HP 64762/3

# 8086/8088 Emulators Softkey Interface

## User's Guide

HEWLETT PACKARD

HP Part No. 64762-97003
Printed in U.S.A.
August, 1992

Edition 3

# 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 Company**
**P.O. Box 2197**
**1900 Garden of the Gods Road**
**Colorado Springs, CO 80901-2197, 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 | 64762-90902, February 1989 E0289 |
| Edition 2 | 64762-97001, July 1990 |
| **Edition 3** | **64762-97003, August 1992** |

# Using this Manual

This manual will show you how to use the HP 64762/3 (8086/88) emulators with the Softkey 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, and allowing the target system to insert wait states.

This manual does not:

- Show you how to use every Softkey Interface command and option. The *Softkey Interface Reference* describes the interface command syntax in detail.

## Organization

**Chapter 1** “Introduction to the 8086/8088 Emulator.” This chapter introduces 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, using a sample program. The chapter describes the sample program and how to:

- load programs into the emulator
- display and modify memory
- display registers
- step through programs
- run programs
- set software breakpoints
- use the analyzer

This chapter also includes information about the OMF-86 file format and symbol tree, when not to enable software breakpoints, and behavior of the processor while single-stepping.

**Chapter 3** "In-Circuit Emulation." This chapter shows you how to install the emulator probe into a target system and discusses other "in-circuit" emulation topics. It also shows how to connect SYS RESET to a target system.

**Chapter 4** "Configuring the Emulator." This chapter describes the emulation configuration options. These options include:

- restricting the emulator to real-time execution
- selecting a target system clock source
- allowing background cycles to be seen by the target system
- allowing the target system to insert wait states
- selecting foreground or background emulation monitors
- adding code to the background monitor
- allowing DMA accesses to emulation memory
- selecting the internal 8087 numeric coprocessor

**Chapter 5** "Using the Emulator." This chapter describes emulation topics not covered in the "Getting Started" chapter. It explains how to save memory to absolute files, and how to use the Terminal Interface features from within the Softkey Interface.

**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 emulation software so that you may customize it, if necessary.

## Conventions

Example commands throughout the manual use the following conventions:

| | |
|---|---|
| **bold** | Commands, options, and parts of command syntax. |
| ***bold italic*** | Commands, options, and parts of command syntax which may be entered by pressing softkeys. |
| normal | User specified parts of a command. |
| $ | Represents the HP-UX prompt. Commands which follow the "$" are entered at the HP-UX prompt. |
| <RETURN> | The carriage return key. |

# Notes

# Contents

**A Foreground Monitor Description**

# Illustrations

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, while giving information about the operation of the processor. The emulator allows you to control target system execution. You 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 the emulator features. The chapters that follow show you how to use these features.

### 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. You can select which RQ/GT pin the internal 8087 will use (if enabled). Additionally, you can select the internal 8087 as the driver of the 8086/88 INTR input, and specify the internal interrupt vector (if the internal 8087 drives the 8086/88 INTR input).

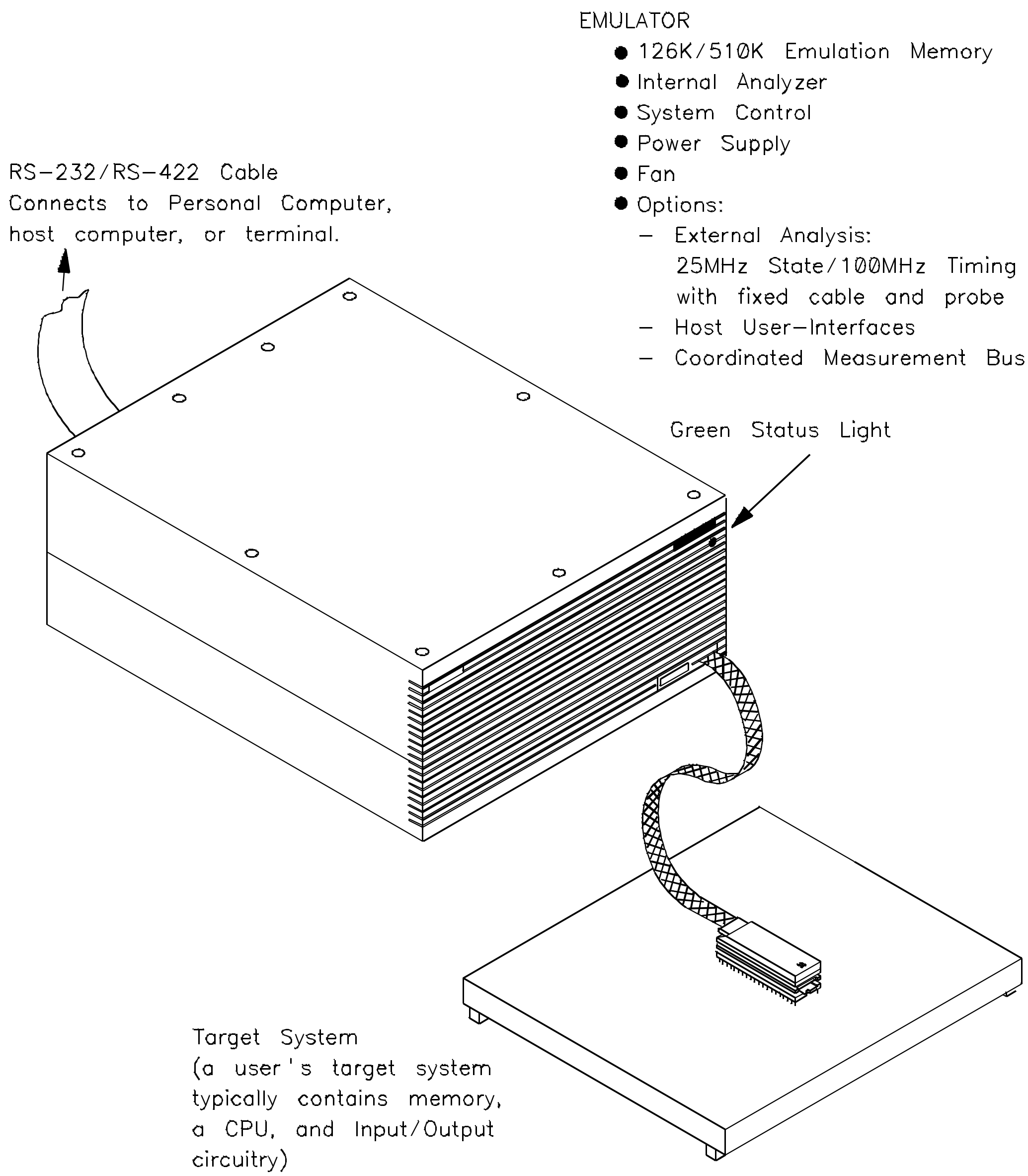



**Figure 1-1. The HP 64762/3 Emulator for the 8086/8088**

## 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 which 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, emulation ROM, target system RAM, target system ROM, or as guarded memory. The emulator issues an error message for guarded memory accesses. You can configure the emulator so that writes to memory defined as ROM cause a break in emulator execution (into the emulation monitor program).

## External DMA Access to Emulation Memory

You can enable DMA access to 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 information. The emulation analyzer captures 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 Softkey 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 in scientific decimal notation.

## Single-Step

You can direct the emulation processor to execute one or more instructions.

## Breakpoints

You can set up the emulator/analyzer interaction to break emulator execution into the background monitor when the analyzer finds a specific state.

You also can define software breakpoints in your program. The emulator uses the 8086/88 single-byte interrupt facility for 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 to from your program into the monitor, the emulator replaces the original opcode.

## Reset Support

You can reset the emulator from the emulation system. 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;
- present cycles to, or hide cycles from, the target system when executing in background;
- present either a FLUSH or a NOP queue status to the target system while in background (and in the maximum mode);
- 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*. This emulator mode suspends foreground operation so that the emulation processor can access target system resources. The background monitor does not occupy processor address space.

## Real-Time Execution

Real-time operation is 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 restricted to real-time, emulator commands which display/modify registers, or display/modify target system memory or I/O are not allowed.

## Notes

# 2

# Getting Started

## Introduction

This chapter will lead you through a basic tutorial that shows how to use the HP 64762 and HP 64763 emulators (for the 8086 and 8088 microprocessors) with the Softkey Interface.

This chapter will:

- Tell you what to do before you use the emulator in the tutorial.
- Describe the sample program used for this chapter's examples.

This chapter will show you how to:

- Start the Softkey Interface.
- Load programs into emulation and target system memory.
- Enter emulation commands to view execution of the sample program.

## Before You Begin

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

1. Connect the emulator to your computer. The *HP 64700-Series Emulators Softkey Interface Installation Notice* and the *HP 64700-Series Emulators Hardware Installation and Configuration* manual show how to do this.

2. Install the Softkey Interface software on your computer. Refer to the *HP 64700 Series Emulators Softkey Interface Installation Notice* for instructions on installing software.

3. In addition, you should read and understand the concepts of emulation presented in the *HP 64700 System Overview Manual*. The *System Overview* also covers HP 64700 system architecture. A brief understanding of these concepts may help you avoid questions later.

   You should read the *Softkey Interface Reference* manual to learn general operation of the Softkey Interface. This manual contains information specific to the 8086 and 8088 emulators.

## A Look at the Sample Program

Figure 2-1 lists the sample program used in this chapter. The program is a primitive command interpreter. The sample program is shipped with the Softkey Interface and may be copied from the following location.

**/usr/hp64000/demo/emul/hp64762/cmd_rds.S** (8086)
**/usr/hp64000/demo/emul/hp64763/cmd_rds.S** (8088)

### Data Declarations

The DATA area 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.

### 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).

```
HEWLETT-PACKARD: 8086 Assembler
FILE: /users/guest/dir86/cmd_rds.S
LOCATION OBJECT CODE LINE     SOURCE LINE

                        1 "8086"
                        2                GLB     Msgs,Init,Cmd_Input,Msg_Dest
                        3                DATA
    0000                4 Msgs
    0000 436F6D6D61     5 Msg_A          DB      "Command A entered "
    0005 6E64204120
    000A 656E746572
    000F 656420
    0012 456E746572     6 Msg_B          DB      "Entered B command "
    0017 6564204220
    001C 636F6D6D61
    0021 6E6420
    0024 496E76616C     7 Msg_I          DB      "Invalid Command "
    0029 696420436F
    002E 6D6D616E64
    0033 20
    0034                8 End_Msgs
                        9
                       10                ORG     0FFFF0000H
    0000 EA00000000    11                JMP     FAR PTR Init
                       12
                       13                PROG
                       14                ASSUME  DS:DATA,ES:COMN
                       15 ****************************************************
                       16 * The following instructions initialize segment
                       17 * registers and set up the stack pointer.
                       18 ****************************************************
    0000 B80000        19 Init           MOV     AX,SEG Msg_A
    0003 8ED8          20                MOV     DS,AX
    0005 B80000        21                MOV     AX,SEG Cmd_Input
    0008 8EC0          22                MOV     ES,AX
    000A 8ED0          23                MOV     SS,AX
    000C BC00F9        24                MOV     SP,OFFSET Stk
                       25 ****************************************************
                       26 * Clear previous command.
                       27 ****************************************************
    000F 26C6060000    28 Read_Cmd       MOV     Cmd_Input,#0
    0014 0090
                       29 ****************************************************
                       30 * Read command input byte.  If no command has been
                       31 * entered, continue to scan for command input.
                       32 ****************************************************
    0016 26A00000      33 Scan           MOV     AL,Cmd_Input
    001A 3C00          34                CMP     AL,#0
    001C 74F8          35                JE      Scan
                       36 ****************************************************
                       37 * A command has been entered.  Check if it is
                       38 * command A, command B, or invalid.
                       39 ****************************************************
    001E 3C41          40 Exe_Cmd        CMP     AL,#41H
    0020 7407          41                JE      Cmd_A
```

**Figure 2-1. Sample Program Listing**

```
0022 3C42                        CMP     AL,#42H
0024 740C                        JE      Cmd_B
0026 E91200                      JMP     Cmd_I
           ****************************************************
           * Command A is entered.  CX = the number of bytes in
           * message A.  SI = location of the message.  Jump to
           * the routine which writes the messages.
           ****************************************************
0029 B91200        Cmd_A         MOV     CX,#Msg_B-Msg_A
002C BE0000                      MOV     SI,OFFSET Msg_A
002F E90F00                      JMP     Write_Msg
           ****************************************************
           * Command B is entered.
           ****************************************************
0032 B91200        Cmd_B         MOV     CX,#Msg_I-Msg_B
0035 BE0012                      MOV     SI,OFFSET Msg_B
0038 E90600                      JMP     Write_Msg
           ****************************************************
           * An invalid command is entered.
           ****************************************************
003B B91000        Cmd_I         MOV     CX,#End_Msgs-Msg_I
003E BE0024                      MOV     SI,OFFSET Msg_I
           ****************************************************
           * Message is written to the destination.
           ****************************************************
0041 8D3E0001      Write_Msg     LEA     DI,Msg_Dest
0045 F3A4                    REP MOVSB
           ****************************************************
           * The rest of the destination area is filled
           * with zeros.
           ****************************************************
0047 C60500        Fill_Dest     MOV     BYTE PTR [DI],#0
004A 47                          INC     DI
004B 81FF0021                    CMP     DI,#Msg_Dest+20H
004F 75F6                        JNE     Fill_Dest
           ****************************************************
           * Go back and scan for next command.
           ****************************************************
0051 EBBC                        JMP     Read_Cmd

                                 COMN
           ****************************************************
           * Command input byte.
           ****************************************************
0000               Cmd_Input     DBS     1
           ****************************************************
           * Destination of the command messages.
           ****************************************************
0001               Msg_Dest      DDS     3EH
00F9               Stk           DWS     1        ; Stack area.
                             END     Init

Errors=    0
```

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

### Processing Commands

When a command is entered, the instructions from **Exe_Cmd** to **Cmd_A** decide 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” nor “B,” it is an invalid command, 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 display message length and register SI with the appropriate message’s starting location. Then, execution transfers to **Write_Msg**, which writes the 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 COMN area declares memory storage for the command input byte, the destination area, and the stack area.

