HP 64739

H8/536 Emulator Softkey Interface

User's Guide

HEWLETT PACKARD

HP Part No. 64739-97005
Printed in U.S.A.
February 1994

Edition 2

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 **64739-97002, February 1991**

Edition 2 **64739-97005, February 1994**

Using This Manual

This manual introduces you to the following emulators as used with the Softkey Interface.

- HP 64739A H8/536 emulator
- HP 64739B H8/536S emulator

Throughout this documentation, the following names are used to denote the microprocessors listed in the following table of supported microprocessors.

Model	Supported Microprocessors	Reffered to as
HP 64739A(H8/536 emulator)	HD6475368CP HD6435368CP HD6475348CP HD6435348CP	H8/536 H8/536 H8/534 H8/534
HP 64739B(H8/536S emulator)	HD6475368CP HD6435368CP HD6475348CP HD6435348CP HD6475368SCP HD6435368SCP HD6475348SCP HD6435348SCP	H8/536 H8/536 H8/534 H8/534 H8/536S H8/536S H8/534S H8/534S

For the most part, the H8/536 and H8/536S emulators all operate the same way. Differences of between the emulators are described where they exist. Both the H8/536 and H8/536S emulators will be referred to as the "H8/536 emulator". In the specific instances where H8/536S emulator differs from H8/536 emulator, it will be described as "H8/536S emulator".

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.

This manual will not:

- tell you how to use each and every emulator/analyzer command (refer to the *User's Reference* manual)

Organization

Chapter 1 **Introduction to the H8/536 Emulator.** This chapter briefly introduces you to the concept of emulation and lists the basic features of the H8/536 emulator.

Chapter 2 **Getting Started.** This chapter shows you how to use emulation commands by executing them on a sample program. This chapter describes the sample program and how to: load programs into the emulator, map memory, display and modify memory, display registers, step through programs, run programs, set software breakpoints, search memory for data, and use the analyzer.

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

Chapter 4 **Configuring the Emulator.** This chapter shows you how to restrict the emulator to real-time execution, select a target system clock source, allow background cycles to be seen by the target system.

Chapter 5 **Using the Emulator.** This chapter describes emulation topics which are not covered in the "Getting Started" chapter.

Appendix A **Using the Foreground Monitor.** This appendix describes the advantages and disadvantages of foreground and background monitors and how to use foreground monitors.

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.

Contents

4 Configuring the Emulator

5 Using the Emulator

Illustrations

Tables

Notes

1

Introduction to the H8/536 Emulator

Introduction

The topics in this chapter include:

- Purpose of the H8/536 emulator.
- Features of the H8/536 emulator.

Purpose of the H8/536 Emulator

The H8/536 emulator is designed to replace the H8/536 microprocessor in your target system to help you debug/integrate target system software and hardware. The emulator performs just like the processor which it replaces, but at the same time, it gives you information about the bus cycle operation of the processor. The emulator gives you control over target system execution and allows you to view or modify the contents of processor registers, target system memory.

