absolute file name = cmd_rdr.X
Total number of bytes loaded = 00000182
```

Figure 2-4. Sample Program Load Map Listing

From the load map listing, you can see that the sample program occupies locations in four address ranges. The program area, which contains the opcodes and operands of the sample program, occupies locations 400H through 452H. The data area, which contains the ASCII values of the messages the program displays, occupies locations 500H through 533H. The destination area, which contains the command input byte and the locations of the message destination and the stack, occupies locations 600H through 6FAH.

One mapper term must be specified for the example program. Since the program writes to the destination locations, the mapper block containing the destination locations should not be mapped as ROM.

To map memory for the sample program, select:

```
Config Map Modify
```

Using the arrow keys, move the cursor to the “address range” field of term 1. Enter:

```
400..7ff
```

Move the cursor to the “memory type” field of term 1, and press the **Tab** key to select the **eram** (emulation RAM) type.

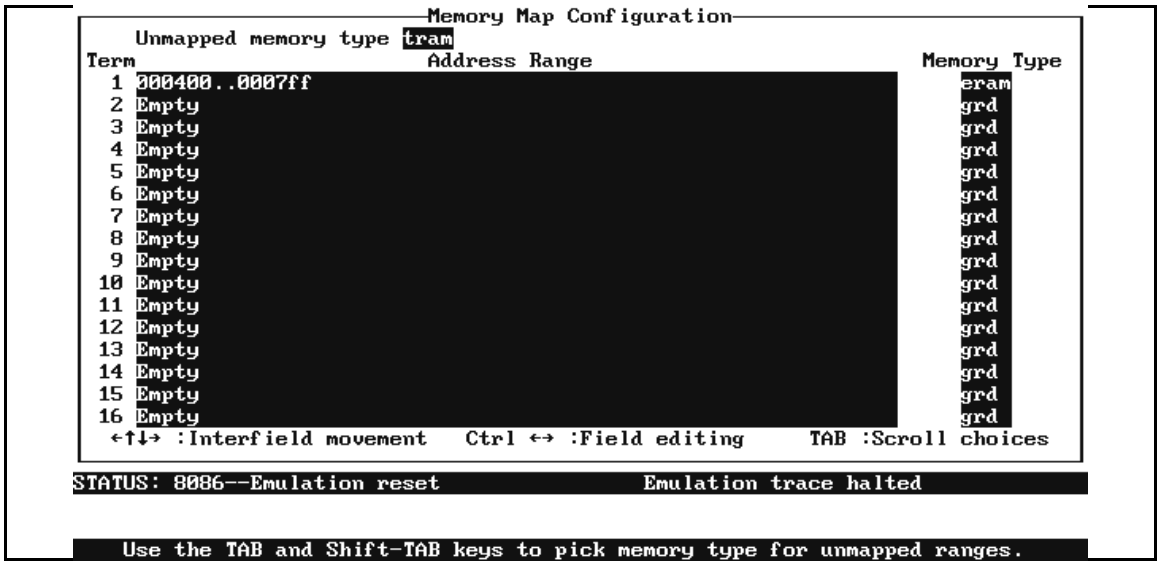


Figure 2-5. Memory Map Configuration

To save your memory map, press the **End** key to move to the last field. Then use the **Enter** key to exit the field. Figure 2-5 shows the memory configuration display.

When mapping memory for your target system programs, you may wish to map emulation memory locations containing programs and constants (locations that should not be written to) to ROM. This will prevent programs and constants from being written over accidentally, and will cause breaks when instructions do so.

Note

The memory mapper reassigns blocks of emulation memory after the insertion or deletion of mapper terms. For example, if you modified the contents of 400..7ff above, added a new term, then deleted term 1, and displayed locations 400..7ff, you would notice that the contents of those locations differ.

Note

Software breakpoints should be removed before altering the memory map. If they are not, SBI instruction (INT 3) opcodes will be left at unknown locations.

Loading Programs into Memory

If you have already assembled and linked the sample program, you can load the absolute file by selecting:

```
Memory Load
```

File Format

Use **Tab** and **Shift-Tab** to select the format of your absolute file. The emulator accepts absolute files in the following formats:

- Intel Object Module Format (OMF86)
- HP absolute
- Intel hexadecimal
- Tektronix hexadecimal
- Motorola S-records

For this tutorial, choose the HP64000 format.

Memory Type

The next field allows you to selectively load the portions of the absolute file which reside in emulation memory, target system memory, or both. Since emulation memory is mapped for sample program locations, you can enter either “emulation” or “both.”

Force Absolute File Read

This option is only available for the HP64000 and Intel OMF formats. It forces the file format readers to regenerate the emulator absolute file (.hpa) and symbol database (.hps) before loading the code. Normally, these files are only regenerated whenever the file you specify (the

output of your language tools) is newer than the emulator absolute file and symbol database.

For more information, refer to the File Format Readers appendix.

File Format Options

Some of the formats, such as the Intel OMF format, have special options. Refer to the File Format Readers appendix of this manual for more information.

Absolute File Name

For most formats, you enter the name of your absolute file in the last field. The HP64000 format requires the linker symbol filename instead. Type **cmd_rdr.l**, and press **Enter** to start the memory load.

Using Symbols

Symbols are available when you load HP or Intel OMF format absolute files.

When you build an absolute file using the HP 64000 development tools, an assembler symbol file (with the same base name as the source file and a ".A" extension) is created. The assembler symbol file contains local symbol information. When you link relocatable assembly modules, a linker symbol file (with the same base name as the absolute file and a ".L" extension) is created. The linker symbol file contains global symbol information and information about the relocatable assembly modules that were combined to form the absolute.

Symbols are a part of the Intel OMF definition. They are contained in the absolute file.

When you load a file using the HP64000 file format, the file format reader collects global symbol information from the linker symbol file and local symbol information from the assembler symbol files. It uses this information to create a single symbol database with the extension .hps. If you load an Intel OMF file, the file format reader obtains all the global and local symbol information from the absolute file and builds a symbol database with the extension .hps. For both formats, the reader builds an absolute file with extension .hpa that is optimized for efficient transfer to the emulator.

The 8086/88 emulator PC Interface provides two methods for manipulating symbols. It allows you to use both global and local symbols to reference memory variables within the PC Interface.

You also can transfer symbols to the emulator for use in displaying memory values with the "**M**emory **D**isplay **M**nemonic" command, the "**A**nalysis **D**isplay" command, and the mnemonic field of single-step commands. (These features are only available if you have the version of emulator firmware that supports emulator symbol-handling.)

Displaying Global Symbols

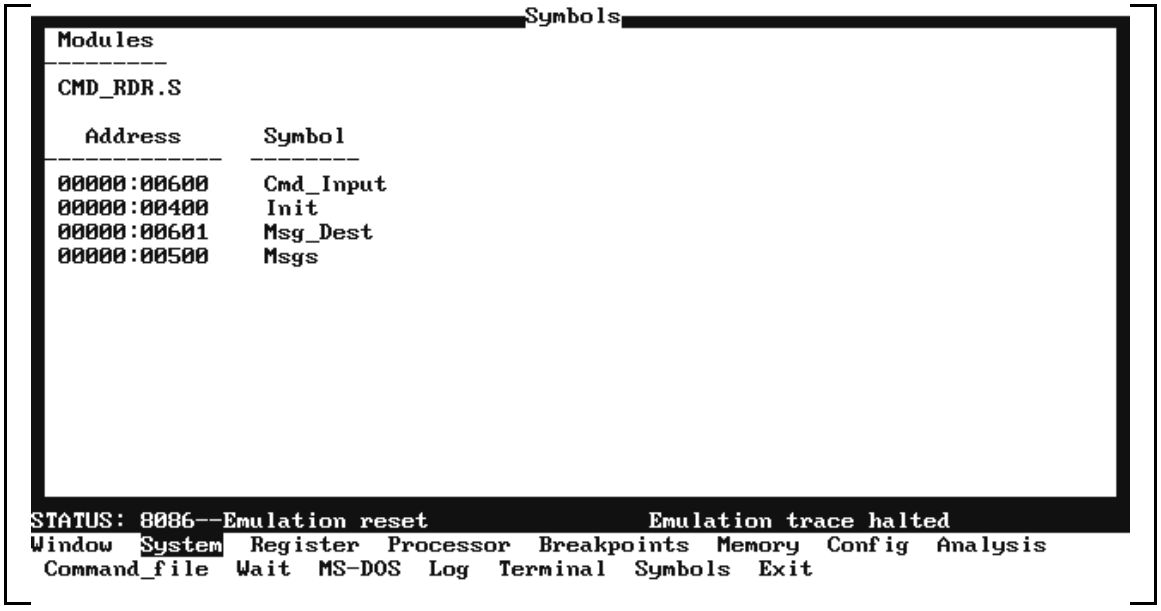
When you load HP format or Intel OMF absolute files into the emulator, the corresponding symbol database is also loaded.

The symbol database also can be loaded with the "**S**ystem **S**ymbols **G**lobal **L**oad" command. Use this command when you load multiple absolute files into the emulator. You can load the various symbol databases corresponding to each absolute file. When you load a symbol database, information from a previous symbol database is lost. That is, only one symbol database can be present at a time.

After a symbol database is loaded, both global and local symbols can be used when entering expressions. You enter global symbols as they appear in the source file or in the global symbols display. To display global symbols, select:

```
System Symbols Global Display
```

The symbols window automatically becomes active. Press <CTRL>**z** to zoom the window. The resulting display follows.



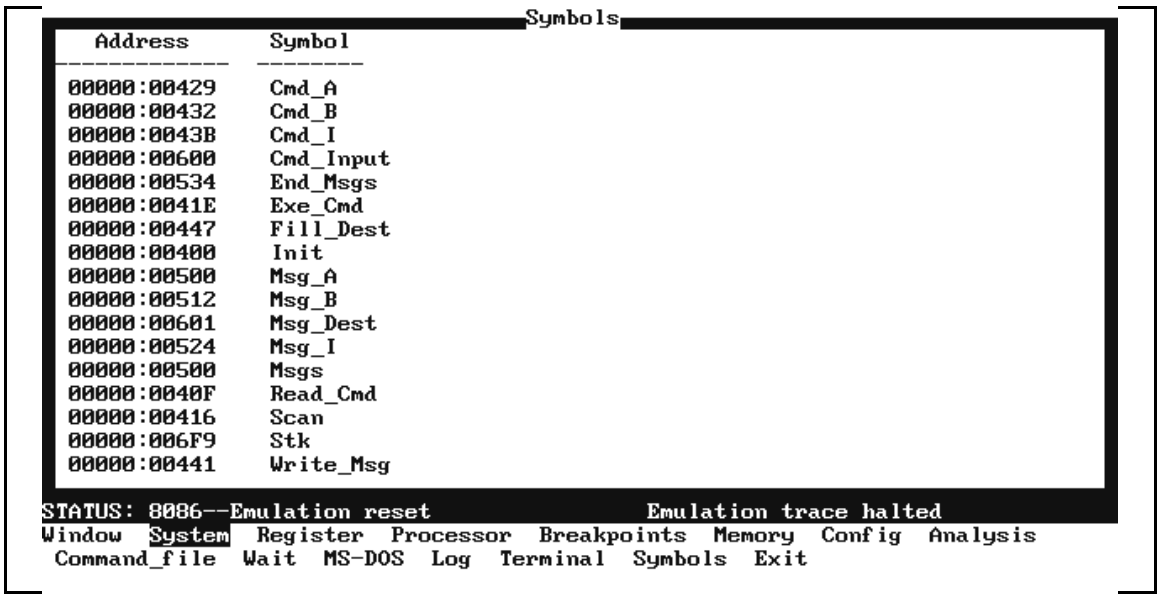
The global symbols display has two parts. The first part lists all the modules that were linked to produce this object file. These module names are used by you when you want to refer to a local symbol, and are case-sensitive. The second part of the display lists all global symbols in this module. These names can be used in measurement specifications, and are case-sensitive. For example, if you want to make a measurement using the symbol **Cmd_Input**, you must specify **Cmd_Input**. The strings **cmd_input** and **CMD_INPUT** are not valid symbol names here.

Load and Display Local Symbols

To load and display local symbols, select:

```
System Symbols Local Display
```

Enter the name of the module you want to display (from the first part of the global symbols list; in this case, **CMD_RDR.S**) and press **Enter**. The resulting display follows.



Note

Emulators with system firmware below version 2.00 do not have symbol handling capability and thus have a restricted symbol command set in the PC Interface. To display local symbols in these emulators, use the command:

```
System Symbols Local
```

After you display local symbols with the "**S**ystem **S**ymbols **L**ocal **D**isplay" command, you can enter local symbols as they appear in the source file or local symbol display. When you display local symbols for a given module, that module becomes the default local symbol module.

Local symbols must be from assembly modules that were linked to form the absolute whose symbol database is currently loaded. Otherwise, no symbols will be found (even if the named assembler symbol file exists and contains information).

Using Local Symbols without Loading and Displaying

If you have not displayed local symbols, you can still enter a local symbol by including the name of the module:

```
module_name:symbol
```

Remember that the only valid module names are those listed in the first part of the global symbols dispaly, and are case-sensitive for compatibility with other systems (such as HP-UX).

When you include the name of a source file with a local symbol, that module becomes the default local symbol module, as with the "**S**ystem **S**ymbols **L**ocal **D**isplay" command.

Transferring Symbols to the Emulator

If your emulator has system firmware version 2.0 or later, yo can use the emulator's symbol-handling capability to improve measurement displays. You do this by transferring the symbol database information to the emulator.

You transfer the global symbols to the emulator with the command:

```
System Symbols Global Transfer
```

Then the global symbol names are displayed instead of the absolute address values in subsequent memory references.

You transfer all local symbols from the absolute file using the command:

```
System Symbols Local Transfer All
```

You can transfer the local symbols from one or more modules using the "**S**ystem **S**ymbols **L**ocal **T**ransfer **G**roup <module_name>" command. Therefore, to transfer the local symbols from an example module named "MODNAME," enter:

```
System Symbols Local Transfer Group
```

Enter "MODNAME," and press **<Enter>.** The symbols are transferred to the emulator.

You can find more information on emulator symbol handling commands in the *Emulator PC Interface Reference*.

To determine which version of emulator firmware is present, use the Terminal Interface **ver** command while in System Terminal Mode. See

the *Emulator PC Interface Reference* for more information on System Terminal mode.

If your emulator does not have symbol handling capability, the symbol transfer commands will not appear in the PC Interface. Also, the displays in this chapter will appear somewhat different. For example, in the memory mnemonic example, the "Symbol" column will be replaced by a "Data" column showing the data found at that address.

Contact your local HP Sales and Service office (listed in the *Support Services* guide) for information on firmware upgrades.

Removing Symbols from the Emulator

If you transfer many symbols to the emulator, you can fill the available memory used to store symbolic references. The PC Interface allows you to remove global and local symbols to free symbol memory in the emulator.

You can delete all global symbols in the emulator with the "**S**ystem **S**ymbols **G**lobal **R**emove" command, or all local symbols with the "**S**ystem **S**ymbols **L**ocal **R**emove **A**ll" command. You can delete local symbols from one or more modules with the "**S**ystem **S**ymbols **L**ocal **R**emove **G**roup <module_name>" command.

The Scope of Symbols

It is possible for a symbol to be local in one module and global in another, which may result in some confusion. For example, suppose symbol "XYZ" is defined as a global in module A and as a local in module B, and that these modules are linked to form the absolute file. After you load the absolute file (and the corresponding symbol database), entering "XYZ" in an expression refers to the symbol from module A. Then, if you display local symbols from module B, entering "XYZ" in an expression refers to the symbol from module B, *not the global symbol*.

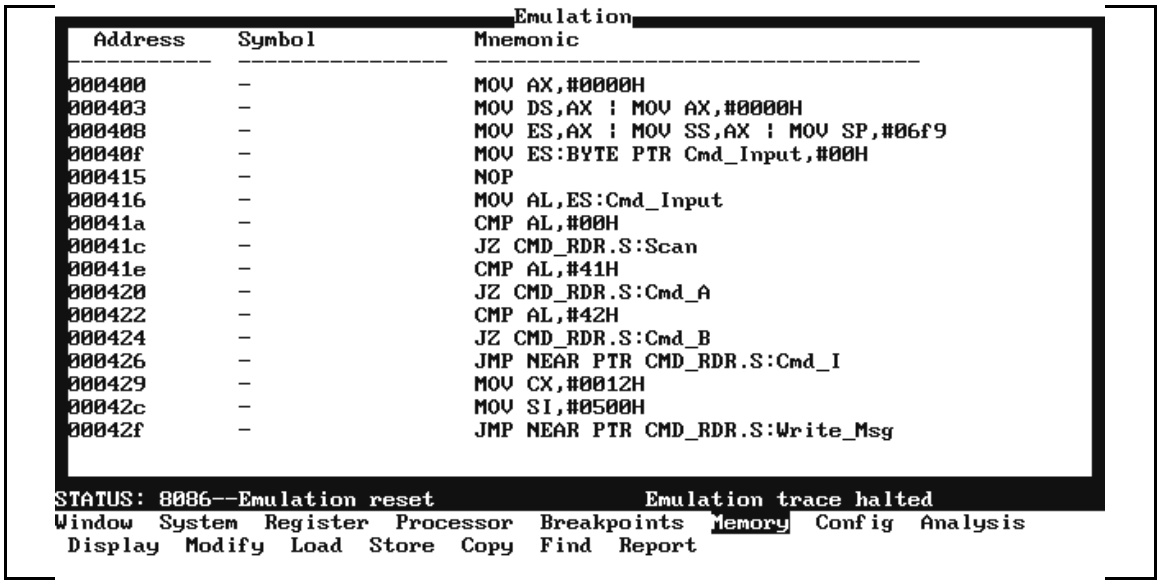
Now, if you want to enter "XYZ" to refer to the global symbol from module A, you must display the local symbols from module A (since the global symbol is also local to that module). Loading local symbols from a third module, if it was linked with modules A and B and did not contain an "XYZ" local symbol, also would cause "XYZ" to refer to the global symbol from module A.

Display Memory in Mnemonic Format

Once you have loaded a program into the emulator, you can verify that the program was loaded by displaying memory in mnemonic format. To do this, select:

```
Memory Display Mnemonic
```

Enter the address range "400..431." The emulation window automatically becomes active. You can press <CTRL>**z** to zoom the memory window. The resulting display follows.

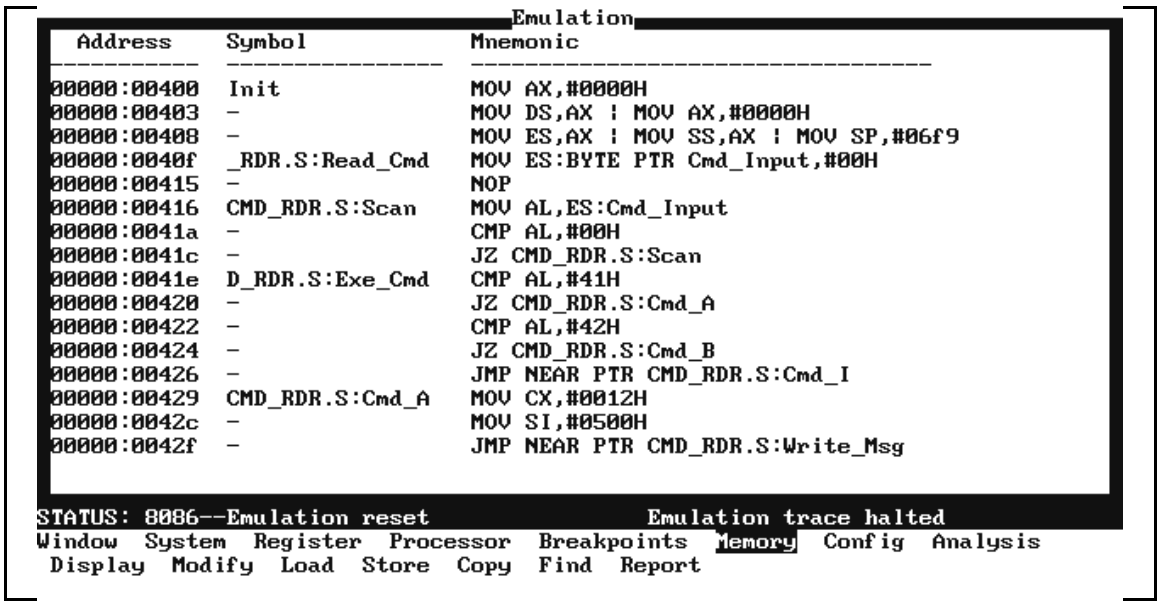


Notice that the symbol column is empty. That is because the symbols are related to segment:offset addresses, not physical addresses. To see the symbols, you can enter an address range using segment:offset addresses or symbol names. For example, enter:

```
Memory Display Mnemonic
```

Enter the address range **Init..Init+31**. The resulting display follows.

If you want to see the rest of the program memory locations, select the "**M**emory **D**isplay **M**nemonic" command and enter the range 0000:0432 to 0000:0452.



When Symbol Handling is Supported

If your emulator supports symbol handling, when you issue a "**M**emory **D**isplay **M**nemonic" command, the memory list will include symbols as shown in the display. (You must first transfer the symbols to the emulator as described earlier in this chapter. You also must enter address information in segment:offset or symbol form, as described earlier.)

If your emulator supports symbols, but no symbols have been transferred to the emulator, only "-" will be displayed in the Symbol column.

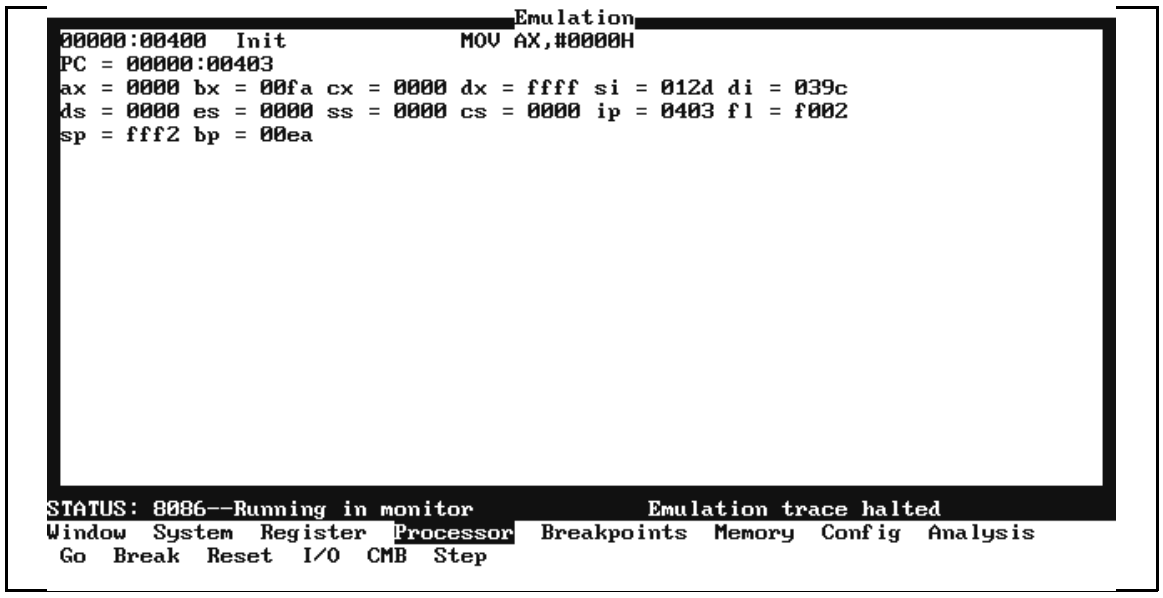
Step Through the Program

The emulator allows you to execute one instruction or several instructions with the step command. To begin stepping through the sample program, select:

Processor **S**tep **A**ddress

Enter a step count of 1, enter the symbol **Init** (defined as a global in the source file), and press **Enter** to step from the program's first address,

400H. The emulation window will automatically become the active window. Press <CTRL>**z** to view a full screen of information. The executed instruction, the program counter address (CS:IP), and the resulting register contents are displayed as shown in the following listing.



Note

You cannot display registers if the processor is reset. Use the "**P**rocessor **B**reak" command to start executing in the monitor. You can display registers while the emulator is executing a user program (if execution is not restricted to real-time). Emulator execution will temporarily break to the monitor.

An individual step may execute multiple instructions. When this happens, the step display includes all those instructions.

To continue stepping through the program, you can select:

Processor **S**tep **P**c

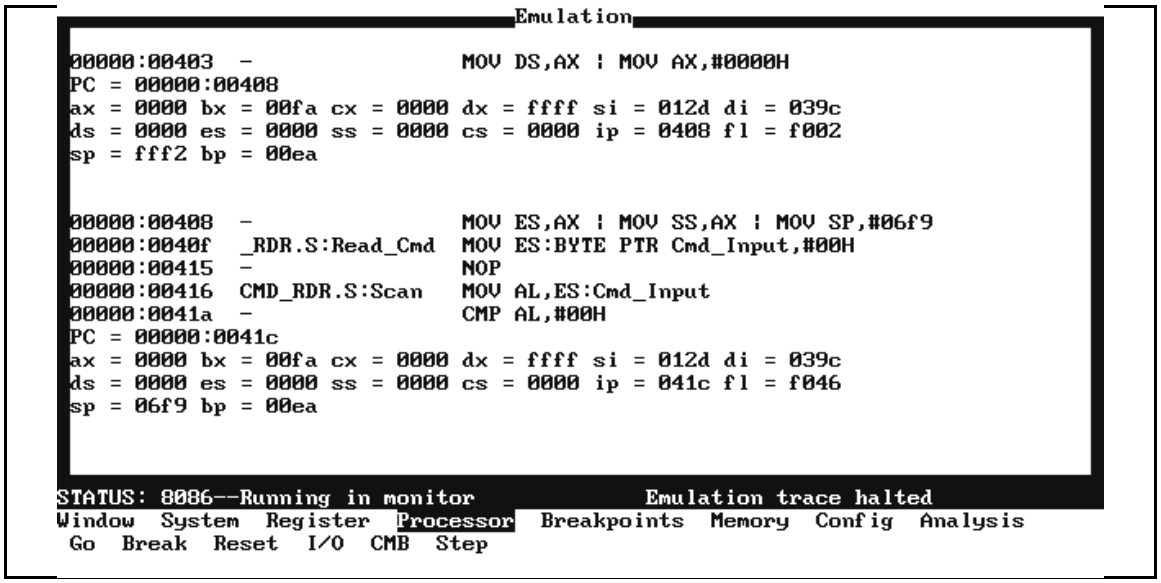
After selecting the command above, you have the opportunity to change the previous step count. If you wish to step the same number of times, you can press **Enter** to start the step. To repeat the previous command, you can press <CTRL>**r**.

Specifying a Step Count

If you want to continue to step a number of times from the current program counter, select:

Processor **S**tep **P**c

The previous step count is displayed in the "number of instructions" field. You can enter a number from 1 through 99 to specify the number of times to step. Type 5 into the field, and press **Enter**. The resulting display follows.



When you specify step counts greater than one, only the last instruction and register contents after the instruction are displayed.

Modify Memory

The preceding step commands show the sample program is executing in the **Scan** loop, where it continually reads the command input byte to check if a command has been entered. To simulate the entry of a sample program command, you can modify the command input byte by selecting:

```
Memory Modify Bytes
```

Now, enter the address of the memory location to be modified, an equal sign, and new value of that location, for example, **Cmd_Input=41H** or **Cmd_Input="A"**. (The **Cmd_Input** label was defined as a global symbol in the source file.)

To verify that 41H was written to **Cmd_Input** (600H), select:

```
Memory Display Bytes
```

Type the address 600 or the symbol **Cmd_Input**, and press **Enter**. This command automatically activates the emulation window. The resulting display is as follows:

```
                                  Symbols
  Address       Symbol
-------------  ---------
 00000:00429    Cmd_A
 00000:00432    Cmd_B
 00000:0043B    Cmd_I

                                 Emulation
  Address     Data                                                Ascii
-----------  ------------------------------------------------  ----------------
00000:00600  41                                                A


                                 Analysis


STATUS: 8086--Running in monitor              Emulation trace halted
Window  System  Register  Processor  Breakpoints  Memory  Config  Analysis
 Display  Modify  Load  Store  Copy  Find  Report
```

Run the Program

To start the emulator executing the sample program, select:

```
Processor Go Pc
```

The status line will show that the emulator is “Running user program.”

Search Memory for Data

You can search the message destination locations to verify that the sample program writes the appropriate messages for the allowed commands. The command “A” (41H) was entered above, so the “Command A entered “ message should have been written to the **Msg_Dest** locations. To search the destination memory location for the command message, select:

```
Memory Find
```

Enter the range of the memory locations to be searched, 600 through 620, and enter the data **"Command A entered"**. The resulting information in the memory window shows you that the message was written correctly.

To verify that the sample program works for the other allowed commands, you can modify the command input byte to “B” and search for **"Entered B command"**. Or, you can modify the command input byte to “C” and search for **"Invalid command"**.

Break Into the Monitor

To break emulator execution from the sample program to the monitor program, select:

```
Processor Break
```

The status line shows that the emulator is "Running in monitor."

While the break will occur when possible, the actual stopping point may be many cycles after the break request (dependent on the type of instruction being executed and whether the processor is in a hold state).

Using Software Breakpoints

Software breakpoints are handled by the 8086/88 single-byte interrupt (SBI) facility. When you define or enable a software breakpoint, the emulator will replace the opcode at the software breakpoint address with a breakpoint interrupt instruction (INT 3).

Note

In order for the software breakpoints feature to work, a stack must be set up in the user program.

Note

You must only set software breakpoints at memory locations that contain instruction opcodes (not operands or data). If you set a software breakpoint at a memory location that is not an instruction opcode, the software breakpoint instruction will never execute and the break will never occur.

Note

Because software breakpoints are implemented by replacing opcodes with the single-byte interrupt instructions, you cannot define software breakpoints in target ROM.

Note

Do not add, set, remove, or disable software breakpoints while the emulator is running user code. If you enter any of these commands while the emulator is running user code in the area where the breakpoint is being modified, program execution may be unreliable.

Note

Remove software breakpoints before altering the memory map or changing the monitor type. If you do not remove the breakpoints, SBI instruction (INT 3) opcodes will be left at unknown locations.

When the emulator detects a vector fetch from the single-byte interrupt area (in other words, the INT 3 instruction has executed), it generates a break to background request, which causes an NMI response, as with the "**P**rocessor **B**reak" command. Since the system controller knows the locations of defined software breakpoints, it can determine whether the SBI was an enabled software breakpoint or a single-byte interrupt instruction in your target program.

If the SBI was generated by a software breakpoint, execution breaks to the monitor, and the breakpoint interrupt instruction (INT 3) is replaced by the original opcode. A subsequent run or step command will execute from this address.

If the SBI was generated by a single-byte interrupt instruction in the target system, execution still breaks to the monitor. An "undefined breakpoint" status message is displayed. To continue with program execution, you must run or step from the target program's breakpoint interrupt vector address.

Defining a Software Breakpoint

To define a breakpoint at the address of the **Cmd_I** label of the sample program (43BH), select:

```
Breakpoints Add
```

Enter the local symbol "Cmd_I." After the breakpoint is added, the emulation window becomes active and shows that the breakpoint is set.

You can add multiple breakpoints in a single command by separating them with a semicolon. For example, you could type **2010h;2018h;2052h** to set three breakpoints.

Run the program by selecting:

```
Processor Go Pc
```

The status line shows that the emulator is running the user program. Modify the command input byte to an invalid command by selecting:

```
Memory Modify Bytes
```

Enter an invalid command, such as "Cmd_Input=75h." The following messages result:

```
ALERT:  Software breakpoint:
00000:0043b
STATUS: Running in monitor
```

To continue program execution, select:

```
Processor Go Pc
```

Displaying Software Breakpoints

To view the status of the breakpoint, select:

```
Breakpoints Display
```

The resulting display shows that the breakpoint was cleared.

Setting a Software Breakpoint

When a breakpoint is hit, it is disabled. To reenable the software breakpoint, you can select:

```
Breakpoints Set Single
```

As with the "**B**reakpoints **A**dd" command, the breakpoint window becomes active and shows that the breakpoint is set.

Clearing a Software Breakpoint

If you wish to clear a software breakpoint that does not get hit during program execution, you can select:

```
Breakpoints Clear Single
```

Using the Analyzer

The 8086/8088 emulation analyzer has 47 trace signals, which monitor internal emulation lines (address, data, and status lines). Optionally, you may have an additional 16 trace signals, which monitor external input lines. The analyzer collects data at each pulse of a clock signal, and saves the data (a trace state) if it meets a "storage qualification" condition.

Note

Emulators which do not have the optional external analyzer will not display the Internal option shown in the examples. Enter the first part of the command to execute the example.

Resetting the Analysis Specification

To be sure that the analyzer is in its default or power-up state, select:

```
Analysis Trace Reset Internal
```

Specifying a Simple Trigger

Suppose you want to trace the states of the sample program that follow the read of a "B" (42H) command from the command input byte. To do this, you must modify the default analysis specification by selecting:

```
Analysis Trace Modify Internal
```

The analyzer trigger specification is shown. Use the arrow keys to move to the "Trigger on" field. Type in "a" and press **Enter**.

Now you enter the analzyer pattern specification form. Use the arrow keys to move the cursor to the filed under the heading “addr” next to the row marked “a=.” Type in the address of the command input byte, using either the global symbol **Cmd_Input** or address 0000:0600, and press **Enter**.

The “Data” field is now highlighted. Type in 0XX42 and press **Enter**. 42H is the value of the “B” command and the “X"s specify “don’t care” values. When 42H is read from the command input byte (600H), a lower byte read is performed because the address is even.

Now the “Status” field is highlighted. Use the **Tab** key to view the status qualifiers that may be entered.

The status qualifiers are defined as follows.

Qualifier	**Status Bits (46..36)**	**Description**
exec	0xx xxxx xxxxB	Executed instruction state
procopf	1xx xx01 x100B	Processor opcode fetch cycle
procmr	1xx xx01 x101B	Processor memory read cycle
procmw	1xx xx01 x110B	Processor memory write cycle
procior	1xx xx01 x001B	Processor I/O read cycle
prociow	1xx xx01 x010B	Processor I/O write cycle
dmaior	1xx xxx0 x001B	DMA I/O read cycle
dmaiow	1xx xxx0 x010B	DMA I/O write cycle
dmamr	1xx xxx0 x101B	DMA memory read cycle
dmamw	1xx xxx0 x110B	DMA memory write cycle
procinta	1xx xx01 x000B	Processor interrupt acknowledge cycle
prochalt	1xx xx01 x011B	Processor halt acknowledge cycle
opcode	1xx xxxx x100B	Opcode fetch
memread	1xx xxxx x101B	Memory read cycle

Qualifier	Status Bits (46..36)	Description
memwrite	1xx xxxx x110B	Memory write cycle
ioread	1xx xxxx x001B	I/O port read cycle
iowrite	1xx xxxx x010B	I/O port write cycle
proc	1xx xx01 xxxxB	Processor (not DMA) cycle
dma	1xx xxx0 xxxxB	DMA cycle
coproc	1xx xx11 xxxxB	Coprocessor cycle
intack	1xx xxxx x000B	Interrupt acknowledge cycle
halt	1xx xxxx x011B	Halt acknowledge cycle
grd	1xx x1xx xxxxB	Guarded memory access
rom	1xx 1xxx xxxxB	Access to ROM cycle
procr	1xx xx01 xx01B	Processor read cycle
procw	1xx xx01 xx10B	Processor write cycle

Select the **memread** status and press **Enter**.

Figure 2-6 shows the resulting analysis pattern specification. To save the new specification, use the **End Enter** key sequence to exit from the form.

Figure 2-7 shows the analyzer trigger specification. To save the trigger specification, press **End**, then **Enter**.

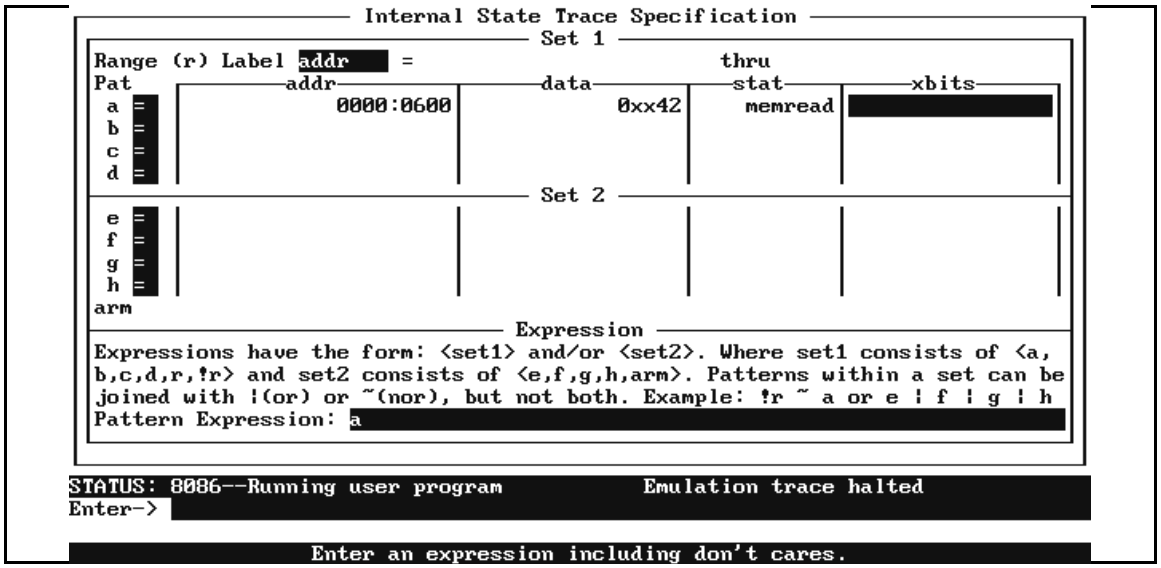


Figure 2-6. Analyzer Pattern Specification

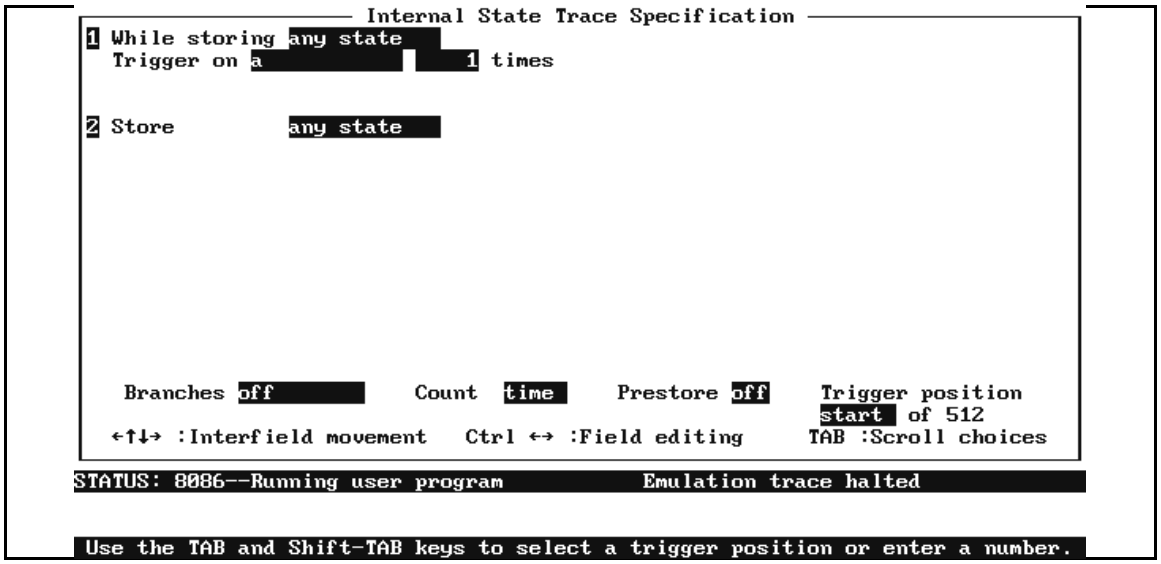


Figure 2-7. Analyzer Trigger Specification

Starting the Trace

To start the trace, select:

```
Analysis Begin Internal
```

A message on the status line shows that the trace is running. You do not expect the trigger to be found because no commands were entered. Modify the command input byte to "B" by selecting:

```
Memory Modify Bytes
```

Enter **Cmd_Input=42** or **Cmd_Input="B"**. The status line now shows that the trace is complete.

Displaying the Trace

To display the trace, select:

```
Analysis Display
```

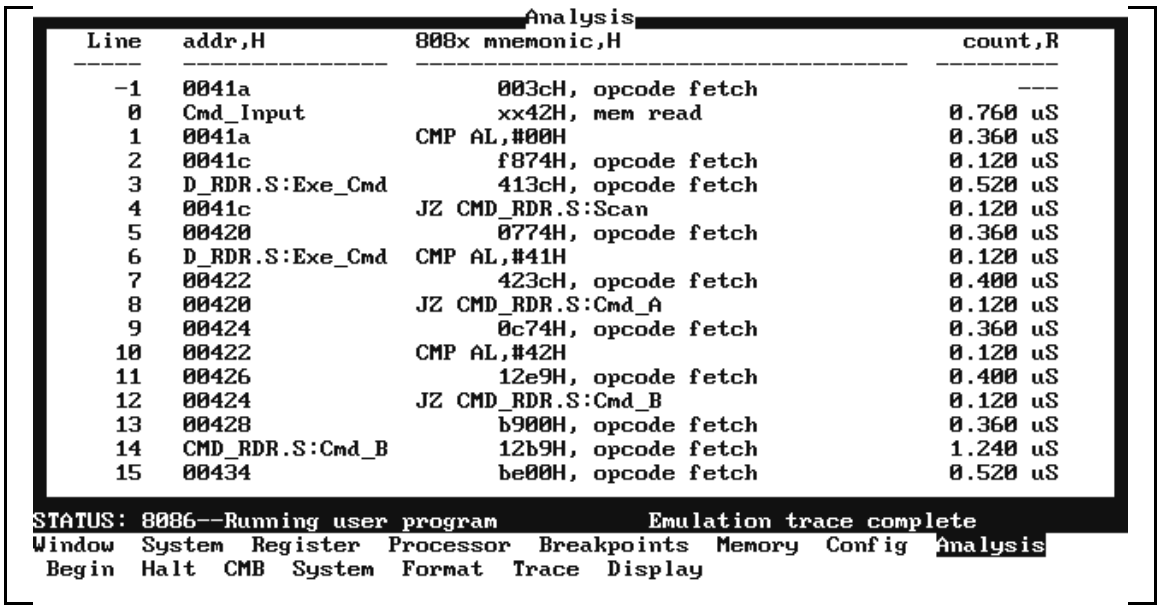
Note

If you choose to dump a complete trace into the trace buffer, it will take a minute or so to display the trace.

You now have two fields in which to specify the states to display. Use the right arrow key to move the cursor to the "Ending state to display" field. Type 260 into the ending state field, press **Enter**, and use <CTRL>**z** to zoom the analysis window. Press the **Home** key to move to the top of the trace list.

When symbol transfer is supported, the PC Interface will show the states available for display and the default disassembly mode. By specifying the disassembly mode, you can display addresses, symbols, or both addresses and symbols in the trace list. The default is "Both," indicating that addresses and symbols will be displayed. To display only symbols, select "Symbols" as the disassembly mode. To display only addresses, select "Addresses" as the disassembly mode.

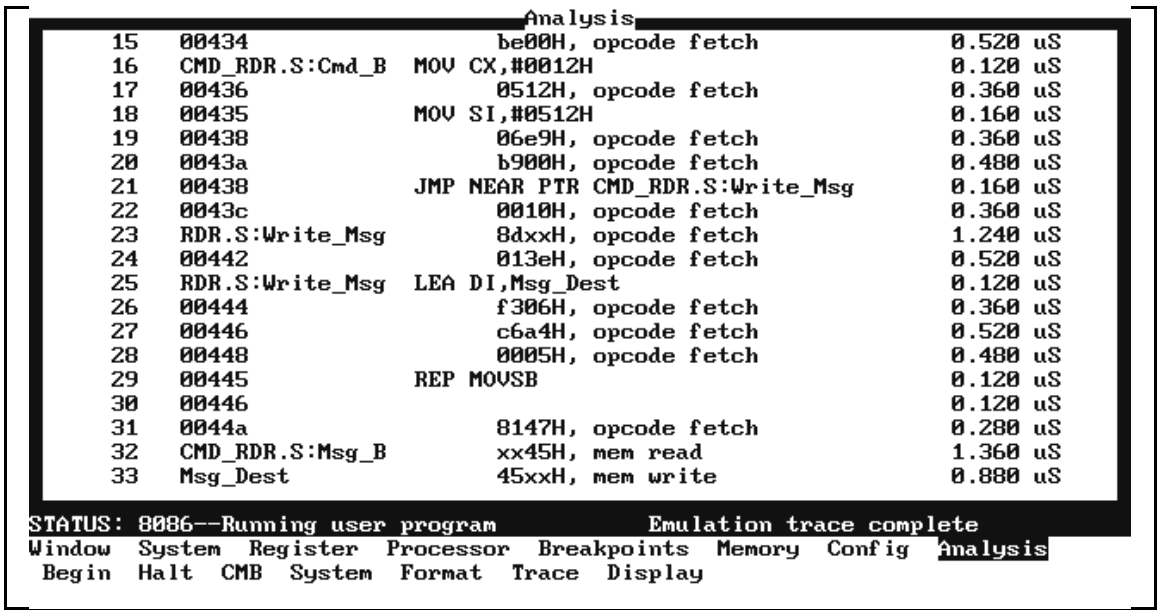
The resulting trace is similar to the trace shown in the following display. (The trace listings that follow are of program execution on the 8086 emulator. Trace listings of program execution on the 8088 emulator look different because of the 8-bit data bus. For example, opcodes are fetched a byte at a time.)



Line 0 in the preceding trace list shows the state that triggered the analyzer. The trigger state is always on line 0. The other states show the exit from the **Scan** loop, the **Exe_Cmd** and **Cmd_B** instructions. Notice that the trace list includes prefetches of instructions that do not get executed (lines 11 and 13).

Press the **PgDn** key to see more lines of the trace.

The following trace list shows the jump to **Write_Msg** and the beginning of the repeat instruction.



Changing the Trace Format

You can modify the trace list format to suit your needs. You can:

- Widen the address column to accommodate longer symbol names.
- Change the port base; to octal, for example.
- Change the count from relative to absolute.

The default trace display format for the 8086/88 emulator includes:

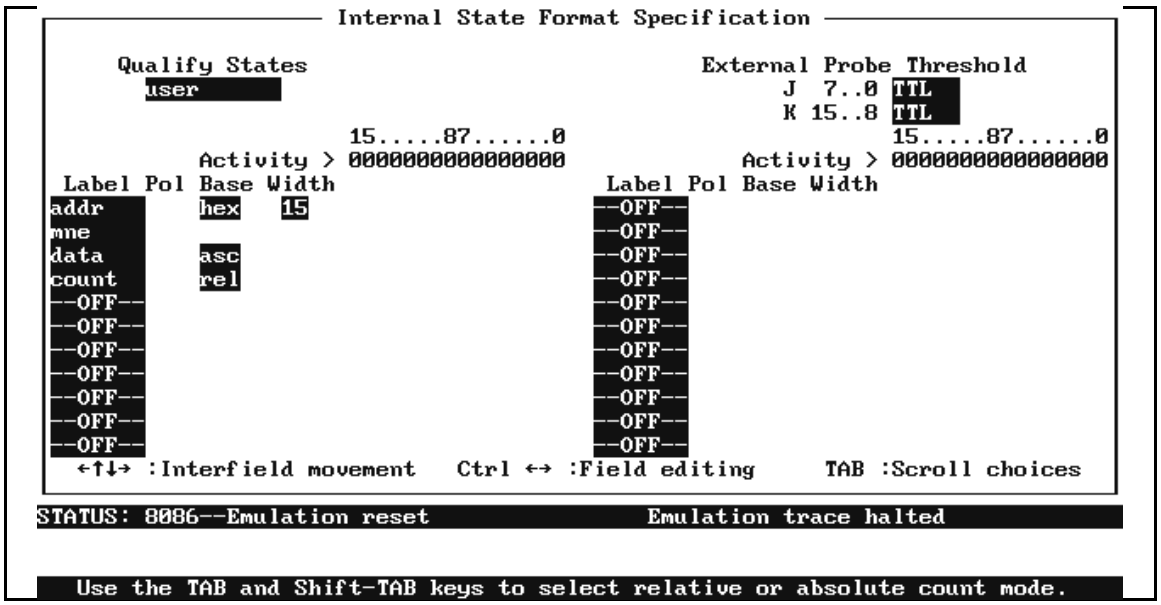
- Trace line number (which is always displayed)
- Hexadecimal address
- 8086/88 mnemonic
- Relative count

You can make the trace format more useful for this example by adding a data character.

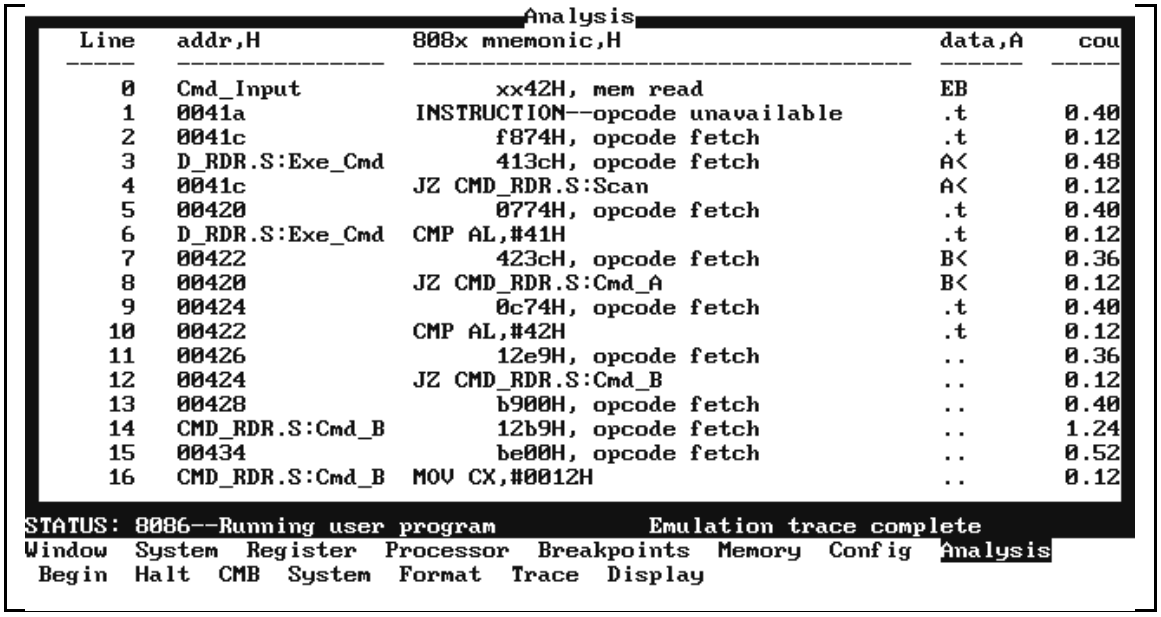
To change the trace list format, enter:

```
Analysis Format Internal
```

Use the cursor keys to move the arrow to the **count** field. Press the **Tab** key until it says **data**. Press **Enter. Tab** in the new field until it shows **asc**. Press **Enter**. **Tab** to change the field name to **count**. Press **Enter**. The following display is the result.



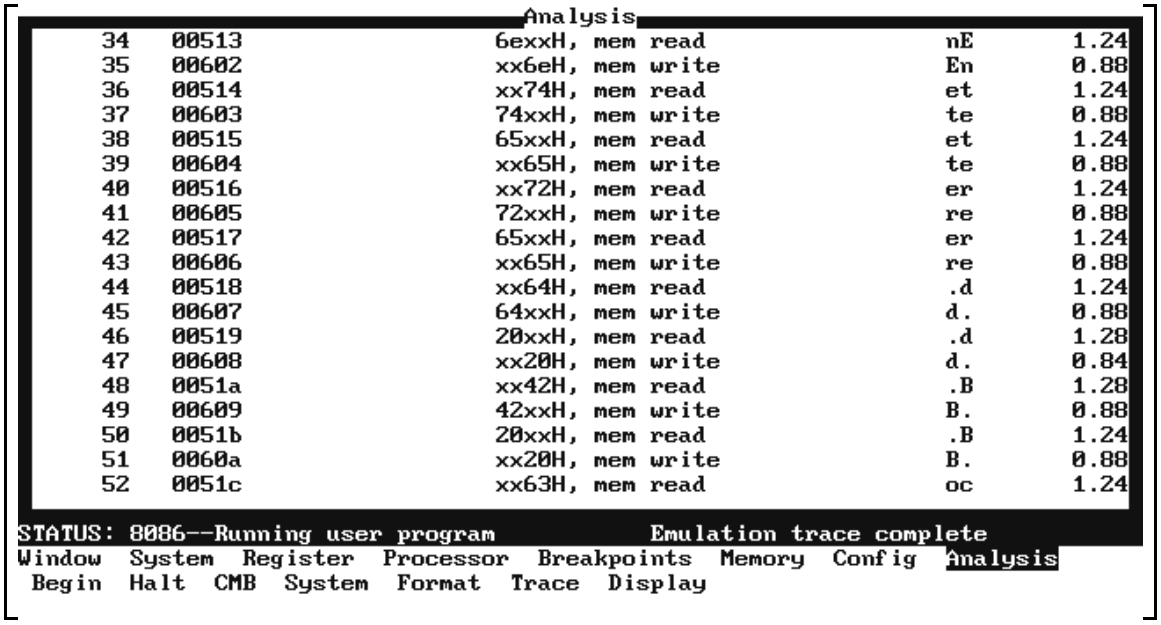
Press the **End** and **Enter** keys to save the new trace format specification. When you display the trace list now with the "**A**nalysis **D**isplay" command, the trace format will resemble the following listing.



Note

Notice that symbols are displayed in the trace list. If your emulator does not have symbol-handling capability, the symbols won't be displayed.

To see the bus activity for moving the message to the destination location, press the **Page Down** key twice. You'll see the display that follows.



For a Complete Description

For a complete description of using the HP 64700-Series analyzer with the PC Interface, refer to the *Analyzer PC Interface User's Guide.*

Copying Memory

You can copy the contents of one range of memory to another. This is a useful feature to test things like the relocation of programs. To test whether the sample program is relocatable within the same segment, copy the program to an unused, but mapped, area of emulation memory. For example, select:

```
Memory Copy
```

Enter 400 through 452 as the source memory range to be copied, and enter 700 as the destination address.

To verify that the program is relocatable, run it from its new address by selecting:

```
Processor Go Address
```

Enter 700. The status line shows that the emulator is "Running user program." You may wish to trace program execution or enter valid and invalid commands and search the message destination area (as shown earlier in this chapter) to verify that the program is working correctly from its new address.

Resetting the Emulator

To reset the emulator, select:

```
Processor Reset Hold
```

The emulator is held in a reset state (suspended) until a "**P**rocessor **B**reak," "**P**rocessor **G**o," or "**P**rocessor **S**tep" command is entered. A CMB execute signal also will run the emulator if reset.

You can also specify that the emulator begin executing in the monitor after reset instead of remaining in the suspended state. To do this, select:

```
Processor Reset Monitor
```

Exiting the PC Interface

There are three ways to exit the PC Interface. You can exit the PC Interface using the "locked" option, which restores the current configuration next time you start up the PC Interface. You can select this option as follows.

```
System Exit Locked
```

Another way to execute the PC Interface is with the "unlocked" option, which reinitializes the emulator the next time you start the PC Interface. You can select this option with the following command.

```
System Exit Unlocked
```

Or, you can exit the PC Interface without saving the current configuration using the command:

```
System Exit No_save
```

See the Emulator PC Interface Reference for a complete description of the system exit options and their effect on the emulator configuration.

3

In-Circuit Emulation

Introduction

The emulator is *in-circuit* when it is plugged into the target system. This chapter covers topics on in-circuit emulation.

This chapter:

- Describes the issues concerning the installation of the emulator probe into target systems.
- Shows you how to install the emulator probe.
- Shows you how to use features related to in-circuit emulation.

Prerequisites

Before performing the tasks described in this chapter, you should be familiar with general emulator operation. Refer to the *System Overview* manual and the "Getting Started" chapter of this manual.

Installing the Emulator Probe into a Target System

The emulator probe has a 40-pin Dual In-Line Package (DIP) connector.

Caution 

Possible Damage to the Emulator Probe. The emulator probe comes with a pin extender. *Do not use the probe without a pin extender installed*. Replacing a broken pin extender is much less expensive than replacing the emulator probe.

Do not use more than one pin extender, unless it is needed for mechanical clearance reasons, because pin extenders degrade signal quality.

The emulator probe also comes with a foam pin protector to: (1) protect the probe from damage due to electrostatic discharge (ESD), and (2) protect the delicate gold-plated pins of the probe connector from damage due to impact. The foam pin protector is conductive and must be removed before running performance verification or using the emulator.

Caution 

Possible Damage to the Emulator Probe. The emulator (emulation) probe contains devices that are susceptible to damage by static discharge. Take precautions before handling the microprocessor connector attached to the end of the probe cable to avoid damaging the internal components of the probe.

Caution 

Possible Damage to the Emulator. Make sure target system power is OFF before installing the emulator probe into the target system.

Caution 

Damage to the Emulator Probe Will Result if the Probe is Incorrectly Installed. Make sure pin 1 of the probe connector is aligned with pin 1 of the socket.

Auxiliary Output Lines

There are three auxiliary output lines provided by the emulator:

Caution

Damage to the Emulator Probe Will Result if the Auxiliary Output Lines are Incorrectly Installed. When installing the auxiliary output lines into the end of the emulator probe cable, make sure that the ground pins on the auxiliary output lines (labeled with white dots) match the ground receptacles in the end of the probe cable.

TGT BUF DISABLE

This active-high output is used when the emulator is configured to allow external DMA accesses to emulation memory (see the "Configuring the Emulator" chapter). Use it to tristate (in other words, select the high Z output) any target system devices on the 808X address/data bus. Target system devices should be tristated since reads from emulation memory (by the emulation processor or an external device) will output data on the user probe.

The TGT BUF DISABLE signal goes true at the start of clock cycle T2 in any bus cycle that accesses emulation memory if external DMA is enabled. It goes false during T4.

8087 INT

This active-high output is the internal 8087's INT output. If you have enabled the internal 8087 (see the "Configuring the Emulator" chapter), are using the internal 8087 interrupts, but have not configured the internal 8087 to drive the 808X INTR input, this output must be connected to the target system interrupt controller.

SYSTEM RESET

This active-high, CMOS output should be used to synchronously reset the emulator and the target system. You should use this when an 8089 I/O processor is in the target system, because the coprocessor

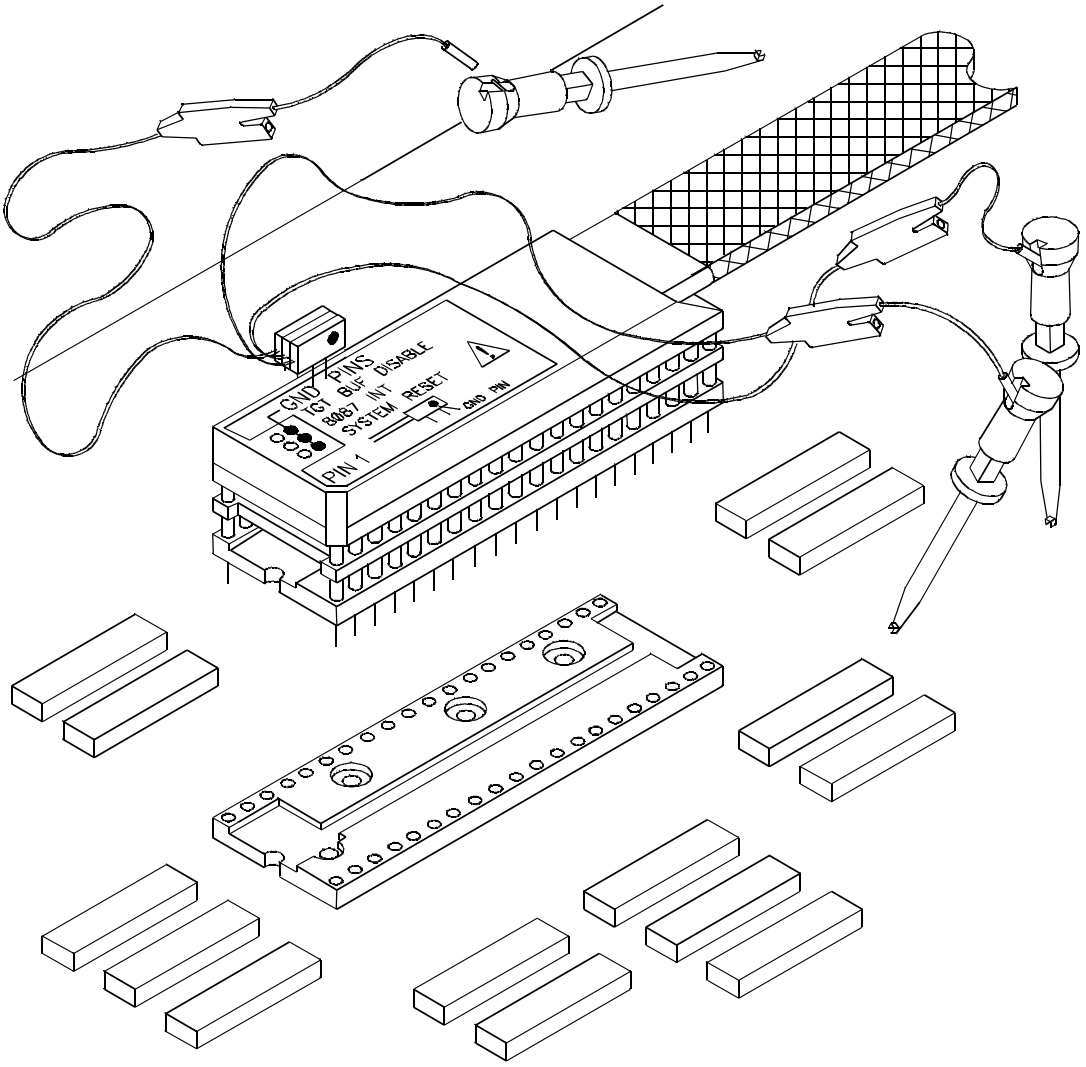


Figure 3-1. Connecting the Emulator Probe

interpretation of the channel attention (CA) input is relative to the last reset.

In-Circuit Configuration Options

The 8086/8088 emulators provide configuration options for the following in-circuit emulation issues. Refer to the chapter on "Configuring the 8086/8088 Emulator" for more information on these configuration options.

Using the Target System Clock Source

You can configure the emulator to use the external target system clock source.

Allowing the Target System to Insert Wait States

High-speed emulation memory provides no-wait-state operation. The emulator may optionally respond to the target system ready lines during emulation memory accesses.

Selecting Visible/Hidden Background Cycles

Emulation processor activity while executing in background can either be visible to the target system (cycles are sent to the emulator probe) or hidden (cycles are not sent to the emulator probe).

Defining the Emulator's Queue Status in Background

When the 8086 is in maximum mode, the queue status is output on lines QS0 and QS1. You can configure the emulator to output either a FLUSH or NOP queue status while it is executing in background.

Running the Emulator from Target Reset

You can specify that the emulator begin executing from target system reset. When the target system RESET line becomes active and then inactive, the 8086/8088 registers are initialized to their reset values. The emulator begins running from 0FFFF0H (this will occur within a few cycles of the RESET signal). To specify a run from target reset, select:

```
Processor Go Reset
```

The status now shows that the emulator is "Awaiting target reset." After the target system is reset, the status line message will change to show the appropriate emulator status.

You also can enter the "**P**rocessor **G**o **R**eset" command with the target system powered down. The emulator will respond with the "Slow clock" status (because the external clock is automatically selected). The emulator will prepare itself internally for foreground operation. When the target is powered up and toggles RESET, the emulator will run from 0FFFF0H.

4

Configuring the 8086/88 Emulator

Introduction

Your 8086 or 8088 emulator can be used in all stages of target system development. For instance, you can run the emulator out-of-circuit when developing target system software, or you can use the emulator in-circuit when integrating software with target system hardware. Emulation memory can be used for or with target system memory. You can use the emulator's internal clock or the target system clock. You can execute target programs in real-time or divert emulator execution into the monitor when commands request access of target system resources (target system memory or I/O, or register contents).

The emulator is a versatile instrument and may be configured to suit your needs at any stage of the development process. This chapter describes the configuration options for the 8086 or 8088 emulator.

This chapter:

- Shows you how to access the emulator configuration options.
- Describes the emulator configuration options (in the order of the configuration display).
- Shows you how to save a particular emulator configuration, and load it again at a later time.
-
-

Prerequisites

Before performing the tasks described in this chapter, you should be familiar with general emulator operation. Refer to the *System Overview* manual and the "Getting Started" chapter of this manual.

Access Emulator Configuration Options

To access the 8086/88 emulator configuration options, select:

```
Config General
```

When you position the cursor to a configuration item, a brief description of the item appears at the bottom of the display.

Note

You can use the System Terminal window to modify the emulator configuration. But, if you do this, some PC Interface features may no longer work properly. We recommend that you only modify the emulator configuration by using the options presented in the PC Interface.

You change items in the configuration screen by using the arrow keys to move to the selected item, then typing a value. Press the **End** and **Enter** keys to exit from the configuration form and save your changes.

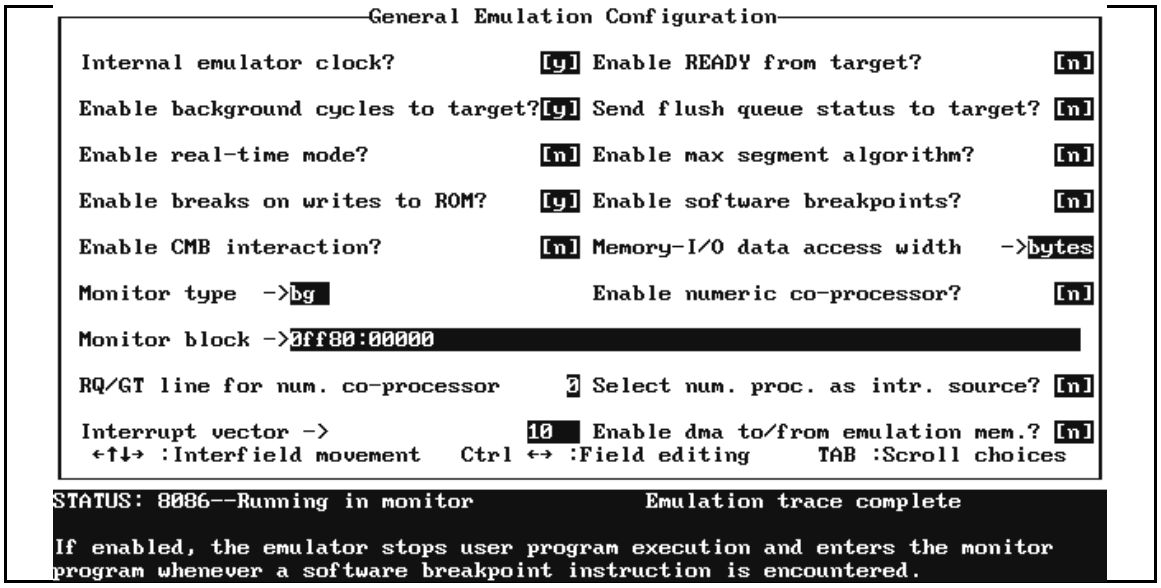


Figure 4-1. General Emulator Configuration

Internal Emulator Clock?

This configuration item allows you to select whether the emulator will be clocked by the internal clock source or by a target system clock source.

Yes The internal 8 MHz clock oscillator is the emulator clock source. This is the default.

No An external target system clock is the emulator clock source. External clock sources must be within the range 2-10 MHz.

Enable READY from Target?

High-speed emulation memory provides no-wait-state operation. The emulator may optionally respond to the target system ready lines during emulation memory accesses.

No When the ready relationship is not locked to the target system, emulation memory accesses ignore the ready signal from the target system (no wait states are inserted).

Yes When the ready relationship is locked to the target system, emulation memory accesses honor the ready signal from the target system (wait states are inserted if requested).

Enable Background Cycles to Target?

Emulation processor activity while running in the background monitor can either be visible to the target system (cycles are sent to the emulator probe) or hidden (cycles are not sent to the emulator probe).

Yes The default emulator configuration specifies that background activity is visible.

If your target system requires that the emulator always appear running (for example, to refresh dynamic memories), you should allow background cycles to be visible to the target system.

When background cycles are visible, they appear to the target system as “reads” from the address range of the monitor. If you need to put the monitor in a memory range where read operations will not cause an undesired interaction, you can change the monitor’s base address. (Refer to the “Monitor Block?” configuration item.)

No When a break occurs and background cycles are disabled (hidden), the emulator appears to the target system to have suspended operation until a return to foreground. When cycles are disabled, background cycles are blocked (S0-S2 remain high and /RD, /WR, /DEN, ALE, and /INTA remain inactive).

Send Flush Queue Status to Target?

When the 8086 is in the maximum mode, the queue status is output on lines QS0 and QS1. The QS0 and QS1 signals allow external processors that receive instructions and operands via the ESC instruction to track the ESC instruction through the queue to see if it executes.

No By default (if in maximum mode), the emulator outputs a NOP status on lines QS0 and QS1 while in background.

Yes The emulator (if in maximum mode) outputs a FLUSH queue status while in background.

Enable Real-Time Mode?

If the emulator must execute target system programs in real-time, you can enable the real-time emulator mode. In other words, when you execute target programs (with the "**P**rocessor **G**o" command), the emulator will execute in real-time.

No The default emulator configuration disables the real-time mode. When the emulator is executing the target program, you can enter emulation commands that require access to target system resources (display/modify registers, target system memory, or target system I/O). If you enter one of these commands, the system controller will temporarily break emulator execution into the monitor.

Yes If your target system program requires real-time execution, you should enable the real-time mode to prevent temporary breaks that might cause target system problems.

Commands Not Allowed in Real-Time Mode

When emulator execution is restricted to real-time and the emulator is running user code, the system refuses all commands that require access to processor registers or target system memory or I/O. The following commands are not allowed when runs are restricted to real-time:

- Register display/modification.
- Target system memory display/modification. (Because the emulator contains dual-port emulation memory, commands that access emulation memory do not require breaks and are allowed while runs are restricted to real-time.)
- I/O display/modification.

If the real-time mode is enabled, these resources can only be displayed or modified while running in the monitor.

Breaking out of Real-Time Execution

The only commands that can break real-time execution are:

```
Processor Reset
Processor Go
Processor Break
Processor Step
```

Enable Max Segment Algorithm?

The "**P**rocessor **G**o" and "**P**rocessor **S**tep" commands allow you to enter addresses in either logical form (segment:offset, for example, 0F000:0FFFF) or physical form (for example, 0FFFFF).

When you enter a physical address (non-segmented) with either a "**P**rocessor **G**o" or "**P**rocessor **S**tep" command, the emulator must convert it to a logical (segment:offset) address.

If you use logical addresses other than the two methods that follow, you must enter addresses in logical form.

No By default, a physical run address is converted such that the low 16 bits of the address become the offset value. The physical address is right-shifted 4 bits and ANDed with 0F000H to yield the segment value.

```
logical_addr = ((phys_addr >> 4) & 0xf000):(phys_addr & 0xffff)
```

Yes Specifies that the low 4 bits of the physical address become the offset. The physical address is right-shifted 4 bits to yield the segment value.

```
logical_addr =  (phys_addr >> 4):(phys_addr & 0xf)
```

Enable Breaks on Writes to ROM?

Emulator execution may optionally break into the monitor when the target (user) program writes data to a location mapped as ROM.

Yes Emulator execution will break into the monitor when the target program writes to ROM locations.

No Target program writes to ROM locations will not cause emulator execution to break into the monitor.

Enable Software Breakpoints?

The software breakpoint feature uses the 8086/88 single-byte interrupt facility. When you add or set a software breakpoint (and software breakpoints are enabled), the emulator will replace the opcode at the software breakpoint address with the breakpoint interrupt instruction (INT 3). When the emulator executes the INT 3 instruction, execution breaks into the monitor.

If your target program uses single-byte interrupt instructions and contains a breakpoint interrupt routine, you may wish to disable the software breakpoints feature. Then, INT 3 instructions do not cause breaks to the monitor.

Refer to the "Getting Started" for information on using software breakpoints.

No The software breakpoints feature is disabled. This is the default emulator configuration, so you must change this item before you can use software breakpoints.

Yes The software breakpoints feature is enabled. The emulator breaks to the monitor when an INT 3 is executed. If the interrupt instruction is a software breakpoint, the original opcode is restored in the user program. A subsequent "**P**rocessor **G**o **P**c" or "**P**rocessor **S**tep **P**c" command will execute from the breakpoint address.

If the INT 3 instruction is not a software breakpoint, an "undefined breakpoint" status message is displayed. To continue with program execution, you must use the "**P**rocessor **G**o **A**ddress" or "**P**rocessor **S**tep **A**ddress" command and specify the target program's breakpoint interrupt vector address.

Enable CMB Interaction?

Coordinated measurements are measurements synchronously made in multiple emulators or analyzers. Coordinated measurements can be made between HP 64700-Series emulators, which communicate over the Coordinated Measurement Bus (CMB).

Multiple emulator starts/stops is one type of coordinated measurement. The CMB signals READY and /EXECUTE are used to perform multiple emulator starts/stops.

This configuration item allows you to enable/disable interaction over the READY and /EXECUTE signals. (The third CMB signal, TRIGGER, is unaffected by this configuration item.)

No The emulator ignores the /EXECUTE and READY lines, and the READY line is not driven.

Yes Multiple emulator starts/stops are enabled. If the

```
Processor CMB Go ...
```

command is entered, the emulator begins executing code when a pulse on the /EXECUTE line is received. The READY line is driven false while the emulator is running in the monitor. It goes true whenever execution switches to the user program.

Note 

CMB interaction will also be enabled when the

```
Processor CMB Execute
```

command is entered.

Memory-I/O Data Access Width?

This configuration item allows you to specify the type of microprocessor cycles used by the monitor program to access target memory or I/O locations. When a command requests the monitor to read or write target system memory or I/O, the monitor program examines this configuration item to determine whether to use byte or word instructions.

Bytes Selecting the byte access mode specifies that the emulator will access target memory using upper and lower byte cycles (one byte at a time). The default emulator configuration specifies a data access width of bytes.

Words Selecting the word access mode specifies that the emulator will access target memory using word cycles (one word at a time).

Monitor Type?

The emulation processor executes the monitor program. It allows the emulation system controller to access target system resources. For example, when you enter a command that requires access to target system resources (display target memory, for example), the system controller writes a command code to a communications area and breaks the execution of the emulation processor into the monitor. The monitor program then reads the command from the communications area and executes the processor instructions to access the target system. After the monitor has performed its task, execution returns to the target program. Monitor program execution can take place in the "background" or "foreground" emulator modes.

In the *foreground* emulator mode, the emulator operates as would the target system processor.

In the *background* emulator mode, foreground execution is suspended so that the emulation processor may be used for communication with the system controller, typically to perform tasks that access target system resources.

A *background monitor* program operates entirely in the background emulator mode. That is, the monitor program does not execute as if it were part of the target program. The background monitor does not take up any processor address space and does not need to be linked to the target program. The monitor resides in dedicated background memory.

A *foreground monitor* program performs its tasks in the foreground emulator mode. That is, the monitor program executes as if it were part of the target program. Breaks into the monitor always put the emulator in the background mode. But, foreground monitors switch back to the foreground mode before performing monitor functions.

The default emulator configuration selects the background monitor. You can change the configuration to select the foreground monitor. Also, you can select two other options: the user background monitor, or the user foreground monitor. These four monitor options are listed below.

bg Default background monitor.

fg Default foreground monitor.

ubg Default background monitor with the addition of a user supplied routine (less than 512 bytes of absolute code), which is executed while in the monitor. User code must first be loaded using the "B_Monitor" memory load option.

ufg User supplied foreground monitor. Allows use of a custom foreground monitor (less than 2K bytes of absolute code) for a specific target system. The user foreground monitor must first be loaded using the "F_Monitor" memory load option.

Note 

Remove software breakpoints before changing the monitor type. Otherwise, SBI instruction (INT 3) opcodes will be left at unknown locations.

Note 

All memory mapper terms are deleted when the monitor type is changed!

Background

The default emulator configuration selects the background monitor by default. When you use the background monitor:

- Target programs should set up the stack in memory mapped as emulation or target RAM. The stack must be present to use software breakpoints.
- Guarded memory accesses can occur if the vector table area, 0-3FFH, is mapped as "guarded memory." (If locations 0-3FFH are not mapped, and unmapped memory is assigned

the “grd” memory type, then these locations act as guarded memory.)

- Halt instructions will cause “processor halted” emulation status. A subsequent break command, followed by a run or step command, repeats the halt instruction.

Note

Stepping into a HLT instruction will not halt the processor.

The 8086 processor will not halt when an interrupt occurs while a HLT instruction is executed.

User Background

The emulator allows you to insert code into the background monitor. Limit your code to four sections of 128 bytes each (the absolute file should be less than 2048 bytes long). Code in the first section gets executed on monitor entry. Code in the second section gets executed once for each loop through the monitor. Code in the third section is executed on monitor exit. And, code in the fourth section executes when the monitor is entered from reset. User code is subject to the following restrictions:

- User code must be at 400H. This is not the absolute address of the user code. It is the offset within the monitor segment. A template for user code programs comes with the emulator and is shown below. *Always refer to the shipped file for the most recent version.*

- The user code must not contain instructions that use the stack (PUSH, POP, CALL, RET, and so on). The background monitor makes no assumptions about the existence of a stack in foreground code and does not contain any instructions that use the stack. Six bytes of monitor memory save values normally saved on the stack: CS, IP, and the flags.

- The user code must not write to monitor locations outside the area to which the user code is restricted. The background monitor uses locations in the reserved 2K bytes to communicate with the emulation system controller.

- The user code must not jump to locations outside the area to which it is restricted. Other locations in the 2K bytes reserved for the monitor contain the monitor program and data. Also, jumping to certain locations outside the range restricted to user code will put the emulator into different modes of operation. These modes allow the background monitor to access target system resources when executing emulation commands. Refer to the “Other Emulator Modes” description in the “Foreground Monitor Description” appendix.

- The user code must not change the contents of the CS or SS registers.

```
"8086"
;@(mktid) Lab Proto(1.00)
;
; Template for using background monitor features in user background code
;
; Following is a memory map of the background monitor.  The monitor always
; occupies 2Kbytes of space.  User code is always installed at offset 400H.
;
;------------------------------------------------------------- --------------
;
;   000H ************************************************
;        *  IP,CS and flag jam area (all 8 bytes used)   *
;   008H ************************************************
;        *  Vector area                                  *
;   00CH ************************************************
;        *  Communications area                          *
;   020H ************************************************
;        *  I/O area 0                                   *
;   030H ************************************************
;        *  I/O area 1                                   *
;   038H ************************************************
;        *  Set BGCPCYC flag                             *
;   040H ************************************************
;        *  Set JAMBKGR flag                             *
;   048H ************************************************
;        *  Reset JAMBKGW flag                           *
;   050H ************************************************
;        *  Set BKGPS flag                               *
;   058H ************************************************
;        *  Reset BKGPS flag                             *
;   060H ************************************************
;        *  Set BKGWTT flag                              *
;   068H ************************************************
;        *  Reset BKGWTT flag                            *
;   070H ************************************************
;        *  Set BKGRFT flag                              *
```

```
;   078H ************************************************
;        *  Reset BKGRFT flag                            *
;   080H ************************************************
;        *  Monitor Area                                 *
;   380H ************************************************
;        *  Register Area                                *
;   400H ************************************************
;        *  Execute on Entry User code area              *
;   480H ************************************************
;        *  Execute while in Monitor User code area      *
;   500H ************************************************
;        *  Execute on Exit User code area               *
;   580H ************************************************
;        *  Execute on Reset User code area              *
;   6E0H ************************************************
;        *  Monitor buffer area                          *
;   7F0H ************************************************
;        *  Background reset area                        *
;   7FFH ************************************************
;

;-------------------------------------------------------------- --------------
; I/O Area 0
;
; A read from this area will bring in the following emulator status flags:
;
; Bit     Flag
;
;  0      Break request
;  1      Run request
;  2      Was Halted
;  3      Sixteen bit processor
;
; A write to this area will set the ready flag true.
;-------------------------------------------------------------- --------------
; I/O Area 1
;
; A read from this area does the same thing as a read from I/O area 0.
;
; A write to this area sets the jam counter to the value written (only bit
; D0 is used).
;-------------------------------------------------------------- --------------
; Locations 38H thru 7FH are special in that they require an opcode
; fetch from the appropriate range to set or reset the indicated flag.
; In all cases except for setting the jam read flag, JAMBKGR, the desired
; function must be called using the macro sfunc (sfunc guarantees that only
; opcode fetches are generated).
;-------------------------------------------------------------- --------------

JAMAREA         EQU     000H
VECTAREA        EQU     008H
COMMAREA        EQU     00CH
IOAREA0         EQU     020H
IOAREA1         EQU     030H
MONAREA         EQU     080H

REGAREA         EQU     380H

ENTRYUAREA      EQU     400H
CONTUAREA       EQU     480H
EXITUAREA       EQU     500H
```

```
RESETUAREA      EQU     580H

BUFAREA         EQU     6E0H
RESETAREA       EQU     7F0H
TRUE            EQU     1
FALSE           EQU     0

SPECEN0         EQU     00001100000B
SPECEN1         EQU     00001000000B
SPECEN2         EQU     00000100000B

BPA             EQU     00000B
BPB             EQU     01000B
BPC             EQU     10000B
BPD             EQU     11000B

IRETTOFG        EQU     00001000000B         ;SPECEN1 + BPA
CLRJAMBKGW      EQU     00001001000B         ;SPECEN1 + BPB

BREAKMASK       EQU     0001B
RUNMASK         EQU     0010B
WASHALTEDMASK   EQU     0100B
SXTNSELMASK     EQU     1000B
CMDAVAIL        EQU     0
CMDCOMPLETE     EQU     0FFFFH
INRFGLOOP       EQU     0FFFFH


; These functions may be useful.  They are called in the following manner:
;
; SFUNC  ame
;
; Where  ame (in lower case!!!) is one of the following:

; Force internal co-processor memory accesses to go to background memory
SETBGCPCYC      EQU     00000111000B         ;SPECEN2 + BPD
setbgcpcyc      ORG     SETBGCPCYC

; Present real status to the target system.
SETBKGPS        EQU     00001011000B         ;SPECEN1 + BPD
setbkgps        ORG     SETBKGPS

; Substitute either nothing or memory read for real status to the target
; (Depending on the setting of the ^cyc^ configuration item)
CLRBKGPS        EQU     00001010000B         ;SPECEN1 + BPC
clrbkgps        ORG     CLRBKGPS

; Send background writes to the target system.
SETBKGWTT       EQU     00001101000B         ;SPECEN0 + BPB
setbkgwtt       ORG     SETBKGWTT

; Send background writes to monitor memory.
CLRBKGWTT       EQU     00001100000B         ;SPECEN0 + BPA
clrbkgwtt       ORG     CLRBKGWTT

; Get background reads from monitor memory.
CLRBKGRFT       EQU     00001110000B         ;SPECEN0 + BPC
clrbkgrft       ORG     CLRBKGRFT

; Get background reads from the target system.
SETBKGRFT       EQU     00001111000B         ;SPECEN0 + BPD
```

```
setbkgrft       ORG     SETBKGRFT


;
; Macros
;
;

SFUNC           MACRO   &SUBADDR
                MOV     BP,#($+6)
                JMP     NEAR PTR &SUBADDR
                MEND

SFUNCRET        MACRO
                JMP     BP
                MEND

MONCALL         MACRO
                MOV     BX,#($+5)
;               JMP     [SI]
                DB      0FFH,024H
                MEND

MONRET          MACRO
                JMP     BX
                MEND

; User code macros
;
; These macros are used to get to and return from user routines. Note that
; if BX is to be used, it must be saved and restored before executing a
; UCODERET.