## Assembling the Sample Program

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

```
$ asm -oe cmd_rds.S > cmd_rds.O  <RETURN>
```

The assembly process creates the assembler listing (cmd_rds.O) and two other files.

- The “cmd_rds.R” file is the relocatable file. You link relocatable files to form the absolute file, which you load into the emulator.

- The “cmd_rds.A” file is the assembler symbol file. It contains information on the local symbols in the sample program.

## Linking the Sample Program

Use the following command to generate the absolute file:

```
$ lnk -o > cmd_rds.M <RETURN>
object files  cmd_rds.R <RETURN>
library files  <RETURN>
Load addresses: PROG,DATA,COMN
400h,500h,600h <RETURN>
more files (y or n)  n <RETURN>
absolute file name  cmd_rds.X <RETURN>
```

Link creates the linker load map listing (cmd_rds.M) and three other files.

- The “cmd_rds.x” file is the file that contains the absolute code to be loaded into the emulator.

- The “cmd_rds.L” file is the linker symbol file. It contains information on the global symbols in the sample program and the relocatable files that combine to form the absolute file.

- The “cmd_rds.K” file is the linker command file. It contains the answers to the questions asked with the above **lnk** command. You can specify the linker command file in the **lnk** command to avoid reanswering the questions shown above (for example, “lnk -o cmd_rds.K > cmd_rds.M”).

**Note**

You no longer need to use the **-h** option when linking programs generated by HP-AxLS language tools. The emulator will work directly with symbolic information contained in the new .x files. The HP-OMF file (.X) is no longer needed. The linker generates the new .x files by default. See the *SRU User's Guide* for more information.

## Entering the Softkey Interface

If you have installed your emulator and Softkey Interface software as directed in the *HP 64700 Series Emulators Softkey Interface Installation Notice*, you are ready to enter the interface. The Softkey Interface can be entered through the **pmon** User Interface Software or from the HP-UX shell.

- If you have used previous HP 64200-Series emulators, you may be familiar with the **pmon**, **msinit**, and **msconfig** method of entering the emulation interface.
- If you want to run the Softkey Interface in multiple windows, you must enter from the HP-UX shell using the **emul700** command. Refer to the *Softkey Interface Reference* manual for more information on running in multiple windows.

### From the "pmon" User Interface

If your PATH environment variable includes /usr/hp64000/bin, you can enter the **pmon** User Interface as follows:

$ **pmon** <RETURN>

If you have not already created a measurement system for the emulator, you can do so with the following commands. First, you must initialize the measurement system:

***MEAS_SYS msinit*** <RETURN>

When this completes, enter the configuration interface with the following command.

***msconfig*** <RETURN>

To define a measurement system for the 8086 emulator, enter:

***make_sys*** em86 <RETURN>

Now, to add the emulator to the measurement system, enter:

***add*** <module_number> ***naming_it*** i8086 <RETURN>

Enter the following command to exit the measurement system configuration interface.

***end*** <RETURN>

If the measurement system and emulation module are named “em86” and “i8086” as above, you can enter the emulation system with the following command:

***em86 default i8086*** <RETURN>

If this command is successful, you will see a display similar to figure 2-2. The status message shows that the default configuration file was loaded. If the command is not successful, you will be given an error message and returned to the **pmon** User Interface. The *Softkey Interface Reference* manual documents error messages.

For more information on creating measurement systems, refer to the *Softkey Interface Reference* manual.

## From the HP-UX Shell

If your PATH environment variable includes /usr/hp64000/bin, you also can enter the Softkey Interface with the following command.

$ **emul700** <emul_name> <RETURN>

The “emul_name” in the command above is the logical emulator name given in the HP 64700 emulator device table (/usr/hp64000/etc/64700tab).

If this command is successful, you will see a display similar to figure 2-2. The status message shows that the default configuration file was loaded. If the command is not successful, you will be given an error message and returned to the HP-UX prompt. The *Softkey Interface Reference* manual documents error messages.

```
                    HP64762A004 A.02.00 18July90
                   8086 EMULATION SERIES 64700

               A Hewlett-Packard Software Product
               Copyright Hewlett-Packard Co. 1990

  All Rights Reserved. Reproduction, adaptation, or translation without prior
  written  permission  is prohibited, except as allowed under copyright laws.

                    RESTRICTED RIGHTS LEGEND

     Use , duplication , or disclosure  by the  Government is  subject to
     restrictions as set forth in subparagraph (c) (1) (II) of the Rights
     in Technical Data and Computer Software clause at  DFARS 52.227-7013.
     HEWLETT-PACKARD Company , 3000 Hanover St. , Palo Alto, CA 94304-1181


STATUS:   Loaded configuration file_____________________________________...R....



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

**Figure 2-2. Softkey Interface Display**

### Using the Default Configuration

The following examples use the default emulator configuration. The address ranges 0 through 1EFFFH and 0FFC00H through 0FFFFFH map to emulation RAM. The background monitor is selected, and software breakpoints are disabled.

## On-Line Help

There are two ways to get on-line help in the Softkey Interface. The first uses the Softkey Interface help facility. The second method allows you to access the firmware resident Terminal Interface on-line help information.

### Softkey Driven Help

To access the Softkey Interface on-line help information, type either “help” or “?” on the command line. You will notice a new set of softkeys. By pressing one of these softkeys and < RETURN> , you can display information on that topic.

For example, you enter the following command to access "system command" help information.

? ***system_commands*** <RETURN>

```
---SYSTEM COMMANDS---

?                          displays the possible help files
help                       displays the possible help files
!                          fork a shell (specified by  shell variable SH)
!<shell cmd>               fork a shell and execute a shell command
cd <directory>             change the working directory
pwd                        print the working directory
cws <SYMB>                 change the working symbol - the working symbol also
                              gets updated when displaying local symbols and
                              displaying memory mnemonic
pws                        print the working symbol
<FILE> p1 p2 p3 ...        execute a command file passing parameters p1, p2, p3

log_commands to <FILE>     logs the next sequence of commands to file <FILE>
log_commands off           discontinue logging commands
name_of_module             get the "logical" name of this module (see 64700tab)
set <ENVVAR> = <VALUE>     set and export a shell environment variable
set HP64KPATH = <MYPATH>   set and export the shell environment variable that
                              specifies the search path for command files
wait                       pause until <cntrl-c> (SIGINT)
--More--(42%)
```

The help information scrolls onto the screen. If there is more than a screen of information, you must press the space bar to see the next screen, or the < RETURN> key to see the next line, just as you do with the HP-UX **more** command. When all the information on the topic has been displayed (or after you press "q" to quit scrolling through information), press < RETURN> to return to the Softkey Interface.

## Pod Command Help

To access the emulator's firmware resident Terminal Interface help information, use the following commands.

***display pod_command*** <RETURN>

***pod_command*** 'help cf mon' <RETURN>

The command enclosed in string delimiters (", ', or ^ ) is any Terminal Interface command. The command's output is seen in the pod_command display. The Terminal Interface help (or ?) command may be used to provide information on any Terminal Interface command or any emulator configuration option (as the above example shows).

```
Pod Commands
  Time                Command
  monitor option
    cf mon=bg
    cf mon=fg
    cf mon=ubg
    cf mon=ufg
    This configuration item allows you to select the desired monitor option.
  bg-- Default background monitor.
  fg-- Default foreground monitor.
  ubg--Default background monitor with the addition of a user supplied
       routine which is executed while in the monitor.  User code must
       first be loaded using the -g option of the load command.
  ufg--User supplied foreground monitor.  Allows use of a foreground
       monitor that has been tailored to a specific target system.  User
       code must first be loaded using the -f option of the load command.
  ALL MAP TERMS ARE DELETED WHEN THE MONITOR OPTION IS CHANGED!!!
  THE SINGLE STEP VECTOR MUST BE LOADED WHEN USING A FOREGROUND MONITOR!!!

STATUS:   8086--Running in monitor______________________________________...R....
 pod_command 'help cf mon'


pod_cmd  perfinit perfrun  perfend                                    ---ETC--
```

## Loading Absolute Files

The **load** command allows you to load absolute files into emulation or target system memory. Use the "load emul_mem" syntax to load only that portion of the absolute file that resides in memory mapped as emulation RAM or ROM.  To load only the portion of the absolute file that resides in memory mapped as target RAM, use the "load user_mem" syntax. Do not specify "emul_mem" or "user_mem" if you want to load both emulation and target memory. For example:

***load*** cmd_rds <RETURN>

Normally, you will configure the emulator and map memory before you load the absolute file. The default configuration is sufficient for the sample program.

## Displaying Symbols

The HP 64762 and HP 64763 emulators can read absolute files in HP-OMF and OMF-86 formats. When you load a program for the first time, the emulator uses the Symbolic Retrieval Utilities (SRU) to build a symbol database for each module. This database associates symbol names and symbol type information (not data types) with logical addresses. You will see a message on screen indicating the module for which the database is being built.

Once a symbol database is created for a particular module, it does not need to be rebuilt unless the module is changed. You can rebuild modules using the **srubuild** utility (see the *SRU User's Guide*). Or, if you reenter emulation without building symbols, the emulator software will automatically rebuild portions of the symbol database as you reference symbols in modified modules.

Global symbol information is immediately available for the file that you loaded. To obtain local symbol information, you need to specify the module that contains the symbols.

You can use the symbol names instead of addresses when entering expressions as part of an emulation command. Therefore, you don't have to remember segment:offset information to make a measurement. Also, the emulator can display symbols as part of a measurement, using the **set symbols on** command. This helps you relate the measurement to your original program.

The Symbolic Retrieval Utilities (SRU) handle symbol scoping and referencing. Symbols are arranged in a "tree" structure that mimics the natural scoping of your source language as much as possible.

Each absolute file has its own symbol tree. Each entry in the symbol tree has a type and a name. If you are not sure what your "language tree" looks like, you can use the **sruprint** program to print portions of your tree.

All emulation references to symbols (both input from the keyboard and output displays) make use of the tree structure to show the scoping of symbols and to make evident the symbols that have the same name but different scopes.

## OMF-86 examples

The following short C code example should help illustrate how OMF-86 symbols are maintained by SRU and referenced in your emulation commands.

```
int *port_one;
main ()
{
int port_value;

  port_one = 255;
  port_value = 10;

  process_port (port_one, port_value);

} /* end main */
```

**/users/dave/control.c**

```
process_port (int *port_num, int port_data)
{
static int i;
static int i2;

  for (i = 0; i <= 64; i++) {
    *port_num = port_data;
    delay ();
    i = 3;
    port_data = port_data + i;
    }
} /* end of process_port */
```

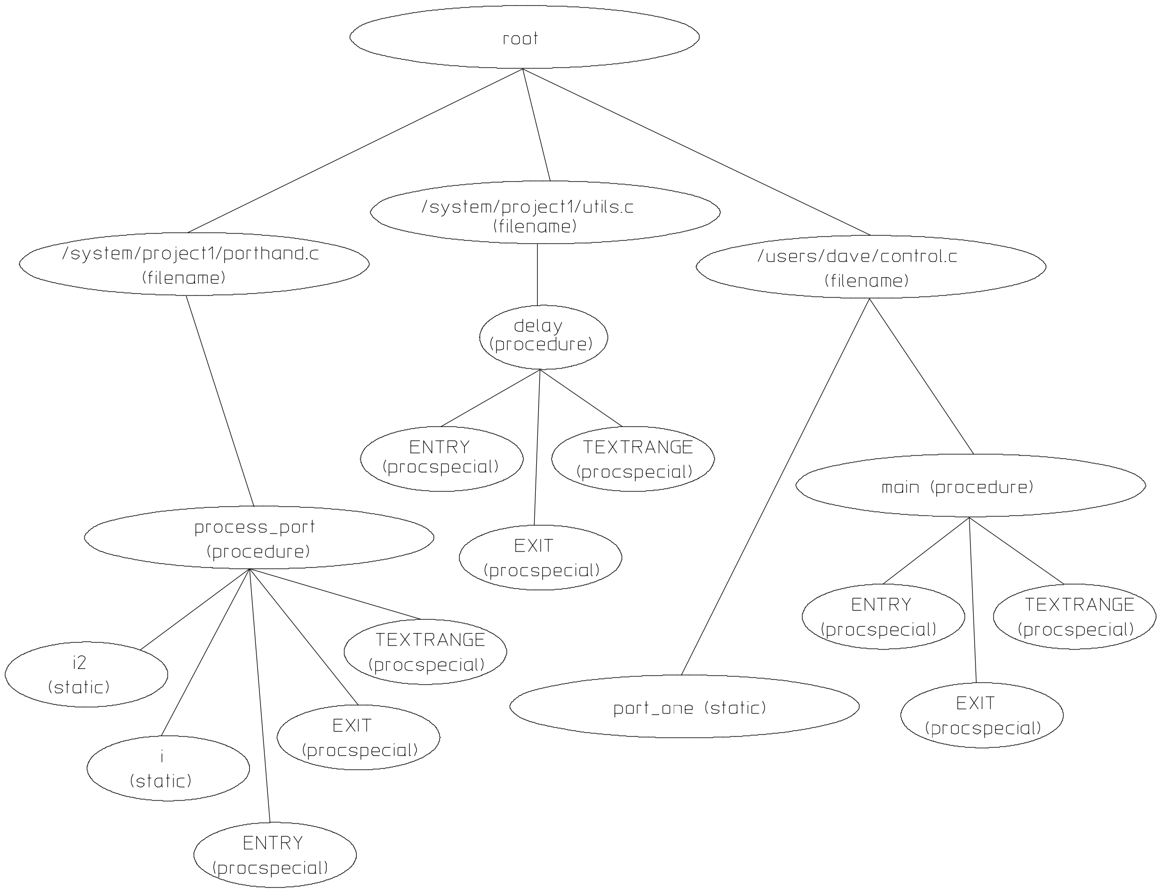
**/system/project1/porthand.c**