UCODECALL       MACRO   &ULOC
                MOV     BX,#($+6)
                JMP     NEAR PTR &ULOC
                MEND

UCODERET        MACRO
                JMP     BX
                MEND

                ASSUME  CS:ORG,DS:ORG,ES:ORG

                ORG     ENTRYUAREA

; User code that is to execute on monitor entry goes here
;
; 1. dont use the stack
; 2. called on entry into the monitor
; 3. dont modify BX!!

                UCODERET



                ORG     CONTUAREA

; User code that is to execute on a continuous basis goes here. This code
; is called whenever the monitor has nothing else to do.
;
```

```
; 1. dont use the stack
; 2. called once each monitor loop
; 3. dont modify BX!!
; ##################################################################- ############
; Example to refresh DRAM
;
; This routine simply reads a word from every memory location below 80000H.
; This might be used as a replacement for DMA type refresh while in
; background.

                LDS     SI,CS:userptr           ;get word ptr to loc to read
                LODSW                           ;read it and inc si
                MOV     WORD PTR CS:userptr,SI  ;save it for next time
                CMP     SI,0                    ;is SI zero?
                JE      modseg                  ;if so skip
                UCODERET                        ;return
modseg:
                MOV     SI,DS                   ;get ds
                CMP     SI,7000H                ;is it 7000H?
                JE      zeroseg                 ;if so skip
                ADD     SI,1000H                ;else add 1000H
                MOV     WORD PTR CS:userptr+2,SI ;save it
                UCODERET                        ;return
zeroseg:
                MOV     SI,0                    ;clear si
                MOV     WORD PTR CS:userptr+2,SI ;put in seg location
                UCODERET                        ;return

; Define data
userptr         DD      0


; ##################################################################- ############
; End example


                ORG     EXITUAREA

; User code that is to execute on monitor exit goes here
;
; 1. dont use the stack
; 2. called on exit from the monitor
; 3. dont modify BX!!

                UCODERET


                ORG     RESETUAREA

; User code that is to execute on monitor reset goes here
;
; 1. dont use the stack
; 2. called when the monitor is reset
; 3. dont modify BX!
; 4. a good place to set up memory/peripheral select lines

                UCODERET
```

Loading User Code

You must load user code to be placed in the background monitor before you can select the user background monitor type. Use the "B_Monitor" memory load option when loading user code.

Foreground

The foreground monitor uses processor address space. The foreground monitor uses 2K bytes of memory (at 0FF800H by default—see the "Monitor Block?" configuration item).

Note

You must *not* use the foreground monitor if you want to make coordinated measurements.

More About the Foreground Monitor

The monitor, whether background or foreground, is the interface between the emulation system controller and the target system. The monitor allows commands that display and modify the contents of target system memory, I/O ports, and processor registers. It also enables commands that step through program execution.

When you select a background monitor, its execution is hidden from the target system (except background cycles, optionally). When the emulator is executing in the monitor, it appears to the target system as if it has suspended operation.

When you select the foreground monitor, the monitor performs its tasks in the foreground emulator mode (where the emulator acts just as the microprocessor it replaces). The monitor remains in the 2K bytes of emulation memory reserved. The remaining emulation memory (either 126K or 510K bytes, depending on the emulator model number) is at your disposal. But, when the foreground monitor is selected, the monitor occupies 2K bytes of 8086 memory space.

When you select the foreground monitor, breaking into the monitor still occurs in background. The rest of the monitor program functions are executed in foreground.

Note

Foreground monitors can cause breaks (a foreground monitor that accesses guarded memory, for example). If these breaks occur consistently within approximately 10 ms of monitor entry, the emulator will become unresponsive. Each time a break to the monitor occurs, an access of guarded memory will occur, which causes a break into the monitor, and so on. If this happens, you must cycle power to the emulator.

Using the Foreground Monitor

When you use the foreground monitor:

- Your program must set up a stack. The foreground monitor assumes that there is a stack in the foreground program. This stack is used to save CS, IP, and the flag word upon entry into the monitor.
- You must set up your vector table to point to locations in the foreground monitor program. The vector table (shown in the following listing) contains assembly language pseudo-ops that define vectors, which point to the proper locations in the foreground monitor. The "step" feature of the emulator uses the single-step interrupt vector, and the software breakpoints feature uses the breakpoint interrupt vector. The segment portion of the logical addresses defined in your vector table should match the location you have selected, or will select, for the monitor program. (The segment values in the vector table file that follows match the default location of the monitor.)
- Guarded memory accesses can occur if no vector table is loaded and the vector table area, 0-3FFH, is mapped as "guarded memory." (If locations 0-3FFH are not mapped, and unmapped memory is assigned a type of "grd", then these locations act as guarded memory.)
- Halt instructions will cause "processor halted" emulation status. A subsequent break command, followed by a run or step command, repeats the halt instruction.

How to Use the Foreground Monitor

The processor's interrupt vector table must be loaded with the correct address of the interrupt service routine associated with that interrupt type. The interrupt type is an integer, which specifies one of 256 memory addresses. The address is found by multiplying the interrupt type by 4. Thus, a type 1 interrupt refers to address 4, a type 2 interrupt refers to address 8, and so on. Each of these memory addresses identifies a complete logical address (two bytes for the segment and two bytes for the offset).

When an interrupt occurs, the processor calculates the interrupt vector address and loads the CS and IP registers with the address of the interrupt routine. Program execution transfers to the new location.

An example program comes with the emulator that performs a partial initialization of the processor's interrupt table. This program is called V64762.S for the 80186/C186 emulator, and V64763.S for the 80188/C188 emulator.

The beginning of the program appears in the following listing.

```
"8086"

; Vector table
;
; This table defines monitor entry points other than by breaking. To use
; these entry points, the processors vector table must be loaded with
; pointers to these locations.

VTABLEAREA      EQU     00420H
MONSEGMENT      EQU     0FF80H
ENTRYSIZE       EQU     0000AH
SBIAREA         EQU     007E8H
NUMEXCVECT      EQU     00040H
USERVECT        EQU     00080H

                ORG     0

                DW      VTABLEAREA+ENTRYSIZE*0   ; zero divide
                DW      MONSEGMENT

; This vector MUST be present to single step!!!
                DW      VTABLEAREA+ENTRYSIZE*1   ; single step
                DW      MONSEGMENT

                DW      VTABLEAREA+ENTRYSIZE*2   ; user nmi
                DW      MONSEGMENT
```

```
; This vector MUST be present to allow the monitor to handle breakpoints
; properly.
                DW      SBIAREA                 ; single byte int.
                DW      MONSEGMENT

                DW      VTABLEAREA+ENTRYSIZE*4  ; overflow
                DW      MONSEGMENT

                ORG     NUMEXCVECT

                DW      VTABLEAREA+ENTRYSIZE*5  ; numeric exception
                DW      MONSEGMENT

                ORG     USERVECT

                DW      VTABLEAREA+ENTRYSIZE*6
                DW      MONSEGMENT

                DW      VTABLEAREA+ENTRYSIZE*7
                DW      MONSEGMENT

                DW      VTABLEAREA+ENTRYSIZE*8
                DW      MONSEGMENT

                DW      VTABLEAREA+ENTRYSIZE*9
                DW      MONSEGMENT

                DW      VTABLEAREA+ENTRYSIZE*10
                DW      MONSEGMENT

                DW      VTABLEAREA+ENTRYSIZE*11
                DW      MONSEGMENT

                DW      VTABLEAREA+ENTRYSIZE*12
                DW      MONSEGMENT

                DW      VTABLEAREA+ENTRYSIZE*13
                DW      MONSEGMENT
```

Loading this program into the emulator would alter the processor's vector table to:

```
0000H  |  xxxxxx  |
       |----------|
0002H  |  xxxxxx  |
       |----------|
0004H  | 042AH    |  IP
       |----------|      Type 1 interrupt (single-step)
0006H  | 0FF80H   |  CS
       |----------|
0008H  |  xxxxxx  |
       |----------|
000AH  |  xxxxxx  |
       |----------|
000CH  | 07E8H    |  IP
       |----------|      Type 3 interrupt (breakpoint)
000EH  | 0FF80H   |  CS
       |----------|
0010H  |  xxxxxx  |
```

About the Vector Program and Monitor Segment

The monitor uses the single-step and breakpoint hardware interrupts to single-step or break user code. The vector program sets up the vector table to point to the functional procedures located within the monitor program.

The addresses at 06H and 0EH in the interrupt vector table correspond to the address specified in the "Monitor Segment?" emulation configuration question. If you want to use a different address for the monitor, you must modify this configuration (whether you are using the foreground or background monitor). In addition, if you use a foreground monitor, the statement in the vector program (V6476X.S) for the monitor segment (MONSEGMENT EQU XXXX) must reflect any change made to the monitor block location.

For example, if a monitor segment of 0F000H is specified in the configuration, the MONSEGMENT variable in the vector program (V6476X.S) should be equated to 0F000H. The address values at 04H and 0CH are offsets in the segment, and point to the offset location of the interrupt routine within the monitor segment. These values should remain constant unless the "Monitor Offset?" configuration question is modified.

The statements that appear as comments in the vector program can be used to further load the vector table. You can remove comments from this program to load the vector table with the addresses of additional interrupt service routines found in the monitor program. Or you may prefer to load these vector addresses with pointers to custom service routines.

Assembling, Linking, and Loading the Vector Program

The vector program must be assembled, linked and loaded into the emulator if you want to use the foreground monitor. The load address specified to the linker is arbitrary. All addresses are ORGed. The foreground monitor is already resident in the emulator's ROM at the default location. Therefore, no assembling or linking is required. You

simply specify the foreground monitor in the “Monitor Type?” configuration question. If you want to modify the monitor, the foreground monitor source code is in a separate file. Then this program would need to be assembled, linked, and loaded into the emulator.

User Foreground

If you need a customized monitor, you can load it into the 2K byte area reserved for the monitor. When customizing the foreground monitor, you must maintain the basic communication protocol between the monitor and the emulation system controller.

The foreground monitor program source file comes with the emulator, and is described in the “Foreground Monitor Description” appendix.

Loading a User Foreground Monitor

You must load the monitor before you can select the user foreground monitor type. Use the “F_Monitor” memory load option.

Monitor Block?

The default emulator configuration locates the monitor at 0FF800H. You can relocate the monitor to any 2K byte boundary except 0H. The location of the background monitor may be important to specify which target system locations are read if background cycles are made visible to the target system (which is the default case). The location of foreground monitors is important because they will occupy part of the processor address space. Foreground monitor locations must not overlap the locations of target system programs.

When entering monitor block addresses, you must only specify addresses on 2K byte boundaries. Otherwise, an invalid syntax message is displayed.

Note

Relocating the monitor removes all memory mapper terms.

Enable Numeric Coprocessor?

The HP 64762/3 emulators contain an internal 8087 numeric coprocessor. You typically use the internal 8087 when target system hardware containing an 8087 is not yet developed. The internal 8087 allows you to execute and debug code that contains instructions for the 8087 coprocessor. When the target system hardware is developed, the internal 8087 is typically disabled and external DMA is enabled (see the "Enable DMA to/from Emulation Memory?" section that follows).

Yes When the internal 8087 numeric coprocessor is enabled, the emulator's internal 8087 coprocessor will respond to numeric opcodes in the instruction stream. An 8086/8088 RQ/GT line is taken by the 8087 when it is enabled. The RQ/GT line used is selectable using the "RQ/GT Line for Numeric Coprocessor?" configuration item below.

No When the internal 8087 is disabled, it will not operate and numeric opcodes will be ignored by the emulator. Both RQ/GT lines are available to the target system when the internal 8087 is disabled.

RQ/GT Line for Numeric Coprocessor?

If the internal 8087 numeric coprocessor is enabled, an 8086/8088 RQ/GT line allows the 8087 to acquire the local bus. The other RQ/GT line is available for target system use.

0 The internal 8087 uses the RQ/GT0 line. If the internal 8087 is enabled, the emulator will ignore this line from the target system.

1 The internal 8087 uses the RQ/GT1 line. If the internal 8087 is enabled, the emulator will ignore this line from the target system.

Select Numeric Processor as INTR Source?

When the internal 8087 is enabled, you can select either the target system or the internal 8087 to drive the 8086/88 INTR input.

If the internal 8087 is enabled but does not drive the 808X INTR input, use the 8087 INT auxiliary output line to drive the interrupt controller in the target system. See the "Auxiliary Output Lines" section in the "In-Circuit Emulation" chapter.

No When the target system is selected as the INTR source, the signal appearing on the INTR input of the user probe is applied to the emulation processor.

Yes When the internal 8087 is selected as the INTR source, the INT output of the internal 8087 numeric coprocessor drives the INTR input.

Interrupt Vector?

If you enable the internal 8087 and select it as the source for the emulation processor INTR input, the value specified for this configuration question will be jammed onto the data bus during interrupt acknowledge cycles.

The default emulator configuration specifies a value of 10H, which points to the numeric exception interrupt vector.

Enable DMA to/from Emulation Memory?

If external DMA access to emulation memory is enabled, target system devices that reside on the local 8086/8088 bus and conform to the 808X MAX mode bus timing can access emulation memory. An external 8087, for example, meets this requirement.

Yes If external DMA is enabled, you must connect the auxiliary output line TGT BUF DISABLE so that any target system devices that can drive the 808X addr/data bus are tristated when TGT BUF DISABLE is high. This is because any reads from emulation memory by the emulation processor or an external device will output data on the user probe. (The TGT BUF DISABLE signal goes active at the start of T2 in any bus cycle which accesses emulation memory; it goes inactive in T4.)

Enabling external DMA accesses of emulation memory will automatically configure the emulator to send flush queue status to the target while in background and while the 8086 is in MAX mode. You can subsequently reconfigure the emulator to send a NOP queue status while in background although this is not recommended. In particular, stepping through numeric instructions may not work properly if NOP queue status is selected.

No If external DMA is disabled, external devices cannot access emulation memory and will be unable to track the operation of emulation memory instructions. Then, the TGT BUF DISABLE line need not be used.

Storing an Emulator Configuration

The PC Interface lets you store a particular emulator configuration so that it may be reloaded later. The following information is saved in the emulator configuration.

- Emulator configuration items.
- Memory map.
- Break conditions.
- Trigger configuration.
- Window specifications.

To store the current emulator configuration, select:

```
Config Store
```

Enter the name of the file to which the emulator configuration will be saved.

Loading an Emulator Configuration

If you want to reload a previously stored emulator configuration, select:

```
Config Load
```

Enter the configuration file name and press **Enter**. The emulator will be reconfigured with the values specified in the configuration file.

Notes

5

Using the Emulator

Introduction

The "Getting Started" chapter shows you how to use the basic features of the 8086/8088 emulator. This chapter describes the more in-depth features of the emulator.

This chapter shows you how to:

- Make coordinated measurements.
- Store the contents of memory into absolute files.

This chapter also discusses register names and classes.

Making Coordinated Measurements

Coordinated measurements are measurements that use multiple emulators or analyzers. Coordinated measurements can be made between HP 64700-Series emulators, which communicate over the Coordinated Measurement Bus (CMB). You also can make Coordinated measurements between an emulator and another instrument connected to the BNC connector.

This section describes coordinated measurements made from the PC Interface, which involve the emulator. These types of coordinated measurements are:

- Running the emulator on reception of the CMB /EXECUTE signal.
- Using the analyzer trigger to break emulator execution into the monitor.

Note

You must use the background monitor to make coordinated measurements. Refer to the "Configuring the Emulator" chapter for more information on the emulation monitor.

Three signal lines on the CMB are active and serve the following functions:

/TRIGGER Active low. The analyzer trigger line on the CMB and on the BNC serve the same logical purpose. They provide a means for the analyzer to drive its trigger signal out of the system or for external trigger signals to arm the analyzer or break the emulator into its monitor.

READY Active high. This line is for synchronized, multi-emulator starts and stops. When CMB run control interaction is enabled, all emulators must break to background upon reception of a false READY signal and will not return to foreground until this line is true.

/EXECUTE Active low. This line serves as a global interrupt signal. Upon reception of an enabled /EXECUTE signal, each emulator is to interrupt whatever it is doing and execute a previously defined process, such as run the emulator or start a trace measurement.

Running the Emulator at /EXECUTE

Before you can specify that the emulator run upon receipt of the /EXECUTE signal, you must enable CMB interaction. To do this, select:

```
Config General
```

Use the arrow keys to move the cursor to the "Enable CMB Interaction? [n]" question, and type "y." Use the **End Enter** key sequence to exit from the lower right-hand field in the configuration display.

To specify that the emulator begin executing a program upon reception of the /EXECUTE signal, select:

```
Processor CMB Go
```

Now you may select either the current program counter ("Pc," in other words, the current CS:IP) or a specific address.

The command you enter is saved and is executed when the /EXECUTE signal becomes active. Also, you will see the message "ALERT: CMB execute; run started."

Using the Analyzer Trigger to Break into the Monitor

To break emulator execution into the monitor when the analyzer trigger condition is found, you must modify the trigger configuration. To access the trigger configuration, select:

```
Config Trigger
```

The trigger configuration display contains two diagrams, one for each internal TRIG1 and TRIG2 signal.

To use the internal TRIG1 signal to connect the analyzer trigger to the emulator break line, move the cursor to the highlighted "Analyzer" field in the TRIG1 portion of the display. Use the **Tab** key to select the "----->>" arrow, which shows that the analyzer is driving TRIG1. Next, move the cursor to the highlighted "Emulator" field. Use the **Tab** key to select the arrow pointing toward the emulator (<<-----).

This breaks emulator execution into the monitor when the TRIG1 signal is driven. Figure 5-1 shows the trigger configuration display.

Note

If your emulator does not have the optional external analyzer, or the external analyzer is aligned with the emulation analyzer (**A**nalaysis **S**ystem command), the "Timing" cross trigger option will not be displayed.

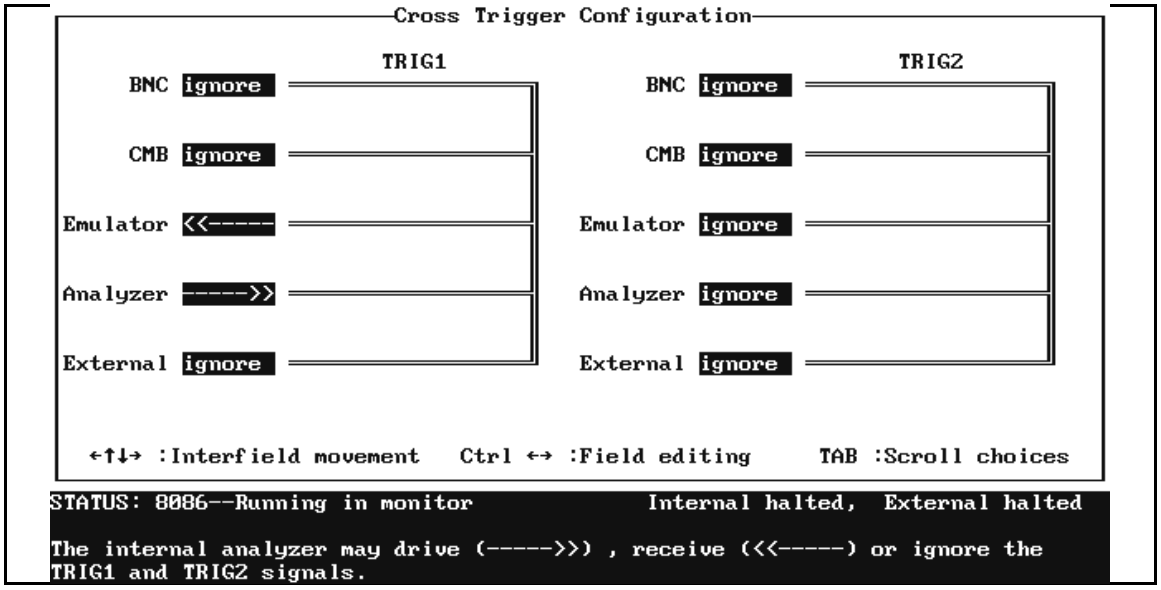


Figure 5-1. Cross Trigger Configuration

Storing Memory Contents to an Absolute File

The "Getting Started" chapter shows you how to load absolute files into emulation or target system memory. You also can store emulation or target system memory to an absolute file with the following command.

```
Memory Store
```

Note

You can name the absolute file with a total of 8 alphanumeric characters, and optionally, you can include an extension of up to 3 alphanumeric characters.

Caution 

The "Memory Store" command writes over an existing file if it has the same name that is specified in the command. You may want to verify beforehand that the specified filename does not already exist.

Register Names and Classes

The following table lists the register names and classes that may be used with the display/modify register commands.

<REG CLASS>	**<REG NAME>**	**Description:**
*	ah, al, ax, bh, bl, bx, ch, cl, cx, dh, dl, dx, bp, si, di, ds, es, ss, sp, ip, cs, fl	All Basic Registers
gen	ax, bx, cx, dx	General Registers
seg	ds, es, ss, cs	Segment Registers
ptr	bx, bp, si, di, ds, es	Pointer Registers
ncp (Internal 8087 numeric coprocessor registers)	ctrl stat iptr optr opc tag st[0], st[1], st[2], st[3], st[4], st[5], st[6], st[7]	Control Word Status Word Exception Pointer Instruction Address Exception Pointer Operand Address Exception Pointer Instruction Opcode Tag Word Register Stack Registers

Notes

A

Foreground Monitor Description

Introduction

The monitor program is the interface between the emulation system controller and the target system. The emulation system controller uses its own microprocessor to accept and execute emulation, system, and analysis commands. The emulation processor (the 8086 or 8088) executes the monitor program.

The monitor program makes possible emulation commands that access target system resources. (The only way to access target system resources is through the emulation processor.) For example, when you enter a command to modify target system memory, the monitor program executes instructions to write the new values to target system memory.

When the emulation system controller recognizes that an emulation command needs to access target system resources, it writes a command code to a communications area and breaks into the monitor. The monitor reads this command (and any associated parameters) from the communications area and executes the appropriate 8086/88 instructions to access these target system resources.

Breaks into the Monitor

When a break condition occurs, the emulation processor's NMI is used to enter the monitor. The IP, CS, and flag information, normally saved on the stack during an NMI, are jammed into monitor program storage locations. (The background portion of the monitor makes no assumptions about the existence of a stack.)

Emulator Modes (Foreground, Background)

The two emulator modes are foreground and background.

Foreground

Foreground is the mode in which all emulation processor cycles appear on the emulation probe, and the emulator executes as if it were a real 8086/8088 microprocessor. In foreground mode, the emulation microprocessor typically executes out of target system or emulation memory. (But, it may operate out of memory reserved for the monitor when a foreground monitor is selected.)

Background

Background is the mode in which instruction execution does not appear normally on the emulator probe. Background cycles may be visible (on the emulator probe), or hidden from the target system. But, when background cycles are visible, they appear as reads. When background cycles are hidden, the emulator appears to the target system to be in a suspended state. In background mode, the emulation microprocessor executes out of memory reserved for the monitor.

Modes in Which the Foreground Monitor Operates

The foreground monitor operates in both background and foreground. The difference between the foreground monitor and the background monitor is that when the background monitor is used, all monitor functions execute in background. When the foreground monitor is used, the monitor functions execute in foreground. Part of the foreground monitor executes in background because emulator breaks always put the emulator in the background mode. The portion of the foreground monitor that executes in background sets up the IP, CS, and flags for return to foreground (where execution of monitor functions takes place).

Other Background Modes

The emulator may be operated in additional modes while in background. These additional emulator modes can:

- Present unmodified cycles (real status) to the target system (allows the emulator to perform writes to target memory while in background).
- Allow background writes to target system memory.

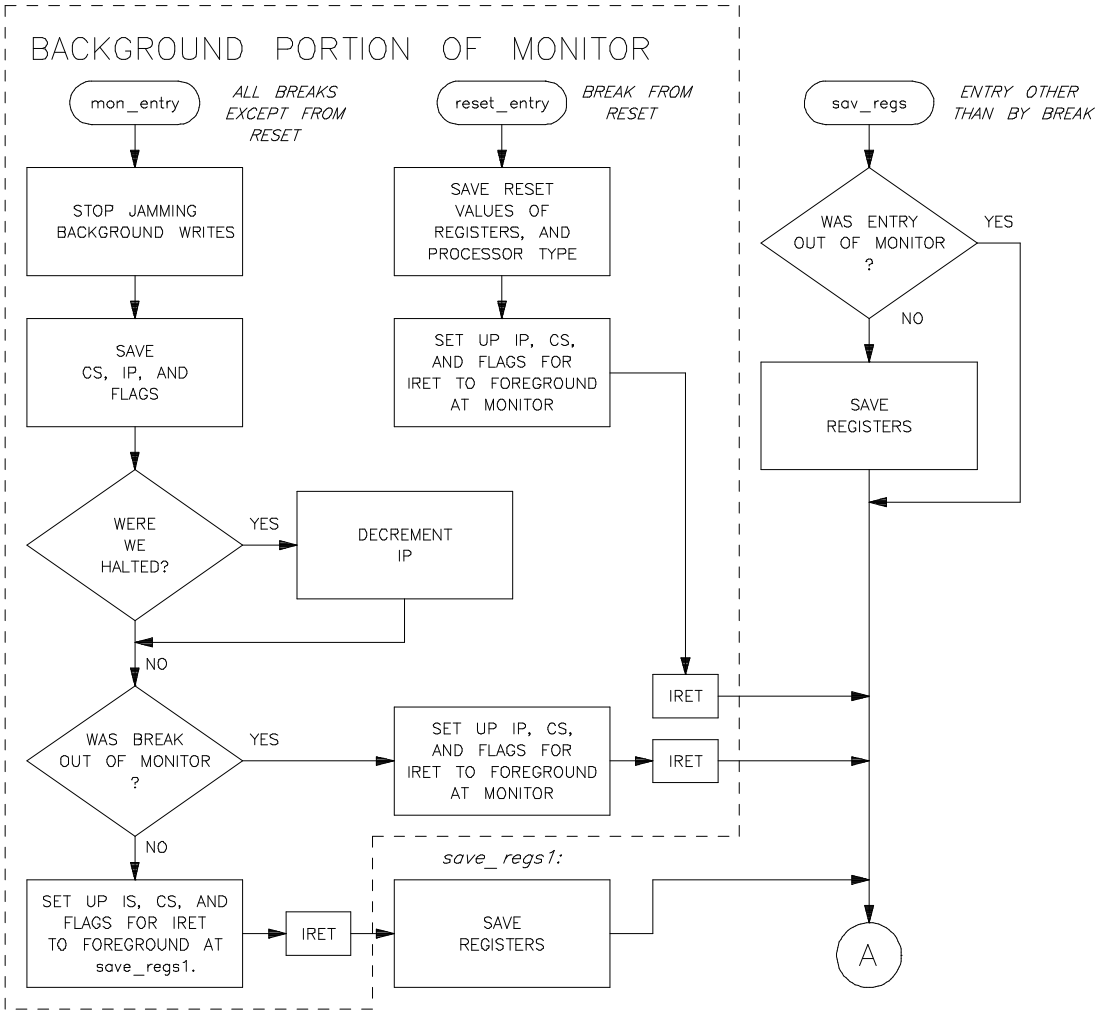
- Allow background reads from target system memory.

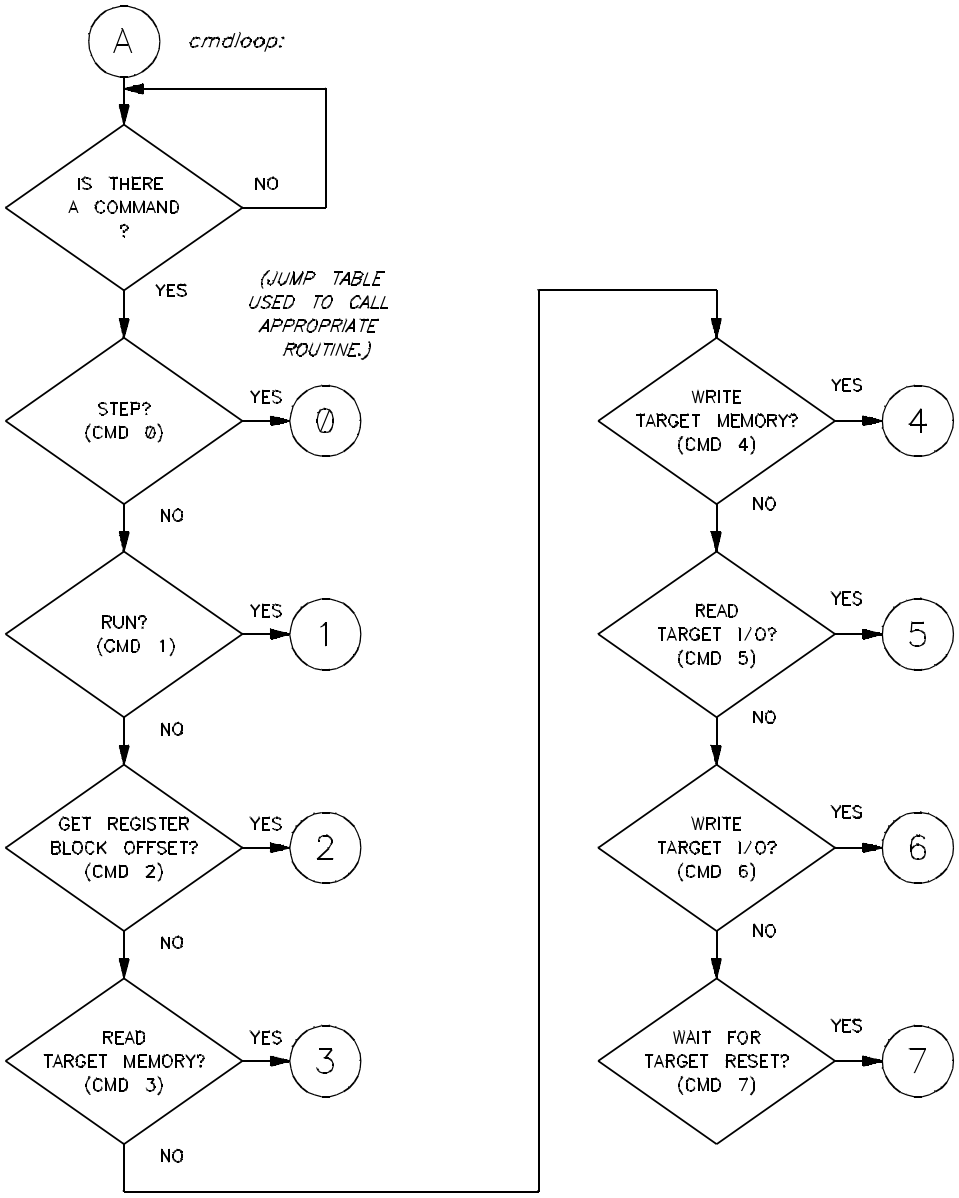
These additional modes are set and reset by opcode fetches to special locations in the monitor area (40H through 7FH). These modes (and the instructions which set and reset them) are documented in the foreground monitor listing. The portion of the foreground monitor that executes in background does not use any of these additional modes.

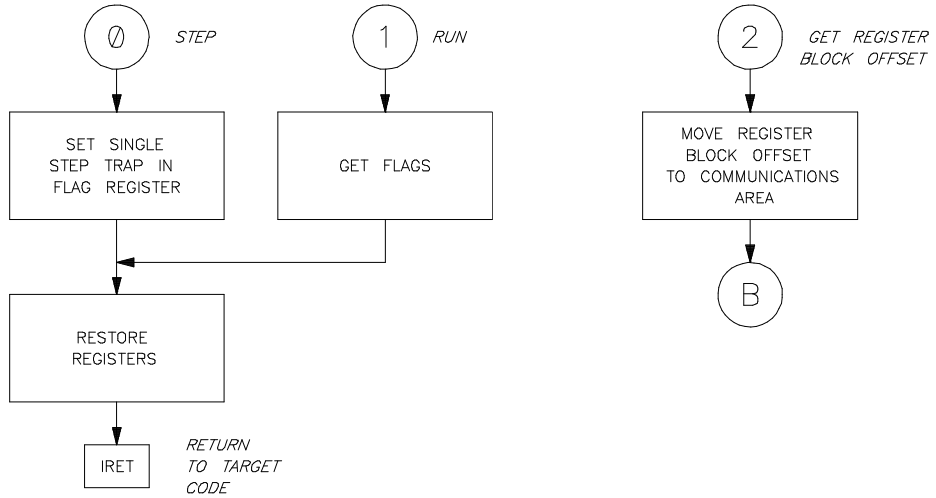
Listing

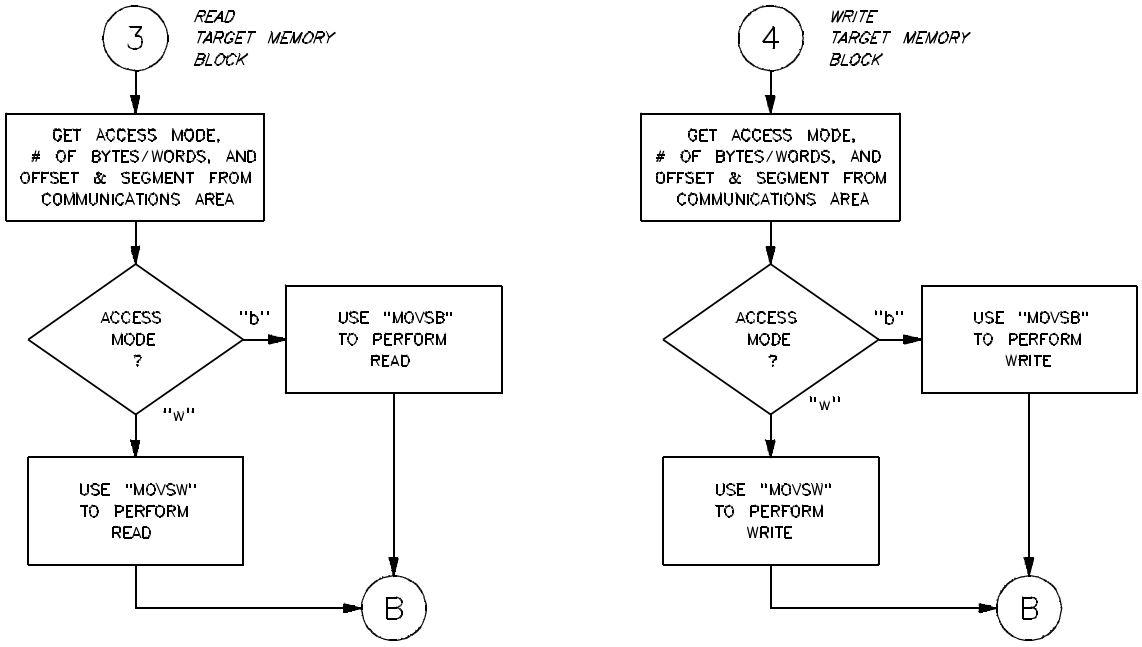
The foreground monitor is resident in the emulator, and it may be selected without having to load any code. But, the foreground monitor comes with the emulator on a floppy disk so that you may customize it, if necessary. Refer to the floppy disk's foreground monitor source file for the latest program listing.

Flowchart

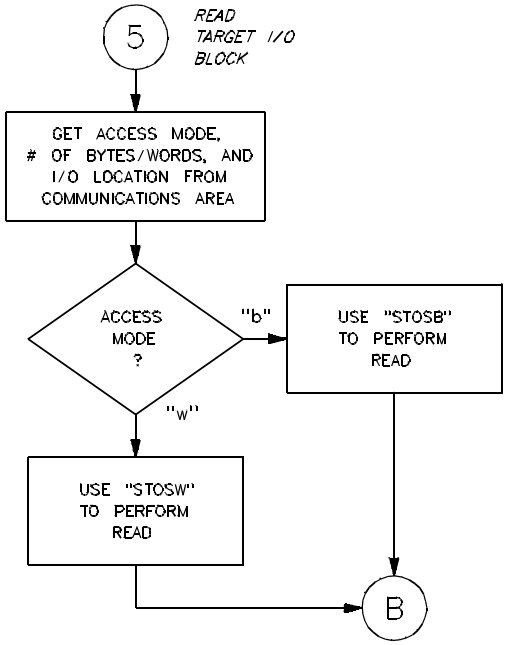

BACKGROUND PORTION OF MONITOR
mon_entry
ALL BREAKS EXCEPT FROM RESET
reset_entry
BREAK FROM RESET
sav_regs
ENTRY OTHER THAN BY BREAK
STOP JAMMING BACKGROUND WRITES
SAVE RESET VALUES OF REGISTERS, AND PROCESSOR TYPE
WAS ENTRY OUT OF MONITOR ?
YES
NO
SAVE CS, IP, AND FLAGS
SET UP IP, CS, AND FLAGS FOR IRET TO FOREGROUND AT MONITOR
SAVE REGISTERS
WERE WE HALTED?
YES
DECREMENT IP
NO
IRET
WAS BREAK OUT OF MONITOR ?
YES
SET UP IP, CS, AND FLAGS FOR IRET TO FOREGROUND AT MONITOR
IRET
NO
save_regs1:
SET UP IS, CS, AND FLAGS FOR IRET TO FOREGROUND AT save_regs1.
IRET
SAVE REGISTERS
A

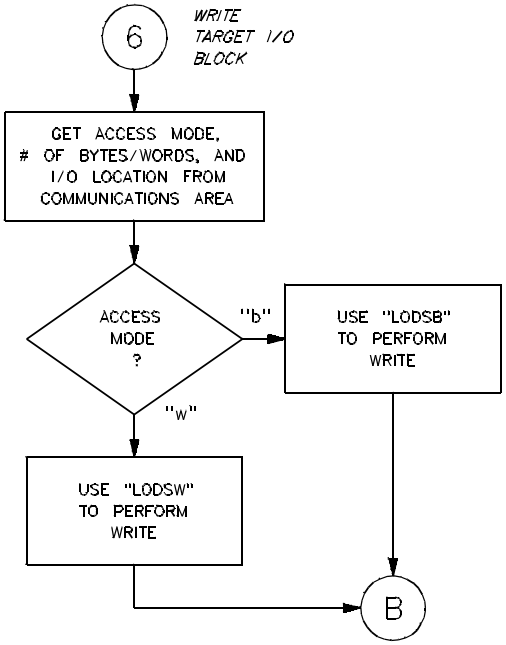

A
cmdloop:
IS THERE A COMMAND ?
NO
YES
(JUMP TABLE USED TO CALL APPROPRIATE ROUTINE.)
STEP? (CMD 0)
YES
0
NO
RUN? (CMD 1)
YES
1
NO
GET REGISTER BLOCK OFFSET? (CMD 2)
YES
2
NO
READ TARGET MEMORY? (CMD 3)
YES
3
NO
WRITE TARGET MEMORY? (CMD 4)
YES
4
NO
READ TARGET I/O? (CMD 5)
YES
5
NO
WRITE TARGET I/O? (CMD 6)
YES
6
NO
WAIT FOR TARGET RESET? (CMD 7)
YES
7

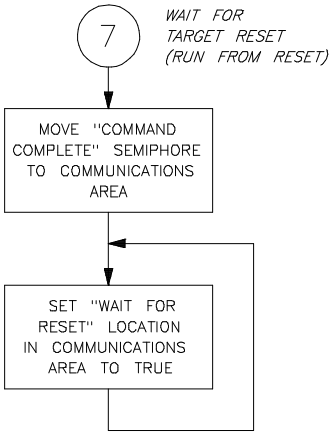


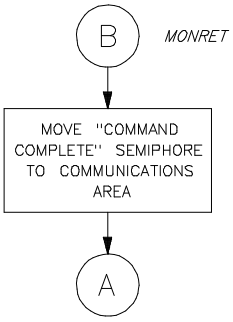


A-6 Foreground Monitor Description


5
READ
TARGET I/O
BLOCK
GET ACCESS MODE,
OF BYTES/WORDS, AND
I/O LOCATION FROM
COMMUNICATIONS AREA
ACCESS
MODE
?
"b"
USE "STOSB"
TO PERFORM
READ
"w"
USE "STOSW"
TO PERFORM
READ
B


6
WRITE
TARGET I/O
BLOCK
GET ACCESS MODE,
OF BYTES/WORDS, AND
I/O LOCATION FROM
COMMUNICATIONS AREA
ACCESS
MODE
?
"b"
USE "LODSB"
TO PERFORM
WRITE
"w"
USE "LODSW"
TO PERFORM
WRITE
B


7
WAIT FOR
TARGET RESET
(RUN FROM RESET)
MOVE "COMMAND
COMPLETE" SEMIPHORE
TO COMMUNICATIONS
AREA
SET "WAIT FOR
RESET" LOCATION
IN COMMUNICATIONS
AREA TO TRUE


B
MONRET
MOVE "COMMAND
COMPLETE" SEMIPHORE
TO COMMUNICATIONS
AREA
A

Notes

File Format Readers

Introduction

Two *file format readers* come with the 8086/88 PC Interface. The OMF Reader reads Intel OMF86 files. The HP64000 reader reads HP64000 absolute file format. Both readers convert the absolute code and symbol information obtained from the OMF or HP64000 files into two files that are usable with the HP 64762/3 Emulator. This means that you can use available language tools to create OMF86 or HP64000 absolute files, then load those files into the emulator using the 8086/88 PC Interface.

The readers can operate from within the PC Interface or as a separate process. You may need to execute the reader as a separate process if there is not enough memory on your personal computer to operate the PC Interface and reader simultaneously. You also can operate the reader as part of a "make file."

Note

In this appendix, both the HP64000 Reader and the OMF Reader are called *reader* unless the information is specific to one of those readers.

What the Reader Does

The OMF reader takes any OMF86 absolute file in the form "<file>.<ext>," and produces two new files, an "absolute" file and an ASCII symbol file, that will be used by the 8086/88 PC Interface. These new files are named: "<file>.hpa" and "<file>.hps." The HP64000 reader produces the same files, but obtains information from the .X file (absolute code), .L file (linker symbols) and .A file (assembler symbols).

The Absolute File

The reader creates an absolute file (<file>.hpa). The absolute file is a binary memory image optimized for efficient downloading into the emulator.

The ASCII Symbol File

The ASCII symbol file (<file>.hps) produced by the reader contains global symbols, module names, local symbols, and, when using applicable development tools such as a "C" compiler, program line numbers. Local symbols evaluate to a fixed (static, not stack relative) address.

Note 

You must use the required options for your specific language tools to include symbolic ("debug") information in the OMF absolute file or HP64000 symbol files. The reader will only convert symbol information that is present in the file.

The symbol file contains symbol and address information in the following form:

```
 module_name1
 module_name2
 ...
 module_nameN
global_symbol1  00100:01234
global_symbol2  00100:05678
...
global_symbolN  00100:0ABCD
|module_name|# 1234              00200:00872
|module_name|local_symbol1   00200:00653
|module_name|local_symbol2   00200:00872
...
|module_name|local_symbolN   00200:00986
```

Each symbol is sorted alphabetically in the order: module names, global symbols, and local symbols.

Line numbers will appear similar to a local symbol except that "local_symbolX" will be replaced by "#NNNNN" where NNNNN is a five digit decimal line number. For the HP64000 format, the addresses

associated with global and local symbols are in segment:offset format (not physical addresses).

Note

The PC Interface displays line number symbols in brackets. Therefore, the symbol “MODNAME:line 345” will be displayed as “MODNAME:[345]” in mnemonic memory and trace list displays.

The space preceding module names is required. Although formatted for readability here, a single tab separates symbol and address.

The local symbols obey static scoping rules. This means that to access a variable named “COUNT” in a function named “Foo” in a source file module named “main.c,” you would enter “main.c:Foo.COUNT.” See table B-1.

Table B-1. How to Access Variables

Module Name	Function Name	Variable Name	You Enter:
main.c	Foo	COUNT	main.c:Foo.COUNT
main.c	bar	COUNT	main.c:bar.COUNT
main.c	line number 23		main.c: line 23

You enter line number symbols by typing the following on one line in the order shown:

module name
colon (:)
space
the word “line”
space
the decimal line number

For example:

```
main.c: line 23
```

Location of the Reader Programs

The readers are installed in the directory named \hp64700\bin by default, with the PC Interface. This directory must be in the environment variable PATH for the readers and PC Interface to operate properly. This is usually defined in the "\autoexec.bat" file.

The following examples assume that you have "\hp64000\bin" included in your PATH variable. If not, you must supply the directory name when executing a reader program.

Using a Reader from MS-DOS

OMF Reader

The command name for the OMF Reader is **RDOMF86.EXE**. To execute the Reader from the command line, enter:

```
RDOMF86 [-q] [-u] [-m] <filename>
```

-q	specifies the "quiet" mode. This option suppresses the display of messages.
-u	defeats removal of a leading underscore in the symbol name (for example, "_symbol"). When used, a symbol name containing a leading underscore will be left alone.
-m	removes duplicate module names generated by some construction tools. Some tools enclose all functions and variables in a module within a block (or function) whose name is the same as that of the module (or source file). When you use this option, the Reader ignores the first enclosing block in a module if its name matches the module name.
<filename>	is the name of the file containing the Intel OMF absolute program. You can include an extension in the file name.

The following command will create the files “TESTPROG.HPA” and “TESTPROG.HPS.”

```
RDOMF86 TESTPROG.ABS
```

HP64000 Reader

The command name for the HP 64000 Reader is **RHP64000.EXE**. To execute the Reader from the command line, for example, enter:

```
RHP64000 [-q] <filename>
```

-q	This option specifies the “quiet” mode, and suppresses the display of messages.
<filename>	This represents the name of the HP 64000 linker symbol file (file.L) for the absolute file to be loaded.

The following command will create the files “TESTPROG.HPA” and “TESTPROG.HPS”

```
RHP64000 TESTPROG.L
```

Using a Reader from the PC Interface

The 8086/88 PC Interface has a file format option under the “**M**emory **L**oad” command. After you select OMF86 or HP64000 as the file format, the appropriate reader will operate on the file you specify. After this completes successfully, the 8086/88 PC Interface will accept the absolute and symbol files produced by the Reader.

To use the Reader from the PC Interface:

1. Start up the 8086/88 PC Interface.
2. Map memory (if it isn’t already done by your configuration file). Then select “**M**emory **L**oad.” The memory load menu will appear.
3.
4.

5. Specify the file format as “OMF86” (default) or “HP64000.” You can use the **Tab** and **Shift-Tab** keys to cycle through the choices.

6. Select a memory type. You can use the **Tab** and **Shift-Tab** keys to cycle through the choices:
   - **emulation** (loads address ranges mapped to emulation memory only)
   - **target** (loads address ranges mapped to target memory only)
   - **both** (loads all address ranges)

7. Specify Y or N for “Force Absolute file Read?” If you select Y, the OMF or HP64000 file is always read, even if it is older than the .HPA and .HPS files with the same name.

8. For the Intel OMF format only:
   - Select whether leading underscores should be removed from symbol names found in the absolute file.
   - Select whether duplicate module names found in the absolute should be removed.

9. Specify a filename in the appropriate format.

   - For the OMF format, select an OMF absolute file (“TESTFILE.OMF,” for example). The file extension can be something other than “.OMF,” but “.HPA,” “.HPT,” or “.HPS” cannot be used.

   - For the HP64000 format, select the linker symbol file (“TESTFILE.L,” for example). “.HPA,” “.HPT,” or “.HPS” cannot be used.

Note

The “<filename>.HPT” file is a temporary file used by the reader to process the symbols.

Using the file that you specify (TESTFILE.OMF, for example), the PC Interface does the following:

- It checks to see if two files with the same base name and extensions .HPS and .HPA already exist (for example, TESTFILE.HPS and TESTFILE.HPA).
- If TESTFILE.HPS and TESTFILE.HPA don't exist, the PC Interface starts the reader. The new absolute file, TESTFILE.HPA, is then loaded into the emulator.
- If TESTFILE.HPS and TESTFILE.HPA already exist but the create dates and times are earlier than the .OMF file creation date/time, the PC Interface has the reader recreate the files. The new absolute file, TESTFILE.HPA, is then loaded into the emulator.
- If TESTFILE.HPS and TESTFILE.HPA already exist, but the creation dates and times are later than those for the .OMF file, the PC Interface does not call the reader. The current absolute file, TESTFILE.HPA, is then loaded into the emulator.

Note

Date/time checking is done only within the PC Interface. When you run a reader at the MS-DOS command line prompt, it will always update the absolute and symbol files.

When a reader operates on a file, a status message will be displayed indicating that the reader is operating and the file type that is being read. When the reader is finished, another message will be displayed showing that the absolute file is being loaded.

The PC Interface executes the Reader with the "**-q**" (quiet) option by default.

If the Reader Won't Run

If your program is very large, the PC Interface may run out of memory while attempting to create the database file. You need to exit the PC Interface. Then execute the program at the MS-DOS command prompt to create the files.

Including the Reader in a Make File

You may wish to incorporate the “RDOMF86” or “RHP64000” process as the last step in your “make file,” or as a step in your code construction process. Then you won’t need to exit the PC Interface due to memory space limitations.

Index

C

Notes