```
delay()
{
int i,j;
int waste_time;

  for (i = 0; i <= 256000; i++)
    for (j = 0; j <= 256000; j++)
      waste_time = 0;

} /* end delay */
```

**/system/project1/utils.c**

## OMF-86 Symbol Tree

The OMF-86 symbol tree as built by SRU would appear as follows (this is not a complete symbol tree):




SRU does not build tree nodes for variables that are dynamically allocated on the stack at run-time, such as i and j within the delay () procedure. SRU has no way of knowing where these variables will be at run time and therefore cannot build a corresponding symbol tree entry with run-time addresses.

These are examples of referencing different symbols in the programs shown earlier are:

```
control.c:main
control.c:port_one
```

## SRU searches for symbols

SRU has symbol-searching capability. It also has the ability to explicitly set a “current working symbol” (cws), which allows you to refer to symbols relative to the cws.

When the shell variable HP64KSYMBPATH is set to be a blank-separated list of symbols, a “search list” is set. When a symbol is entered without the leading colon or dot, which forces it to be global, the following happens:

The current working symbol (if there is one) is prefixed to the entered symbol. If the resulting symbol exists, it will be the symbol that is used.

For each entry in HP64KSYMBPATH:

- Prefix the entry with the entered symbol. If the symbol exists, that is the symbol to use.
- Otherwise, remove the last entry in the HP64KSYMBPATH’s symbol and repeat the previous step.

In addition to the “HP64KSYMBPATH” environment variable and cws, a search algorithm to resolve symbol references on the command line is used. These actions will occur when using the filename as an element of the symbol in the command line.

- If the first element of the entered symbol is a filename, SRU will construct a module name from the filename. The module is defined as the basename of the filename, with the extension removed. For example, modulename “PORTHAND” is derived from the path and filename “/system/project1/porthand.c”.
  – If the request was for the address of a line number, SRU will check to see if the symbol < modulename> .< filename> exists. If it does, it will

assume that is the symbol you want. Otherwise it will return the message “symbol not found”.

– If the request was *not* for the address of a line number, SRU will check to see if the symbol with the < filename> replaced with < modulename> exists. If the new symbol exists, SRU will assume that is the symbol you want. Otherwise it will return the message “symbol not found”.

- If no module was derived from the filename, SRU will return the message “symbol not found”.

You can reference different variables with matching identifiers by specifying the complete scope. You can also save on keystrokes by specifying a scope with **cws**. For example, if you are making many measurements involving symbols in the file “porthand.c”, you could specify:

```
cws porthand.c:process_port
```

Then

```
i
BLOCK_1.i
```

are prefixed with “porthand.c:process_port” before the database lookup.

If a symbol search with the current working symbol prefix is not successful, the last scope on the current working symbol is stripped. The symbol you specified is then retested with the modified current working symbol. This does not change the actual current working symbol.

For example, if you set the current working symbol as

```
cws porthand.c:process_port.BLOCK_1
```

and made a reference to symbol i2, the retrieval utilities attempt to find a symbol called

```
porthand.c:process_port.BLOCK_1.i2
```

which would not be found.

The symbol utilities would then strip BLOCK_1 from the current working symbol, yielding

```
porthand.c:process_port.i2
```

which is a valid symbol. If this is still not a valid symbol, this process is repeated until the symbol is found or until there are no more elements in the cws.

You can also specify the symbol type if conflicts arise. Although not shown in the tree, assume that a procedure called “port_one” is also defined in “control.c”. This would conflict with the identifier “port_one” which declares an integer pointer.

SRU can resolve the difference. You must specify:

```
control.c:port_one(static)
```

to reference the variable, and

```
control.c:port_one(procedure)
```

to reference the procedure address.

## For More Information About Symbols

For information about using a special default prefix for low-level symbols, when working with 3rd party symbols, see the file *langinfo.hp*. That file also describes the language used to reduce ambiguous error messages.

More information about symbols and SRU is contained in the *SRU User’s Guide* and in the **--SYMB--** syntax pages in the *Softkey Interface Reference*. If you received a separate manual describing SRU, you can refer to it if this chapter does not contain all of the information you need.

## Displaying Global Symbols

To display global symbols, enter the following command.

***display global_symbols*** <RETURN>

Listed are: address ranges associated with a symbol, the segment with which the symbol is associated, and the offset of that symbol within the segment.

The module names are listed under the heading “Filename Symbols.” For programs where several different object files are linked to form a single absolute, you will see several names listed here. You can enter these names as part of a symbol expression to specify symbols local to a particular module.

```
Global symbols in cmd_rds
Static symbols
Symbol name ____________________ Address range __ Segment _____________ Offset
Cmd_Input                        0000:0600         COMM                    0000
Init                             0000:0400         PROG                    0000
Msg_Dest                         0000:0601         COMM                    0001
Msgs                             0000:0500         DATA                    0000

Filename symbols
Filename ______________________________________________________________________
cmd_rds.S







STATUS:   8086--Running in monitor______________________________________........
```

## Displaying Local Symbols

When displaying local symbols, you must include the name of the source file in which the symbols are defined. For example,

***display local_symbols_in*** `cmd_rds.S: <RETURN>`

```
Symbols in cmd_rds.S:
Static symbols
Symbol name ____________________ Address range __ Segment _____________ Offset
Cmd_A                            0000:0429         PROG                      0029
Cmd_B                            0000:0432         PROG                      0032
Cmd_I                            0000:043B         PROG                      003B
Cmd_Input                        0000:0600         COMM                      0000
End_Msgs                         0000:0534         DATA                      0034
Exe_Cmd                          0000:041E         PROG                      001E
Fill_Dest                        0000:0447         PROG                      0047
Init                             0000:0400         PROG                      0000
Msg_A                            0000:0500         DATA                      0000
Msg_B                            0000:0512         DATA                      0012
Msg_Dest                         0000:0601         COMM                      0001
Msg_I                            0000:0524         DATA                      0024
Msgs                             0000:0500         DATA                      0000
Read_Cmd                         0000:040F         PROG                      000F
Scan                             0000:0416         PROG                      0016

STATUS:   8086--Running in monitor______________________________________........
```

## Displaying Data

You can display the data values of a variable in your program using the **display data** command. For example, suppose you want to see the values of the Cmd_Input byte and all the message strings. Enter:

***set symbols on*** `<RETURN>`

***display data*** `Cmd_Input char, Msg_A` ***thru*** `Msg_B-1 char, Msg_B` ***thru*** `Msg_I-1 char, Msg_I` ***thru*** `End_Msgs char <RETURN>`

You'll see the following display:

```
 Data   :update
   address   label          type        data
 0000 0600  CO|Cmd_Input  char        .
 0000 0500   DATA|Msgs    char[]      Command A entered
 0000 0512  cmd_rd:Msg_B  char[]      Entered B command
 0000 0524  cmd_rd:Msg_I  char[]      Invalid Command

STATUS:   8086--Running user program    Emulation trace complete________........
```

The command **set symbols on** displays the label column, which indicates the symbol associated with each address. It also enables symbol display in other measurement screens, such as display memory, display registers, and display trace.

## Displaying Memory in Mnemonic Format

You can display the absolute code in memory in mnemonic format. To display the memory of the “cmd_rds” program, enter:

***display memory*** `Init` ***mnemonic*** `<RETURN>`

Notice that you can use symbols when specifying expressions. The command above uses the global symbol **Init** to specify the starting address for memory display.

```
 Memory  :mnemonic :file = cmd_rds.S:
   address  label          data
 0000 0400   PROG|Init    B80000      MOV AX,#0000H
 0000 0403                8ED8B80000  MOV DS,AX | MOV AX,#0000H
 0000 0408                8EC08ED0BC  MOV ES,AX | MOV SS,AX | MOV SP,#06F9H
 0000 040F  cmd:Read_Cmd  26C6060006  MOV ES:BYTE PTR COMM|Cmd_Input,#00H
 0000 0415                90          NOP
 0000 0416  cmd_rds:Scan  26A00006    MOV AL,ES:COMM|Cmd_Input
 0000 041A                3C00        CMP AL,#00H
 0000 041C                74F8        JZ P|cmd_rds.S:Scan
 0000 041E  cmd_:Exe_Cmd  3C41        CMP AL,#41H
 0000 0420                7407        JZ |cmd_rds.S:Cmd_A
 0000 0422                3C42        CMP AL,#42H
 0000 0424                740C        JZ |cmd_rds.S:Cmd_B
 0000 0426                E91200      JMP NEAR PTR |cmd_rds.S:Cmd_I
 0000 0429  cmd_rd:Cmd_A  B91200      MOV CX,#0012H
 0000 042C                BE0005      MOV SI,#0500H
 0000 042F                E90F00      JMP NEAR PTR cmd_rd:Write_Msg

STATUS:   8086--Running user program____________________________________........
```

## Running the Program

The **run** command lets you execute a program in memory. Entering the **run** command by itself causes the emulator to begin executing at the current program counter address. The **run from** command allows you to specify an address at which execution is to start.

### From Transfer Address

The **run from transfer_address** command begins code execution at a previously defined "start address." Transfer addresses are defined in assembly language source files with the END assembler directive (pseudo instruction). For example, the sample program defines the address of the label **Init** as the transfer address. The following command will start execution at the address of the **Init** label.

***run from transfer_address*** `<RETURN>`

### From Reset

The **run from reset** command specifies that the emulator begin executing from target system reset (see the "Running From Reset" section in the "In-Circuit Emulation" chapter).

## Displaying Memory Repetitively

You can display memory locations repetitively to constantly update the screen information. For example, to display the **Msg_Dest** locations of the sample program repetitively (in blocked byte format), enter the following command.

***display memory*** `Msg_Dest` ***repetitively blocked bytes*** `<RETURN>`

## Modifying Memory

The sample program is a primitive command interpreter. The program receives commands through a byte sized memory location labeled **Cmd_Input**. You can use the modify memory feature to send a command to the program. For example, to enter the command "A" (41H), use the following command.

***modify memory*** `Cmd_Input` ***bytes to*** `41h <RETURN>`

Or:

***modify memory*** `Cmd_Input` ***string to*** `'A'` `<RETURN>`

After the memory location is modified, the repetitive memory display shows that the “Command A entered ” message is written to the destination locations.

```
 Memory  :bytes :blocked :repetitively
   address       data       :hex                                    :ascii
 0000 0601-08    43   6F   6D   6D   61   6E   64   20      C o m m  a n d
 0000 0609-10    41   20   65   6E   74   65   72   65      A   e n  t e r e
 0000 0611-18    64   20   00   00   00   00   00   00      d   . .  . . . .
 0000 0619-20    00   00   00   00   00   00   00   00      . . . .  . . . .
 0000 0621-28    00   00   00   00   00   00   00   00      . . . .  . . . .
 0000 0629-30    00   00   00   00   00   00   00   00      . . . .  . . . .
 0000 0631-38    00   00   00   00   00   00   00   00      . . . .  . . . .
 0000 0639-40    00   00   00   00   00   00   00   00      . . . .  . . . .
 0000 0641-48    00   00   00   00   00   00   00   00      . . . .  . . . .
 0000 0649-50    00   00   00   00   00   00   00   00      . . . .  . . . .
 0000 0651-58    00   00   00   00   00   00   00   00      . . . .  . . . .
 0000 0659-60    00   00   00   00   00   00   00   00      . . . .  . . . .
 0000 0661-68    00   00   00   00   00   00   00   00      . . . .  . . . .
 0000 0669-70    00   00   00   00   00   00   00   00      . . . .  . . . .
 0000 0671-78    00   00   00   00   00   00   00   00      . . . .  . . . .
 0000 0679-80    00   00   00   00   00   00   00   00      . . . .  . . . .

STATUS:   8086--Running user program____________________________________........
```

## Breaking into the Monitor

The **break** command allows you to divert emulator execution from the user program to the monitor. You can continue user program execution with the **run** command. To break emulator execution from the sample program to the monitor, enter the following command.

***break*** `<RETURN>`

## 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**

The user program must define a stack for correct software breakpoint operation.

**Note**

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

**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 executing user code in the area of the breakpoint modification, program execution may be unreliable.

**Note**

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

When software breakpoints are enabled and 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. This causes an NMI response, as with the **break** command.

Since the system controller knows the locations of defined software breakpoints, it can decide whether the SBI was an enabled software breakpoint or a single-byte interrupt instruction in your target program.

If the SBI was 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, and an "undefined breakpoint" status message is displayed. To continue program execution, you must run or step from the target program's breakpoint interrupt vector address.

When software breakpoints are disabled, the emulator executes INT 3 instructions as the 8086 processor normally would.

## Enabling/Disabling Software Breakpoints

When you initially enter the Softkey Interface, software breakpoints are disabled. If your program contains single-byte interrupt (INT 3) instructions, you may want to disable the software breakpoints feature. This allows your program to operate normally (that is, the execution of these instructions will not cause execution to break into the monitor). To enable the software breakpoints feature, enter the following command.

***modify software_breakpoints enable*** <RETURN>

When software breakpoints are enabled and you set a software breakpoint, an INT 3 instruction will be placed at the specified address. When the INT 3 is executed, program execution will break into the monitor.

Because there is no way to distinguish between a vector fetch and a normal memory read, problems can arise if your code reads from address location 0000C hexadecimal while software breakpoints are enabled. The result will be a break into the monitor, accompanied by the message "undefined breakpoint."

This may be a problem, for example, in memory test code. The only solution is to not enable software breakpoints during the execution of this code.

You can often use the analysis break capability to work around this problem by setting an analysis break after the test code. At that point normal software breakpoints can be enabled.

## Setting a Software Breakpoint

To set a software breakpoint at the address of the **Cmd_I** label, enter the following command.

***modify software_breakpoints set*** `cmd_rds.S:Cmd_I <RETURN>`

Notice that when you use local symbols in expressions, you must include the source file that defines the symbol.

Enter the following command to continue executing the sample program:

***run*** `<RETURN>`

Now, modify the command input byte to an invalid command for the sample program.

***modify memory*** `Cmd_Input` ***bytes to*** `75h <RETURN>`

A message on the status line shows that the software breakpoint was hit. The status line also shows that the emulator is now executing in the monitor.

## Displaying Software Breakpoints

To display software breakpoints, enter the following command.

***display software_breakpoints*** `<RETURN>`

The software breakpoints display shows the inactivated breakpoint. When breakpoints are hit, they become inactivated. To reactivate the breakpoint so that it is “pending,” you must reenter the **modify software_breakpoints set** command.

```
Software breakpoints  :enabled
   address          label          status
 0000 043B         cmd_rd:Cmd_I   inactivated









STATUS:   8086--Running in monitor        Software break: 00000:0043b___........
```

## Clearing a Software Breakpoint

To remove software breakpoint defined above, enter the following command.

***modify software_breakpoints clear***
cmd_rds.S:Cmd_I <RETURN>

The breakpoint is removed from the list, and the original opcode is restored if the breakpoint was pending.

To clear all software breakpoints, you can enter the following command.

***modify software_breakpoints clear*** <RETURN>

# Displaying Registers

Enter the following command to display registers. You can display the basic registers, or an individual register.

***display registers*** <RETURN>

## Stepping Through the Program

The **step** command allows you to step through program execution one or more instructions at a time. Also, you can step from the current program counter or from a specific address. To step through the example program from the address of the software breakpoint set earlier, enter the following command.

***step*** <RETURN> <RETURN> <RETURN> ...

You can continue to step through the program just by pressing the < RETURN> key. When any command appears on the command line, it may be entered by pressing < RETURN> .

```
Registers

  AX 0075  BX 00FA  CX 0000  DX FFFF   BP 00EA  FL F006           p

Step_PC 0000:043BH  MOV CX,#0010H
Next CS:IP 0000:043EH
  IP 043E  SP 06F9  SI 0512  DI 0621   CS 0000  SS 0000  DS 0000  ES 0000
  AX 0075  BX 00FA  CX 0010  DX FFFF   BP 00EA  FL F006           p

Step_PC 0000:043EH  MOV SI,#0524H
Next CS:IP 0000:0441H
  IP 0441  SP 06F9  SI 0524  DI 0621   CS 0000  SS 0000  DS 0000  ES 0000
  AX 0075  BX 00FA  CX 0010  DX FFFF   BP 00EA  FL F006           p

Step_PC 0000:0441H  LEA DI,COMM|Msg_Dest
Next CS:IP 0000:0445H
  IP 0445  SP 06F9  SI 0524  DI 0601   CS 0000  SS 0000  DS 0000  ES 0000
  AX 0075  BX 00FA  CX 0010  DX FFFF   BP 00EA  FL F006           p

STATUS:   8086--Stepping complete_______________________________________........
```

Enter the following command to continue sample program execution from the current program counter.

***run*** <RETURN>

During the process of breaking into the monitor, the HP 64762 emulator “jams” reads from the NMI vector. These “jams” modify the data seen by the processor. This can result in unexpected behavior when stepping instructions that read from the NMI vector location (00008-0000B).

For example, stepping through the instruction

```
MOV AX,WORD PTR 8
```

with a DS value of “0” will load unexpected data into AX. You can use software breakpoints at times to work around this limitation, if the state of the processor must be interrogated immediately after such an instruction.

## Using the Analyzer

HP 64700 emulators contain an emulation analyzer. The emulation analyzer has 47 trace signals, which monitor the internal emulation lines (address, data, and status). 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.

### Specifying a Simple Trigger

Suppose you want to trace program execution after the sample program reads a “B” (42H) command from the command input byte. The following command makes this trace specification.

***trace after*** `Cmd_Input` ***data*** `0xx42h` ***status memread*** `<RETURN>`

The message “Emulation trace started” will appear on the status line. Now, modify the command input byte to “B” with the following command.

***modify memory*** `Cmd_Input` ***bytes to*** `42h <RETURN>`

The status line now shows “Emulation trace complete.”

### Displaying the Trace

The following trace listings are of program execution on the 8086 emulator. Trace listings of program execution on the 8088 emulator look different because of the multiplexed data bus. For example, opcodes are fetched a byte at a time.

Make sure to enable symbol display by typing:

***set symbols on*** `<RETURN>`

To display the trace, enter:

***display trace*** <RETURN>

```
Trace List                 Offset=0
Label:      Address       Data          Opcode or Status          time count
Base:       symbols       hex          mnemonic w/symbols          relative
after   COMM|Cmd_Input      4942        xx42H, mem read            ------------
+001   cmd_rds.S:+0001A     0000  INSTRUCTION--opcode unavailable    360     nS
+002   cmd_rds.S:+0001C     F874        F874H, opcode fetch          120     nS
+003   cmd_rds.:Exe_Cmd     413C        413CH, opcode fetch          520     nS
+004   cmd_rds.S:+0001C     0000  JZ P|cmd_rds.S:Scan                120     nS
+005   cmd_rds.S:+00020     0774        0774H, opcode fetch          360     nS
+006   cmd_rds.:Exe_Cmd     0000  CMP AL,#41H                        120     nS
+007   cmd_rds.S:+00022     423C        423CH, opcode fetch          400     nS
+008   cmd_rds.S:+00020     0000  JZ |cmd_rds.S:Cmd_A                120     nS
+009   cmd_rds.S:+00024     0C74        0C74H, opcode fetch          360     nS
+010   cmd_rds.S:+00022     0000  CMP AL,#42H                        120     nS
+011   cmd_rds.S:+00026     12E9        12E9H, opcode fetch          400     nS
+012   cmd_rds.S:+00024     0000  JZ |cmd_rds.S:Cmd_B                120     nS
+013   cmd_rds.S:+00028     B900        B900H, opcode fetch          360     nS
+014   |cmd_rds.S:Cmd_B     12B9        12B9H, opcode fetch            1.3   uS

STATUS:   8086--Running user program      Emulation trace complete______........
```

Line 0 (labeled “after”) in the trace list above 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 are not executed (lines 11 and 13).

Notice also that the data values for internal cycles (line 4, line 6, on line 8, etc.) are zero. Since internal cycles can happen at any time, the actual data on the bus may be changing. Therefore, the data and status fields of internal cycles are set to zero.

To list the next lines of the trace, press the <PGDN> or <NEXT> key.

The resulting display shows the branch to **Write_Msg** and the beginning of the instructions that move the “Entered B command ” message to the destination locations.

```
Trace List                   Offset=0
Label:      Address       Data           Opcode or Status          time count
Base:       symbols       hex           mnemonic w/symbols          relative
+015   cmd_rds.S:+00034     BE00        BE00H, opcode fetch          480     nS
+016   |cmd_rds.S:Cmd_B     0000  MOV CX,#0012H                      120     nS
+017   cmd_rds.S:+00036     0512        0512H, opcode fetch          400     nS
+018   cmd_rds.S:+00035     0000  MOV SI,#0512H                      120     nS
+019   cmd_rds.S:+00038     06E9        06E9H, opcode fetch          360     nS
+020   cmd_rds.S:+0003A     B900        B900H, opcode fetch          520     nS
+021   cmd_rds.S:+00038     0000  JMP NEAR PTR cmd_rd:Write_Msg      120     nS
+022   cmd_rds.S:+0003C     0010        0010H, opcode fetch          360     nS
+023   cmd_rd:Write_Msg     8D05        8DxxH, opcode fetch            1.2   uS
+024   cmd_rds.S:+00042     013E        013EH, opcode fetch          520     nS
+025   cmd_rd:Write_Msg     0000  LEA DI,COMM|Msg_Dest               120     nS
+026   cmd_rds.S:+00044     F306        F306H, opcode fetch          360     nS
+027   cmd_rds.S:+00046     C6A4        C6A4H, opcode fetch          520     nS
+028   cmd_rds.S:+00048     0005        0005H, opcode fetch          480     nS
+029   cmd_rds.S:+00045     0000  REP MOVSB                          120     nS

STATUS:   8086--Running user program      Emulation trace complete______........
```

## 8086/8088 Analysis Status Qualifiers

The above example used the status qualifier “memread.” The following analysis status qualifiers also can be used with the 8086/8088 emulators.

| 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. |
| **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. |

| Qualifier | Status Bits (46..36) | Description |
|---|---|---|
| **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. |

### For a Complete Description

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

## Exiting the Softkey Interface

There are several options when exiting the Softkey Interface:

- exiting and releasing the emulation system
- exiting with the intent of reentering (continuing)
- exiting locked from multiple emulation windows
- exiting (locked) and selecting the measurement system display or another module

### End Release System

To exit the Softkey Interface, releasing the emulator for use by others, enter the following command.

***end release_system*** <RETURN>

### Ending to Continue Later

You also can exit the Softkey Interface without specifying any options. This locks the emulator. When locked, other users cannot use it. The emulator configuration is saved so that it can be restored the next time you enter (continue) the Softkey Interface.

***end*** <RETURN>

## Ending Locked from All Windows

When you use the Softkey Interface within window systems, the "end" command with no options exits only that window. To end locked from all windows, enter the following command.

***end locked*** <RETURN>

This option only appears when you enter the Softkey Interface via the **emul700** command. When you enter the Softkey Interface via **pmon** and **MEAS_SYS**, only one window is permitted.

Refer to the *Softkey Interface Reference* manual for more information on using the Softkey Interface with window systems.

## Selecting the Measurement System Display or Another Module

When you enter the Softkey Interface via **pmon** and **MEAS_SYS**, you can select the measurement system display or another module in the measurement system when exiting the Softkey Interface. This type of exit is also "locked." That is, you can continue the emulation session later. For example, to exit and select the measurement system display, enter the following command.

***end select measurement_system*** <RETURN>

This option is not available if you entered the Softkey Interface using the **emul700** command.

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 will:

- Describe the issues concerning the installation of the emulator probe into target systems.
- Show you how to install the emulator probe.
- Discuss features of in-circuit emulation.

## Prerequisites

Before performing the tasks described in this chapter, you should be familiar with general emulator operation. Refer to the *HP 64700 Concepts of Emulation And Analysis* 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.

HP discourages the use of more than one pin extender, since pin extenders degrade signal quality. Some installations need the pin extender for mechanical clearance.

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 impact damage. Remove the foam pin protector before running performance verification (see the Terminal Interface **pv** command).

**Caution**

*Possible Damage to the Emulator Probe.* The 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 by static electricity.

**Caution**

*Possible Damage to the Emulator.* Make sure target system power is OFF before installing the emulator probe into the target system. Do not install the emulator probe into the processor socket with power applied to the target system.

**Caution**

***Incorrect Probe Installation Will Damage the Emulator Probe.*** Make sure pin 1 of probe connector is aligned with pin 1 of the socket. When you install the emulation probe, insert the probe into the processor socket so that pin 1 of the connector aligns with pin 1 of the socket (as shown in figure 3-1).

## Auxiliary Output Lines

There are three auxiliary output lines provided by the HP 64762/3 emulators:

**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 emulator 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 the signal to tristate (in other words, select the high Z output) of any target system devices on the 808X address/data bus. Target system devices should be tristated, because 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.

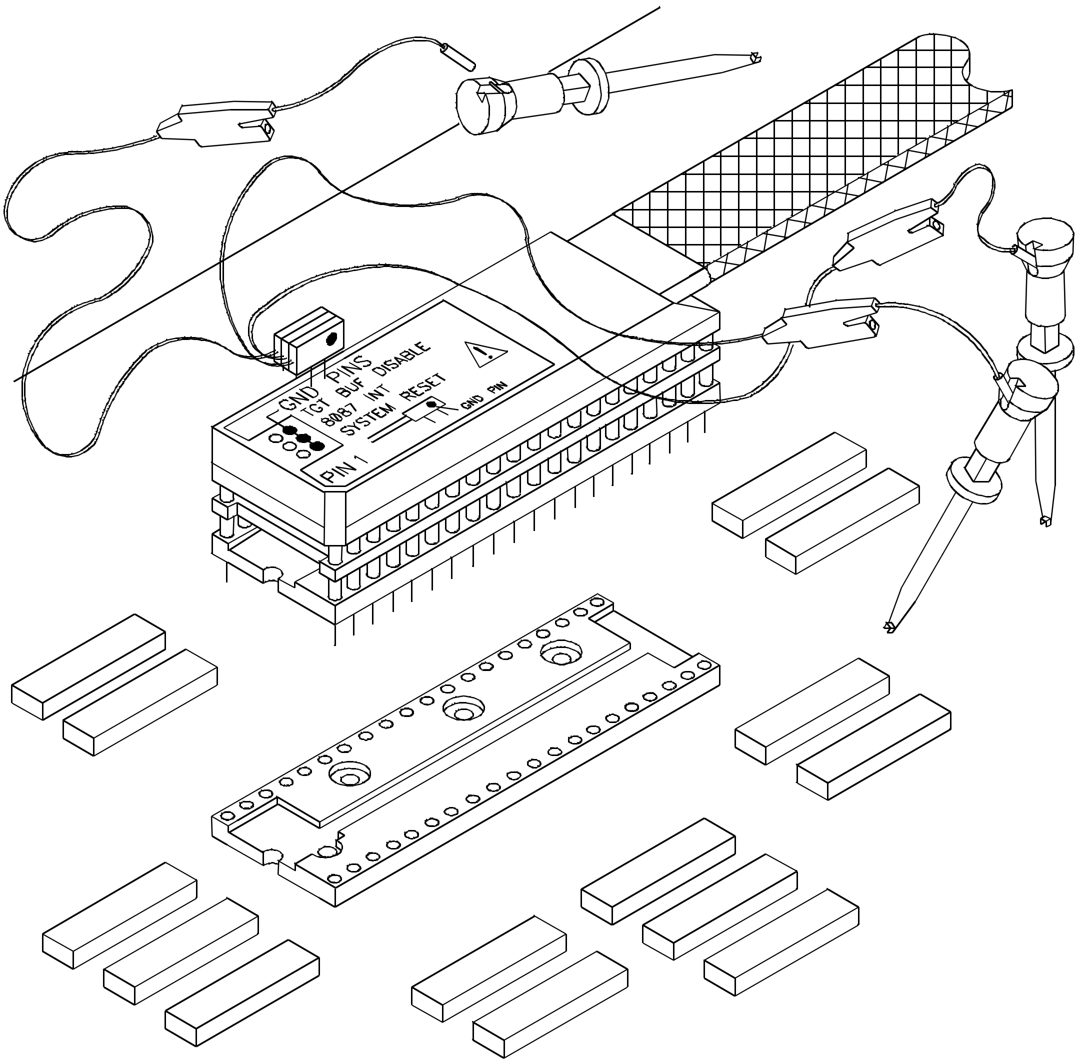



**Figure 3-1. Connecting the Emulator Probe**

**SYSTEM RESET** This active-high CMOS output should be used to synchronously reset the emulator and the target system. You need to use this output when an 8089 I/O processor is in the target system, because the coprocessor 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 items. Refer to the chapter on "Configuring the 8086/8088 Emulator" for more information on these 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. But, 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, and the emulator begins running from 0FFFF0H. (This occurs within a few cycles of the RESET signal). To specify a run from target reset, select:

***run from reset*** <RETURN>

The status now shows that the emulator is “Awaiting target reset.” After the target system is reset, the status line message changes to show the appropriate emulator status.

You also can enter the **run from reset** 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 asserts and negates RESET, the emulator will run from 0FFFF0H.

**Note**

Though the external clock is automatically selected when the “run from reset” command is entered, the emulation configuration does not reflect this change if the internal clock was previously selected. Attempting to modify the emulator configuration (prior to target power up) to select an internal clock does not work. (But, you can enter the **cf clk= int** pod command to reselect the internal clock.)

## Connecting SYS RESET and TGT BUF DISABLE to the Target System

The following diagram shows an example of how the SYS RESET and TGT BUF DISABLE auxiliary signals could be connected in a target system.

Suppose that you want an 8087 processor in the target system to access emulation memory for instructions and/or data. In that case, the emulator configuration would be set to allow direct memory access to emulation memory.

The TGT BUF DISABLE signal would be connected to disable any target memory devices that might drive the data bus during emulation memory accesses. (The emulator will drive "read" data out from the emulation probe.)

The SYS RESET signal is driven high any time the emulation processor is being reset, regardless of whether the source of the reset is the target system or an emulation command. Any devices in the target system that must be reset in unison with the 8086/88 processor should be driven with the SYS RESET signal so that they will be reset in response to an emulation command. This is not necessary if you always start the emulator with a **run from reset** command, because then the target RESET signal can reset all devices.

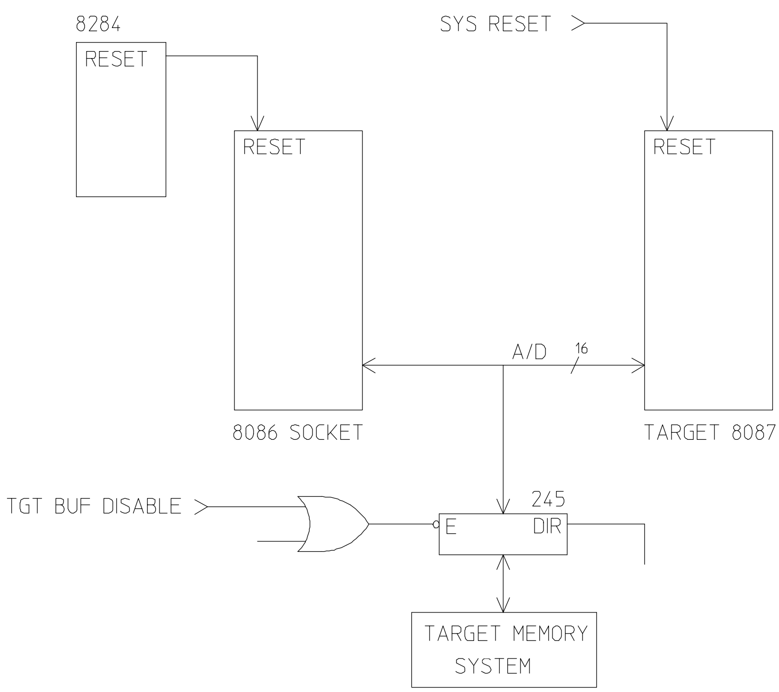



**Figure 3-2. Connecting SYS RESET and TGT BUF DISABLE**

*Note that you should not use SYS RESET to assert the RES input of an 8284, because this will result in a latched reset condition.*

## Notes

# 4

# Configuring the 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, you can allow emulator execution to be diverted into the monitor when commands request access of target system resources (target system memory, register contents, and so on).

The emulator is a versatile instrument and it may be configured to suit your needs at any stage of the development process. This chapter describes the options available when configuring the HP 64762/3 emulators.

Access the configuration options with the following command:

***modify configuration*** <RETURN>

After entering the command above, you will be asked questions regarding the emulator configuration. The configuration questions are listed below, grouped by class.

#### General Emulator Configuration:

- Specifying the emulator clock source (internal/external).
- Selecting monitor entry after configuration.
- Restricting to real-time execution.

#### Memory Configuration:

- Selecting the emulation monitor type.
- Specifying the monitor segment.
- Specifying the monitor offset.
- Mapping memory.

#### Emulator Pod Configuration

- Enabling READY inputs from target system.
- Enabling max segment algorithm for physical run addresses.
- Selecting the size of target memory accesses.
- Enabling background cycles to target system.
- Selecting queue status to target system.
- Enabling the internal numeric coprocessor (internal 8087 configuration questions follow).
- Enabling DMA access to/from emulation memory.

#### Debug/Trace Configuration

- Enabling breaks on writes to ROM.
- Specifying tracing of foreground/background cycles.

**Simulated I/O Configuration.** See the *Simulated I/O* reference manual.

**Interactive Measurement Configuration.** See the chapter on coordinated measurements in the *Softkey Interface Reference* manual.

**External Analyzer Configuration.** See the *Analyzer Softkey Interface User's Guide.*

## General Emulator Configuration

The configuration questions in this section determine general emulator operation.

### Micro-processor Clock Source?

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

#### internal

Selects the internal 8 MHz clock oscillator as the emulator clock source.

**external**

Selects an external target system clock source between 2 and 10 MHz.

**Note**

Changing the clock source drives the emulator into the reset state. If you answer "yes" to the "Enter monitor after configuration?" question that follows, the emulator resets (due to the clock source change) then breaks into the monitor when you save the configuration.

## Enter Monitor After Configuration?

This question allows you to select whether the emulator will be running in the monitor or held in the reset state on completion of the emulator configuration.

The answer to this configuration question is important in some situations. For example, when the external clock is selected and the target system is turned off, do not select reset to monitor. Otherwise, configuration will fail. When an external clock source is specified, this question becomes "Enter monitor after configuration (using external clock)?" and the default answer becomes "no."

**yes**

When you select reset to monitor, the emulator will be running in the monitor after configuration is complete. If the reset to monitor fails, the previous configuration will be restored.

**no**

After the configuration is complete, the emulator will be held in the reset state.

## Restrict to Real-Time Runs?

The "restrict to real-time" question lets you configure the emulator to refuse commands which cause the emulator to break to monitor.

**no**

The emulator accepts all commands, despite whether they require a break to the emulation monitor.

**yes**

When runs are restricted to real-time and the emulator is running the user program, all commands that cause a break (except **reset**, **break**, **run**, and **step**) are refused. For example, the following commands are not allowed when runs are restricted to real-time:

- Display/modify registers.
- Display/modify target system memory.
- Display/modify I/O.

**Caution**

If your target system circuitry depends on constant program execution, you should restrict the emulator to real-time runs. This will help prevent target system damage. But, remember that you can still execute the **reset**, **break**, and **step** commands. Use caution when executing these commands.

## Memory Configuration

The memory configuration questions allow you to select the monitor type, to select the segment and offset address of the monitor, and to map memory. To access the memory configuration questions, you must answer “yes” to the following question.

**Modify memory configuration?**

### Monitor Type?

The monitor is a program executed by the emulation processor. 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. This 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 which 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 use 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. However, 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.

**Note**

All memory mapper terms are deleted when the monitor type changes!

### background

The default emulator configuration selects the background monitor. When using 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 no vector table is loaded and the vector table area, 0-3FFH, maps to "guarded memory." (If locations 0-3FFH are unmapped, a default memory type of "guarded" specifies these locations 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 because the 8086 processor will not halt if 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 executes on monitor entry. Code in the second section executes once for each loop through the monitor. Code in the third section executes on monitor exit. Code in the fourth section executes when entering the monitor from reset.

**Restrictions on User Code Loaded into Background.** Here are the restrictions on the code that you load into the background monitor:

- 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 user code restricted area. 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 its restricted area. Other locations in the 2K bytes reserved for the monitor contain the monitor program and data. Also, jumping to certain locations outside the user code restricted range will put the emulator into different modes. 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.

### Background Monitor Name?

This question will be asked when you select the "user_background" monitor type. Enter the name of the absolute file that contains the code to be inserted into the background monitor. This file will be loaded at the end of the configuration session.

You can reload the background code while in emulation with the following command.

**_load bkg_mon_** abs_file <RETURN>

```
"8086"
;@(mktid) (01.00 19Jan89)

; 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 *****************************************************

;
```

**Figure 4-1.  /usr/hp64000/monitor/bmon8086.S**

```
;-----------------------------------------------------------------------------
; 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
```

**Figure 4-1. /usr/hp64000/monitor/bmon8086.S (Cont'd)**

```
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 <name>
;
; Where <name> (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
```

**Figure 4-1. /usr/hp64000/monitor/bmon8086.S (Cont'd)**

```
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
```

**Figure 4-1. /usr/hp64000/monitor/bmon8086.S (Cont'd)**

```
; 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
```

**Figure 4-1. /usr/hp64000/monitor/bmon8086.S (Cont'd)**

#### foreground

Selecting the foreground monitor uses processor address space. The foreground monitor occupies 2K bytes of memory at 0FF800H by default. See the “Monitor Segment?” and “Monitor Offset?” configuration questions.

---

**Note** Do *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 carries out commands that

- display/modify the contents of target system memory
- display/modify the contents of memory mapped I/O ports
- display/modify the contents of emulation processor registers
- step through program execution.

The background monitor’s execution is normally hidden from the target system. (You can choose to drive background cycles to the target system with the “Enable Background Cycles to Target System?” configuration question). 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. The monitor remains in the 2K bytes of emulation memory reserved, and you still have 126K bytes (or 510K bytes, depending on the emulator model number) of remaining emulation memory. When you select the foreground monitor, the monitor occupies 2K bytes of 8086 memory space.

When the foreground monitor is selected, breaking into the monitor still occurs in background, but the rest of the monitor program executes in foreground.

**Using the Foreground Monitor.** When using 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 on monitor entry.

- You must set up your vector table to point to locations in the foreground monitor program. The vector table (shown in figure 4-2) 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 choose for the monitor program. (The segment values in the vector table file that follows match the default location of the monitor.)

  To change the segment location, modify the EQU statement for the MONSEGMENT variable to the appropriate segment address.

  If you change the “Monitor Offset?” response (later in this section) you must modify the vector table statements which calculate the address offsets.

- You must assemble, link and load the vector table. The load address you specify is unimportant, since all addresses are defined with ORG statements.

- If you use the standard foreground monitor, you don’t need to assemble, link or load it. It is already resident in ROM. If you customize the monitor, you must assemble, link and load it like any other program.

### user_foreground

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

```
"8086"
;@(mktid) (01.00 19Jan89)


; 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
```

**Figure 4-2. /usr/hp64000/monitor/v8086.S**

```
DW      VTABLEAREA+ENTRYSIZE*10
DW      MONSEGMENT

DW      VTABLEAREA+ENTRYSIZE*11
DW      MONSEGMENT

DW      VTABLEAREA+ENTRYSIZE*12
DW      MONSEGMENT

DW      VTABLEAREA+ENTRYSIZE*13
DW      MONSEGMENT
```

**Figure 4-2. /usr/hp64000/monitor/v8086.S (Cont'd)**

the emulation system controller. The foreground monitor program source file comes with the emulator and is described in the "Foreground Monitor Description" appendix.

### Foreground Monitor Name?

This question will be asked when you select the "user_foreground" monitor type. Enter the name of the absolute file that contains the code to be inserted into the background monitor. This file will be loaded at the end of the configuration session.

While you are in emulation, you can reload the foreground monitor ‘with the following command.

***load fg_mon*** abs_file <RETURN>

## Reset Map?

This question will be asked if you change the monitor type or relocate the monitor (see the "Monitor Segment? and Monitor Offset?" section that follows). Changes in the monitor type or location reset the memory map. This question reminds you that the map will be reset and allows you to confirm your decision.

### no

The memory map is not reset, and the monitor type or monitor location (whichever changed and prompted the question) is not changed.

**yes**

The memory map is reset due to the change in monitor type or location.

## Monitor Segment? and Monitor Offset?

The default emulator configuration locates the monitor at 0FF800H (monitor segment = 0FF80H and monitor offset = 0H). You can relocate the monitor to any 2K byte boundary. The location of the background monitor may be important. It will specify which target system locations are read if background cycles are made visible to the target system (which is the default). The location of a foreground monitor is important because it will occupy part of the processor address space. Foreground monitor locations must not overlap target system programs.

When you enter monitor block addresses, you must only specify addresses on 2K byte boundaries. Otherwise, the configuration is invalid, and the previous configuration is restored.

If you relocate the foreground monitor segment, remember to modify the MONSEGMENT definition in the vector table (described earlier). If you change the offset, you need to modify the offset address calculations. Remember to assemble, link and load the vector table program.

**Note**

Relocating the monitor removes all memory mapper terms.

## Mapping Memory

Depending on the emulator model number, emulation memory has 128 or 512 kilobytes, mappable in 1 kilobyte blocks. The monitor occupies 2 kilobytes, leaving 126 or 510 kilobytes of emulation memory which you may use. The emulation memory system does not need wait states.

The memory mapper allows you to characterize memory locations. You can specify whether a certain range of memory is present in the target system or whether you will use 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.

When you select a foreground or user foreground monitor, a 2 kilobyte block is automatically mapped at the address specified by the “Monitor segment?” and “Monitor offset?” questions.

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 “Debug/Trace Configuration” section which follows).

### Determining the Locations to 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 4-3 shows a sample linker load map listing.

```
HP 64000+ Linker

FILE/PROG NAME                PROGRAM    DATA       COMMON     ABSOLUTE
--------------------------------------------------------------------------------
cmd_rds.R                     00000400   00000500   00000600   FFFF0000-FFFF0004

next address                  00000453   00000534   000006FB
XFER address = 00000400  Defined by cmd_rds.R
Current working directory = /users/guest/dir86
Absolute file name = cmd_rds.X
```

**Figure 4-3. Example Load Map Listing**

From the load map listing, you can see that the sample program occupies locations in four address ranges. The program and absolute areas, which contain the opcodes and operands of the sample program, occupy locations 400H through 452H and 0FFFF0H through 0FFFF4H. 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.

Two mapper terms must be specified. Since the program writes to the destination locations, the mapper block containing 600H through 6FAH should not be mapped as ROM memory. To map memory for the sample program, enter the following mapper commands:

***delete all*** <RETURN>

400h ***thru*** 7ffh ***emulation ram*** <RETURN>

0ffc00h ***thru*** 0fffffh ***emulation rom*** <RETURN>

***end*** <RETURN>

Figure 4-4 shows the memory mapper display.

```
Emulation memory blocks: available =  124   mapped =    2    size =  1k   bytes
entry      range        type
 1     400H-    7FFH EMUL/RAM
 2   FFC00H-  FFFFFH EMUL/ROM












STATUS:   Mapping emulation memory, default unspecified blocks:  guarded...R....
 end
```

**Figure 4-4. Memory Mapper Display**

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

## Emulator Pod Configuration

To access the emulator pod configuration questions, you must answer “yes” to the following question.

**Modify emulator pod configuration?**

### Enable READY Inputs From Target System?

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

**no**

When ready inputs from the target system are disabled, emulation memory accesses ignore the ready signal from the target system (no wait states are inserted).

**yes**

When ready inputs from the target system are enabled, emulation memory accesses honor ready signals from the target system (wait states are inserted if requested).

### Enable Max Segment Algorithm?

The run and step 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 run or step 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 run and step 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)
```

### Target Memory Access Size?

When a command requests the monitor to read or write target system memory or I/O, the monitor program will use this configuration item to decide 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).

### Enable Background Cycles to Target System?

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 appears 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 must locate the monitor in memory where read operations will not cause an undesired interaction, you can change the base address of the monitor. (Refer to the “Monitor Segment?” and “Monitor Offset?” configuration questions).

**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 System?

When the 8086 is in 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 Internal Numeric Coprocessor?

The HP 64762/3 emulators contain an internal 8087 numeric coprocessor. You 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, typically out of circuit, 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 Access To/From Emulation Memory?” section which follows).

#### no

When the internal 8087 is disabled, the internal 8087 will not operate and numeric op-codes are ignored by the emulator (unless there is an 8087 in the target system and external DMA is enabled). Both RQ/GT lines are available to the target system when the internal numeric coprocessor is disabled.

#### yes

When the internal 8087 numeric coprocessor is enabled, the emulator’s internal 8087 coprocessor will respond to numeric op-codes in the instruction stream. One of the 8086/8088 RQ/GT lines is taken by the 8087 when it is enabled (see the “Internal Numeric Coprocessor RQ/GT Pin?” section below).

When the internal numeric coprocessor is enabled, you have additional configuration options.

- Selecting the RQ/GT line for the internal 8087.
- Selecting the 8086/8088 INTR source.

### Internal Numeric Coprocessor RQ/GT Pin?

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

| | |
|---|---|
| RQ_GT0 | The RQ/GT0 line is used by the internal 8087. If the internal 8087 is enabled, the emulator will ignore this line from the target system. |

| | |
|---|---|
| RQ_GT1 | The RQ/GT1 line is used by the internal 8087. If the internal 8087 is enabled, the emulator will ignore this line from the target system. |

### INTR Input 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.

| | |
|---|---|
| target | 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. |
| ncp | When the internal 8087 is selected as the INTR source, the INT output of the internal 8087 numeric coprocessor drives the INTR input.<br><br>When the 8086/8088 INTR source is the internal 8087, an additional configuration option allows you to specify the internal interrupt vector (see the “Internal Interrupt Vector?” section below). |

**Internal Interrupt Vector?** If the internal 8087 is selected as the source for the emulation processor INTR input, the value specified for this configuration question is 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 Access To/From Emulation Memory?

If you enable external DMA access to emulation memory, target system devices which reside on the local 8086/8088 bus and conform to the 808X MAX mode bus timing can access emulation memory. For example, an external 8087 meets this requirement.

**no**

If you disable external DMA, external devices cannot access emulation memory and cannot track the operation of emulation memory instructions. Here, the TGT BUF DISABLE line need not be used. (See below.)

**yes**

If you enable external DMA, you must connect the auxiliary output line TGT BUF DISABLE to target system devices that can drive the 808X address/data bus. The devices should be tristated (set to high Z output) 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 at the user probe. (The TGT BUF DISABLE signal goes active at the start of T2 in any bus cycle that accesses emulation memory; it goes inactive in T4.)

Enabling DMA access to/from emulation memory automatically sends “flush” queue status to the target system while the emulator is in background. See the previous topic “Send Flush Queue Status to Target System?”. Queue status can subsequently be set back to NOP although this is not recommended.

## Debug/Trace Configuration

The debug/trace configuration questions allow you to specify breaks on writes to ROM and that the analyzer trace foreground/background execution. To access the debug/trace configuration questions, you must answer “yes” to the following question.

**Modify debug/trace options?**

## Break Processor on Write to ROM?

This question allows you to specify that the emulator break to the monitor upon attempts to write to memory space mapped as ROM. The emulator always prevents the processor from actually writing to memory mapped as emulation ROM. The emulator cannot prevent writes to target system RAM locations mapped as ROM, though the write to ROM break is enabled.

**yes**

The emulator will break into the emulation monitor whenever the user program attempts to write to a memory region mapped as ROM.

**no**

The emulator will not break to the monitor on a write to ROM.

**Note**

The **rom** trace command status option allows you to use "write to ROM" cycles as trigger and storage qualifiers. For example, you could use the following command to trace about a write to ROM:

***trace about status rom*** <RETURN>

## Trace Background or Foreground Operation?

This question allows you to specify whether the analyzer trace only foreground emulation processor cycles, only background cycles, or both foreground or background cycles. When background cycles are stored in the trace, all but mnemonic lines are tagged as background cycles.

**foreground**

Specifies that the analyzer trace only foreground cycles. This is the default.

**background**

Specifies that the analyzer trace only background cycles. (This is rarely a useful setting.)

**both**

Specifies that the analyzer trace both foreground and background cycles. You may wish to specify this option so that all emulation processor cycles may be viewed in the trace display.

## Simulated I/O Configuration

See the *Simulated I/O* reference manual for descriptions of the simulated I/O feature and configuration options.

## Interactive Measurement Configuration

The interactive measurement configuration questions are described in the chapter on coordinated measurements in the *Softkey Interface Reference* manual.

## External Analyzer Configuration

See the *Analyzer Softkey Interface User's Guide* for descriptions of the external analyzer configuration options.

## Saving a Configuration

The last configuration question allows you to save the configuration specifications in a file, which can be loaded into the emulator later.

**Configuration file name? < FILE>**

The name of the last configuration file is shown. No filename is shown if you are modifying the default emulator configuration.

If you press < RETURN> without specifying a filename, the configuration is saved to a temporary file. This file is deleted when you exit the Softkey Interface with the “end release_system” command.

When you specify a filename, the configuration will be saved to two files with extensions of “.EA” and “.EB.” The file with the “.EA” extension is the “source” copy of the file, and the file with the “.EB” extension is the “binary” or loadable copy of the file.

Ending emulation (with the “end” command) saves the current configuration, including the name of the most recently loaded configuration file, into a “continue” file. The continue file is not normally accessed.

## Loading a Configuration

Previously saved configuration files may be loaded with this Softkey Interface command:

***load configuration*** <FILE> <RETURN>

This feature is especially useful after you have exited the Softkey Interface with the **end release_system** command. It saves you from having to modify the default configuration and answer all the questions again.

To reload the current configuration, you can enter the following command.

***load configuration*** <RETURN>

# 5

# Using the Emulator

## Introduction

The “Getting Started” chapter showed you how to use the basic features of the 8086/8088 emulator. This chapter describes more advanced emulator features.

This chapter discusses:

- Register names and classes.
- Features available via **pod_command**.

This chapter shows you how to:

- Store the contents of memory into absolute files.
- Display I/O port locations.

## 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** |
|---|---|---|
| OTHER | AH, AL, BH, BL, CH, CL, DH, DL | 8-Bit Registers |
| BASIC | AX, BX, CX, DX, BP, SI, DI, DS, ES, SS, SP, IP, CS, FL | All Basic Registers |

| **< REG CLASS>** | **< REG NAME>** | **Description** |
|---|---|---|
| GEN | AX, BX, CX, DX | General Registers |
| SEG | DS, ES, SS, CS | Segment Registers |
| PTR | BX, BP, SI, DI, DS, ES | Pointer Registers |
| NCP<br>(Internal 8087 numeric coprocessor registers) | CTRL<br>STAT<br>IPTR<br>OPTR<br>OPC<br>TAG<br>ST0, ST1, ST2, ST3, ST4, ST5, ST6, ST7 | Control Word<br>Status Word<br>Exception Pointer Instruction Address<br>Exception Pointer Operand Address<br>Exception Pointer Instruction Opcode<br>Tag Word<br>Register Stack |

## Features Available via Pod Commands

Several emulation features available in the Terminal Interface but not in the Softkey Interface may be accessed via the following emulation commands.

***display pod_command*** <RETURN>

***pod_command*** '<Terminal Interface command>' <RETURN>

Some Terminal Interface features not available in the Softkey Interface are:

- Copying memory.
- Searching memory for strings or numeric expressions.
- Performing coverage analysis.

Refer to your Terminal Interface documentation for information on how to perform these tasks.

**Note**

Be careful when using **pod_command**. The Softkey Interface and the configuration files assume that the HP 64700 pod configuration is changed only by the Softkey Interface. What you see in **modify configuration** will NOT reflect the HP 64700 pod's configuration if you change the pod's configuration with this command. Also, do not use commands that affect the communications channel. Other commands may confuse the protocol depending on their use. The following commands are not recommended for use with **pod_command**:

**stty**, **po**, **xp -** Do not use, will change channel operation and hang.
**echo**, **mac -** Usage may confuse the protocol in use on the channel.
**wait -** Do not use, will tie up the pod, blocking access.
**init**, **pv -** Will reset pod and force end release_system.
**t -** Do not use, will confuse trace status polling and unload.

## 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.

```
store memory 400h thru 452h to absfile
<RETURN>
```

The above command stores the contents of memory locations 400H-452H in the absolute file "absfile.X." Notice the ".X" extension appended to the specified filename.

## Displaying I/O Port Locations

The 8086/8088 Softkey Interface allows you to display I/O port locations. For example:

***display io_port absolute bytes*** <RETURN>

**Note**

The size of the locations to be displayed (in other words, "bytes" or "words") must agree with the answer given for the "Target memory access size?" emulator pod configuration question.

## Coordinated Measurements

For information on coordinated measurements and how to use them, refer to the "Coordinated Measurements" chapter in the *Softkey Interface Reference* manual.

## Address/Symbol Entry and Display

You can enter addresses in expressions using physical addresses, logical addresses (segment:offset) or symbols. The method you choose affects the address and symbols displays in your measurements (including memory and trace displays).

## Using Symbols

Suppose you want to display the sample program from chapter 2.

***set symbols on***

***display memory*** Init ***mnemonic***

The emulator uses the SRU symbol database information to find the corresponding address for **Init**, which has a logical address of 0000:0400h (physical address of 400h). The emulator then displays memory starting at that location. Symbols are displayed.

```
 Memory  :mnemonic :file = cmd_rds.S:
   address  label          data
 0000 0400   PROG|Init    B80000      MOV AX,#0000H
 0000 0403                8ED8B80000  MOV DS,AX | MOV AX,#0000H
 0000 0408                8EC08ED0BC  MOV ES,AX | MOV SS,AX | MOV SP,#06F9H
 0000 040F  cmd:Read_Cmd  26C6060006  MOV ES:BYTE PTR COMM|Cmd_Input,#00H
 0000 0415                90          NOP
 0000 0416  cmd_rds:Scan  26A00006    MOV AL,ES:COMM|Cmd_Input
 0000 041A                3C00        CMP AL,#00H
 0000 041C                74F8        JZ P|cmd_rds.S:Scan
 0000 041E  cmd_:Exe_Cmd  3C41        CMP AL,#41H
 0000 0420                7407        JZ |cmd_rds.S:Cmd_A
 0000 0422                3C42        CMP AL,#42H
 0000 0424                740C        JZ |cmd_rds.S:Cmd_B
 0000 0426                E91200      JMP NEAR PTR |cmd_rds.S:Cmd_I
 0000 0429  cmd_rd:Cmd_A  B91200      MOV CX,#0012H
 0000 042C                BE0005      MOV SI,#0500H
 0000 042F                E90F00      JMP NEAR PTR cmd_rd:Write_Msg

STATUS:   8086--Running in monitor______________________________________........
```

## Using Physical Addresses

The emulator keeps a cache of the most recently used logical addresses. This cache is used to interpret physical address entries. When you enter a physical address, the cache is searched for a logical address that corresponds to the physical address. If one is found, it will be used for symbol database lookups.

The cache is initialized with all global symbols in the SRU symbol database. Therefore, symbol lookup will work correctly for all addresses until you enter a segment:offset value that cannot be found in the cache. See “Recovering the Symbol Display” later in this section.

Suppose you enter:

***display memory*** `400h` ***mnemonic***

```
 Memory  :mnemonic :file = cmd_rds.S:
   address  label          data
     00400   PROG|Init    B80000      MOV AX,#0000H
     00403                8ED8B80000  MOV DS,AX | MOV AX,#0000H
     00408                8EC08ED0BC  MOV ES,AX | MOV SS,AX | MOV SP,#06F9H
     0040F  cmd:Read_Cmd  26C6060006  MOV ES:BYTE PTR COMM|Cmd_Input,#00H
     00415                90          NOP
     00416  cmd_rds:Scan  26A00006    MOV AL,ES:COMM|Cmd_Input
     0041A                3C00        CMP AL,#00H
     0041C                74F8        JZ P|cmd_rds.S:Scan
     0041E  cmd_:Exe_Cmd  3C41        CMP AL,#41H
     00420                7407        JZ |cmd_rds.S:Cmd_A
     00422                3C42        CMP AL,#42H
     00424                740C        JZ |cmd_rds.S:Cmd_B
     00426                E91200      JMP NEAR PTR |cmd_rds.S:Cmd_I
     00429  cmd_rd:Cmd_A  B91200      MOV CX,#0012H
     0042C                BE0005      MOV SI,#0500H
     0042F                E90F00      JMP NEAR PTR cmd_rd:Write_Msg

STATUS:   8086--Running user program    Emulation trace complete________........
```

The cache is searched for any logical addresses which match the physical address of 400h. Since you entered the symbol **Init** earlier, which had a logical address of 0000h:0400h, a match is found, which makes the symbol lookup possible. The memory locations are displayed starting at physical address 400h. Note that physical addresses are displayed in the address column, rather than the segment:offset representation.

## Using Segment:Offset

If you prefer to enter addresses directly in segment:offset form, you can do so. For example:

***display memory*** 0000h:0400h ***mnemonic***

This displays the same memory locations as before. Symbol information is displayed, since the segment:offset matches the address assigned to the label **Init** in the symbols database. The address column returns to segment:offset display.

```
 Memory  :mnemonic :file = cmd_rds.S:
   address  label          data
 0000 0400   PROG|Init    B80000      MOV AX,#0000H
 0000 0403                8ED8B80000  MOV DS,AX | MOV AX,#0000H
 0000 0408                8EC08ED0BC  MOV ES,AX | MOV SS,AX | MOV SP,#06F9H
 0000 040F  cmd:Read_Cmd  26C6060006  MOV ES:BYTE PTR COMM|Cmd_Input,#00H
 0000 0415                90          NOP
 0000 0416  cmd_rds:Scan  26A00006    MOV AL,ES:COMM|Cmd_Input
 0000 041A                3C00        CMP AL,#00H
 0000 041C                74F8        JZ P|cmd_rds.S:Scan
 0000 041E  cmd_:Exe_Cmd  3C41        CMP AL,#41H
 0000 0420                7407        JZ |cmd_rds.S:Cmd_A
 0000 0422                3C42        CMP AL,#42H
 0000 0424                740C        JZ |cmd_rds.S:Cmd_B
 0000 0426                E91200      JMP NEAR PTR |cmd_rds.S:Cmd_I
 0000 0429  cmd_rd:Cmd_A  B91200      MOV CX,#0012H
 0000 042C                BE0005      MOV SI,#0500H
 0000 042F                E90F00      JMP NEAR PTR cmd_rd:Write_Msg

STATUS:   8086--Running user program    Emulation trace complete________........
```

On the 8018X series processors, many different segment:offset combinations can produce the same physical address. For example:

***display memory*** `003fh:0010h` ***mnemonic***

```
Memory  :mnemonic
  address  label          data
003F 0010                 B80000      MOV AX,#0000H
003F 0013                 8ED8B80000  MOV DS,AX | MOV AX,#0000H
003F 0018                 8EC08ED0BC  MOV ES,AX | MOV SS,AX | MOV SP,#06F9H
003F 001F                 26C6060006  MOV ES:BYTE PTR 0600H,#00H
003F 0025                 90          NOP
003F 0026                 26A00006    MOV AL,ES:0600H
003F 002A                 3C00        CMP AL,#00H
003F 002C                 74F8        JZ 0026H
003F 002E                 3C41        CMP AL,#41H
003F 0030                 7407        JZ 0039H
003F 0032                 3C42        CMP AL,#42H
003F 0034                 740C        JZ 0042H
003F 0036                 E91200      JMP NEAR PTR 004BH
003F 0039                 B91200      MOV CX,#0012H
003F 003C                 BE0005      MOV SI,#0500H
003F 003F                 E90F00      JMP NEAR PTR 0051H

STATUS:   8086--Running in monitor______________________________________........
```

The same memory locations are displayed as before. But, there is no corresponding symbol entry for any of the segment:offset addresses, so no symbols are displayed.

Remember that the cache stores the address translations in a most recently used order. If you now enter a physical address of 400h, the cache matches it with a segment:offset of 003fh:0010h. Symbols will not be displayed.

**Note**

The size of the cache is limited only by system memory.

## Recovering the Symbol Display

If you lose the symbol display, you entered an address in a form that can't be translated to anything matching the available symbols:

- A physical address for which no corresponding logical address exists in the translation cache.
- A segment:offset value having no corresponding symbol value in the symbol database.

To recover the symbol display, enter addresses with the proper segment:offset, or use the symbol itself. Or, you can reload the symbol database with the **load symbols** command to reinitialize the address translation cache for all global symbols.

## 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 microprocessor (in this case, an 8086 or 8088) executes the monitor program.

The monitor program makes possible emulation commands to 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.

When the emulation system controller sees an emulation command that 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, etc.)

The primary 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 from target system or emulation memory. (It may operate from memory reserved for the monitor when a foreground monitor is selected.)

### Background

In background mode, 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 suspended to the target system. In background mode, the emulation microprocessor executes from memory reserved for the monitor.

### Modes in Which the Foreground Monitor Operates

The foreground monitor operates in both background and foreground. When a 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 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.

## Loading Foreground Monitors Larger than 2K Bytes

Two kilobytes of emulation memory are reserved for the monitor program. It is possible to use custom foreground monitors that are greater than 2 kilobytes in length. You must take special steps:

1. Do NOT configure the emulator to enter the monitor after emulator configuration.

2. After configuration, reload the monitor program (as you would a normal program).

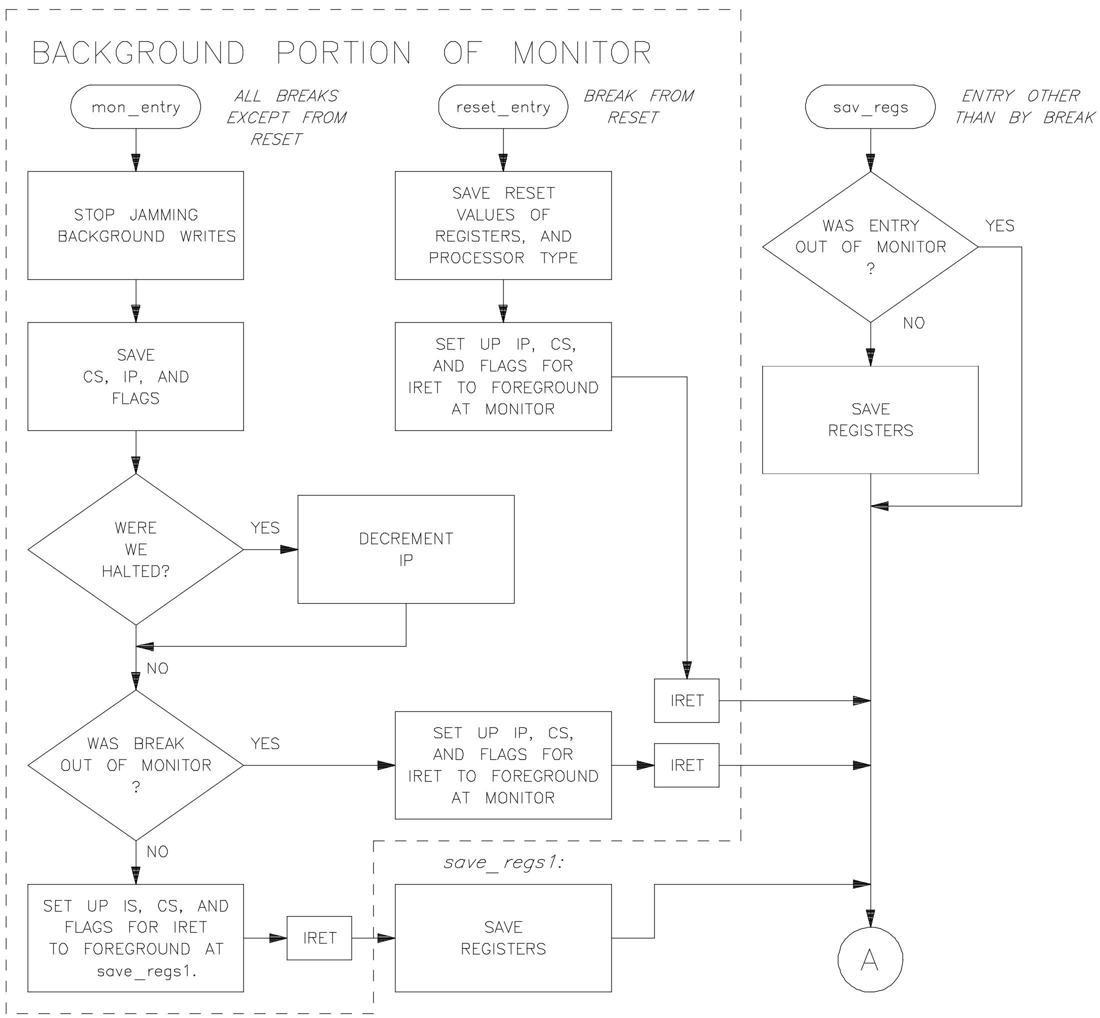
When you specify a foreground monitor name during emulator configuration, the configuration process loads only the 2 kilobytes of memory reserved for the monitor.

## Listing

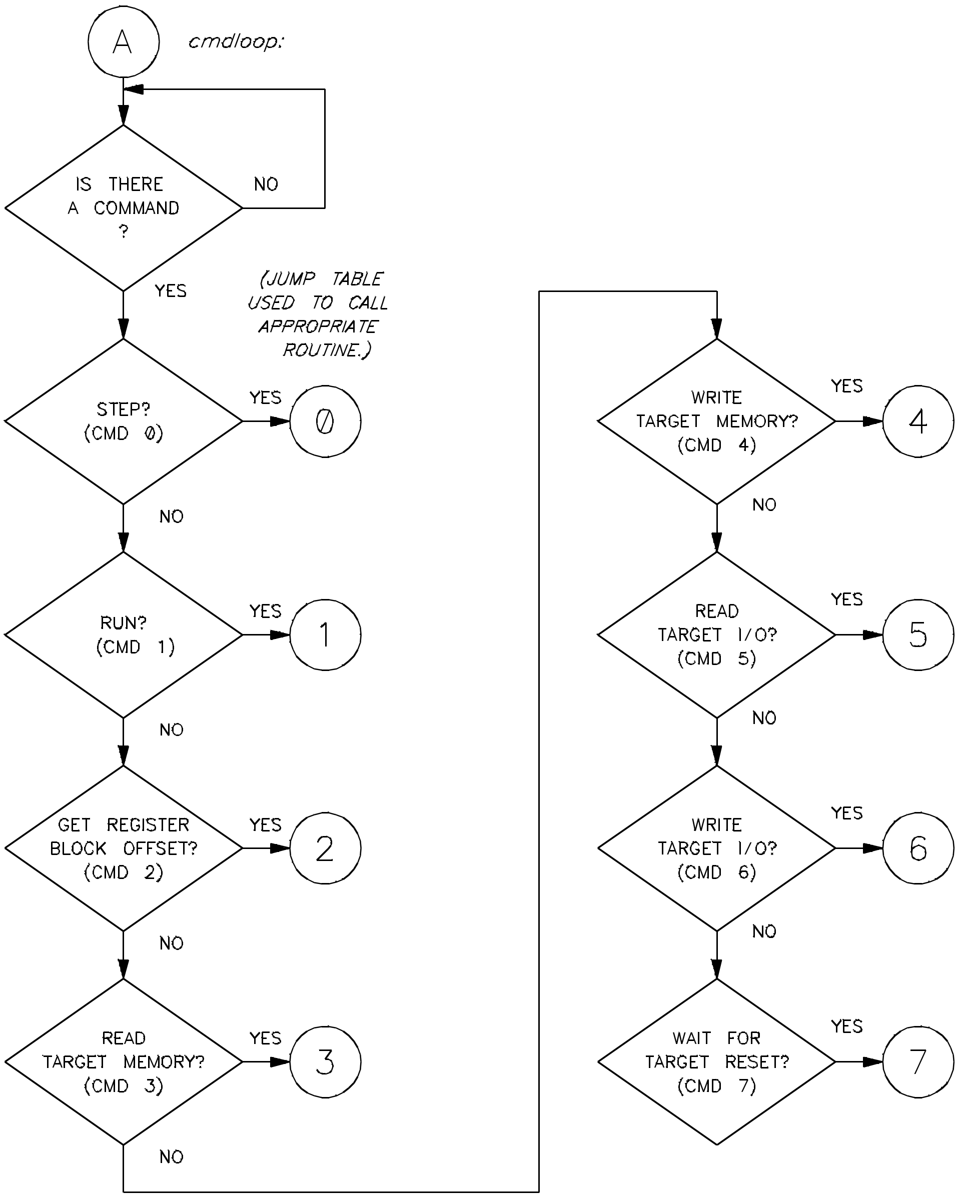
The foreground monitor is resident in the emulator, and it may be selected without having to load any code. The foreground monitor comes with the Softkey Interface so that you may customize it, if necessary. Refer to the foreground monitor source file for the latest listing of the monitor. The foreground monitor can be copied from the following location.

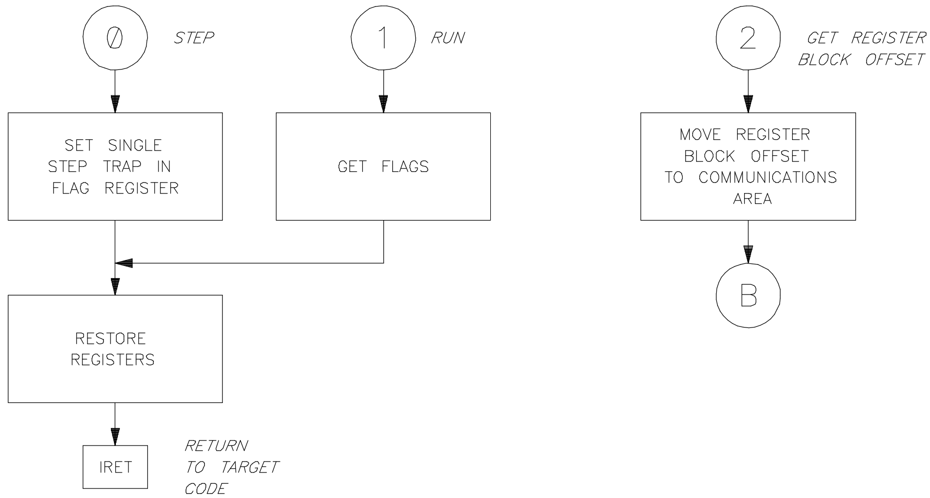
**/usr/hp64000/monitor/fmon8086.S** (8086)
**/usr/hp64000/monitor/fmon8088.S** (8088)

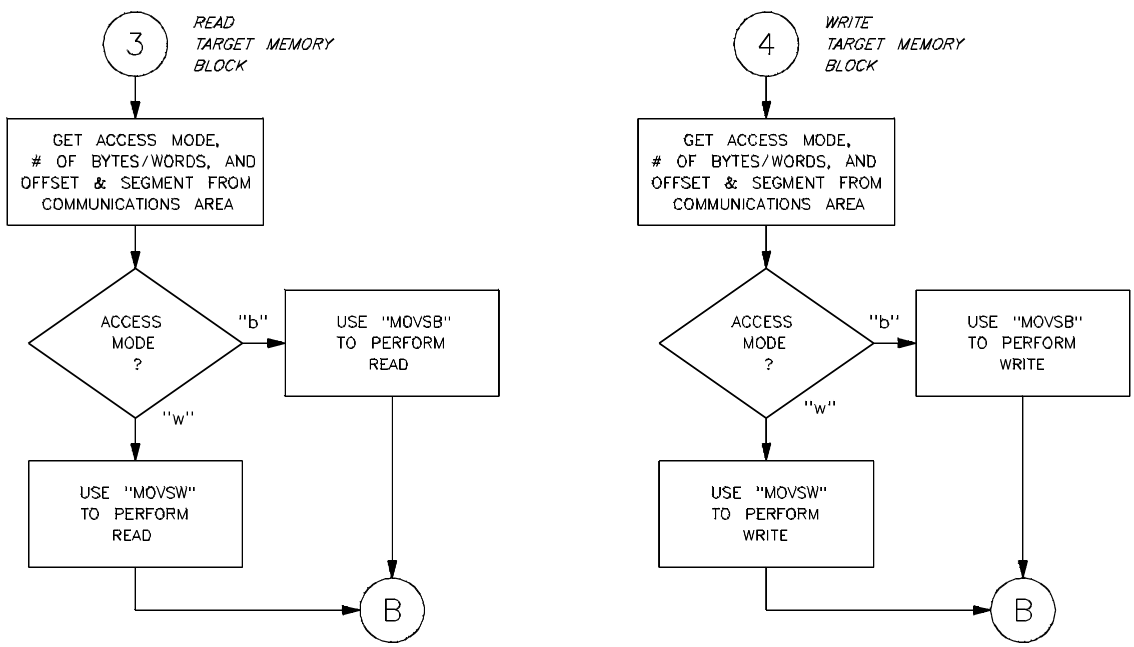
## Flowchart

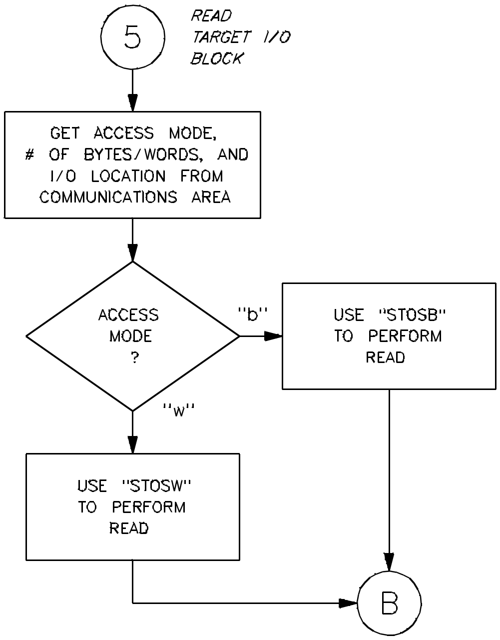


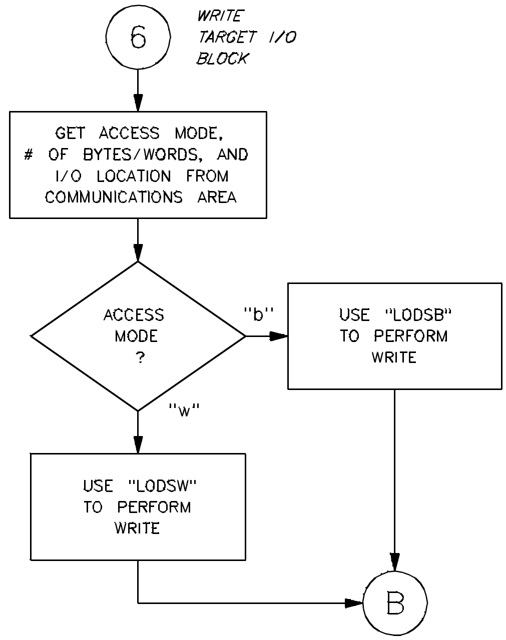
BACKGROUND PORTION OF MONITOR
mon_entry
ALL BREAKS EXCEPT FROM RESET
reset_entry
BREAK FROM RESET
sav_regs
ENTRY OTHER THAN BY BREAK
STOP JAMMING BACKGROUND WRITES
SAVE RESET VALUES OF REGISTERS, AND PROCESSOR TYPE
WAS ENTRY OUT OF MONITOR ?
YES
NO
SAVE CS, IP, AND FLAGS
SET UP IP, CS, AND FLAGS FOR 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

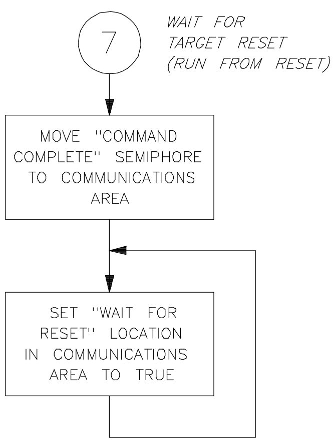

A
cmdloop:
IS THERE A COMMAND ?
NO
YES
(JUMP TABLE USED TO CALL APPROPRIATE ROUTINE.)
STEP? (CMD 0)
YES
0
NO
RUN? (CMD 1)
YES
1
NO
GET REGISTER BLOCK OFFSET? (CMD 2)
YES
2
NO
READ TARGET MEMORY? (CMD 3)
YES
3
NO
WRITE TARGET MEMORY? (CMD 4)
YES
4
NO
READ TARGET I/O? (CMD 5)
YES
5
NO
WRITE TARGET I/O?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

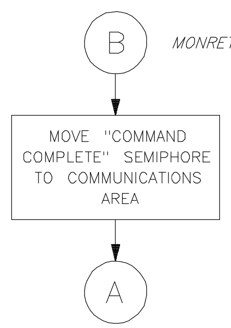

0
STEP
SET SINGLE STEP TRAP IN FLAG REGISTER
1
RUN
GET FLAGS
RESTORE REGISTERS
IRET
RETURN TO TARGET CODE
2
GET REGISTER BLOCK OFFSET
MOVE REGISTER BLOCK OFFSET TO COMMUNICATIONS AREA
B


3
READ TARGET MEMORY BLOCK
GET ACCESS MODE, # OF BYTES/WORDS, AND OFFSET & SEGMENT FROM COMMUNICATIONS AREA
ACCESS MODE ?
"b"
USE "MOVSB" TO PERFORM READ
"w"
USE "MOVSW" TO PERFORM READ
B
4
WRITE TARGET MEMORY BLOCK
GET ACCESS MODE, # OF BYTES/WORDS, AND OFFSET & SEGMENT FROM COMMUNICATIONS AREA
ACCESS MODE ?
"b"
USE "MOVSB" TO PERFORM WRITE
"w"
USE "MOVSW" TO PERFORM WRITE
B


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

# Index