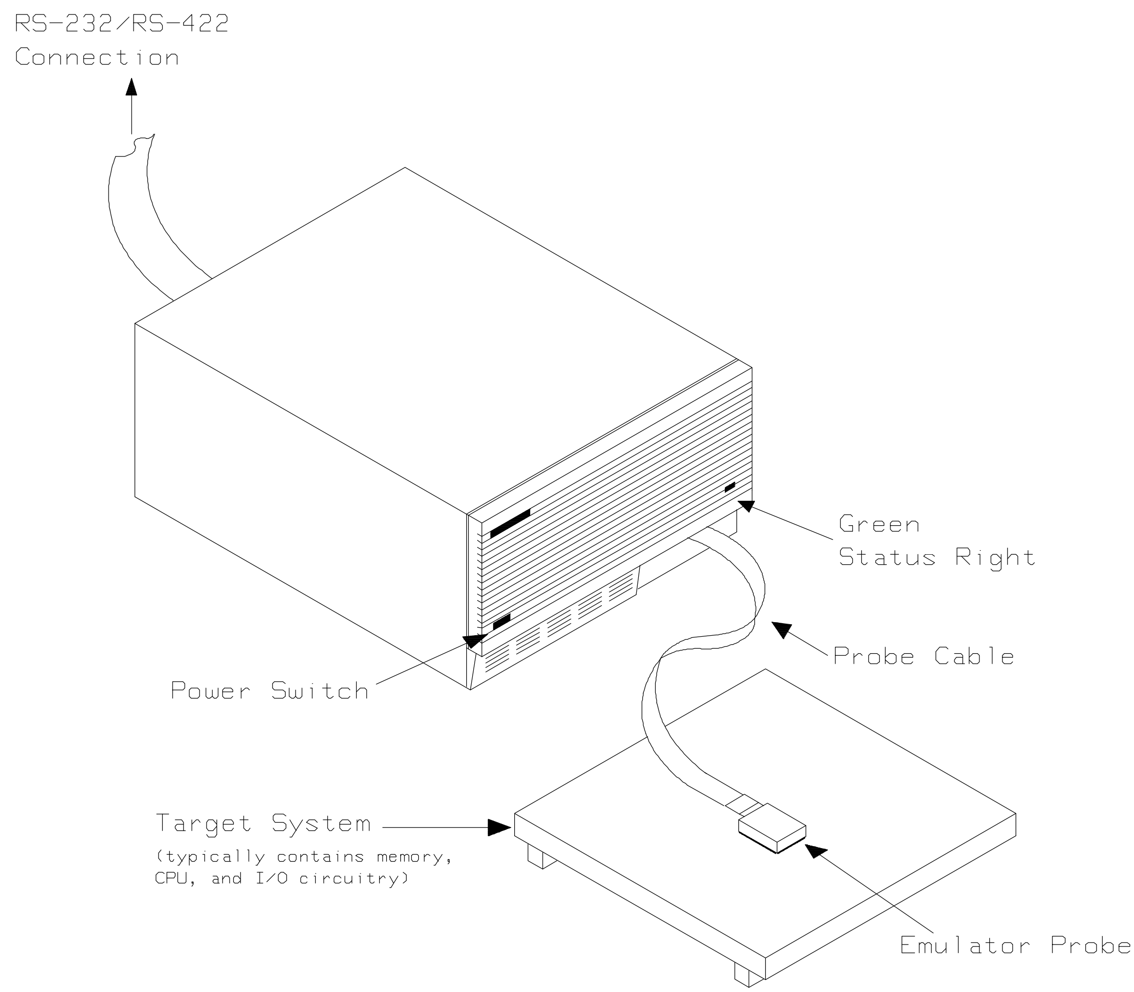

Figure 1-1. HP 64739 Emulator for the H8/536 Emulator

Features of the H8/536 Emulator

This section introduces you to the features of the emulator. The chapters which follow show you how to use these features.

Supported Microprocessors

The H8/536 emulator supports the microprocessors listed in Table 1-1.

Table 1-1. Supported Microprocessors

Model	Supported Microprocessors	Reffered to as
HP 64739A(H8/536 emulator)	HD6475368CP HD6435368CP HD6475348CP HD6435348CP	H8/536 H8/536 H8/534 H8/534
HP 64739B(H8/536S emulator)	HD6475368CP HD6435368CP HD6475348CP HD6435348CP HD6475368SCP HD6435368SCP HD6475348SCP HD6435348SCP	H8/536 H8/536 H8/534 H8/534 H8/536S H8/536S H8/534S H8/534S

Clock Speeds

You can select whether the emulator will be clocked by the internal clock source or by the external clock source on your target system. You must use a clock input conforming to the specification of Table 1-2.

When you use an external crystal, you need to input conforming to the specification of microprocessor.

Table 1-2. Clock Speeds

Clock source	Model	Microprocessor	Clock Speed
Internal	HP 64739A (H8/536 emulator)	H8/536 H8/534	10MHz (System clock)
	HP 64739B (H8/536S emulator)	H8/536 H8/534 H8/536S H8/534S	10MHz (System clock)
External	HP 64739A (H8/536 emulator)	H8/536 H8/534	From 0.5 up to 10MHz (System clock)
	HP 64739B (H8/536S emulator)	H8/536 H8/534	From 0.5 up to 10MHz (System clock)
		H8/536S H8/534S	From 0.5 up to 16MHz (System clock)

Emulation memory

The H8/536 emulator is used with one of the following Emulation Memory Cards.

- HP 64726A 128K byte Emulation Memory Card
- HP 64727A 512K byte Emulation Memory Card
- HP 64728A 1M byte Emulation Memory Card

You can define up to 16 memory ranges (at 256 byte boundaries and least 256 byte in length.) The emulator occupies 2K byte, which is used for monitor program, leaving 126K, 510K, 1022K byte of emulation memory which you may use. You can characterize memory range as emulation RAM (eram), emulation ROM (erom), target system RAM (tram), target system ROM (trom), or guarded memory (grd). The emulator generates an error message when accesses are made to guarded memory locations. You can also configure the emulator so that writes to memory defined as ROM cause emulator execution to break out of target program execution.

Analysis

The H8/536 emulator is used with one of the following analyzers which allows you to trace code execution and processor activity.

- HP 64704A 80-channel Emulation Bus Analyzer
- HP 64703A 64-channel Emulation Bus Analyzer and 16-channel State/Timing Analyzer.
- HP 64794x 80-channel 8K/64K/256K Emulation Bus Analyzer.

The Emulation Bus Analyzer monitors the emulation processor using an internal analysis bus. The HP 64703A 64-channel Emulation Bus Analyzer and 16-channel State/Timing Analyzer allows you to probe up to 16 different lines in your target system.

Registers

You can display or modify the H8/536 internal register contents. This includes the ability to modify the program counter (PC) and code page register (CP) so you can control where the emulator begins executing a target system program.

Single-Step

You can direct the emulation processor to execute a single instruction or a specified number of instructions.

Target System Interface

You can set the interface to the target system to be active or passive during background monitor operation. (See the "Emulator Pod Configuration" section of the "Configuring the Emulator" chapter for further details.)

Breakpoints

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

You can also define software breakpoints in your program. The emulator uses one of H8/536 undefined opcode (1B hex) as software breakpoint interrupt instruction. When you define a software breakpoint, the emulator places the breakpoint interrupt instruction (1B hex) at the specified address; after the breakpoint interrupt instruction causes emulator execution to break out of your program, the emulator replaces the original opcode. Refer to the "Using Software Breakpoints" section of "Getting Started" chapter for more information.

Reset Support

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

Foreground or Background Emulation Monitor

The emulation monitor is a program that is executed by the emulation processor. It allows the emulation controller to access target system resources. For example, when you display target system memory, it is the monitor program that executes H8/536 instructions which 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 can also execute in *background.* The emulator mode in which foreground operation is suspended so that emulation processor can be used to access target system resources. The background monitor does not occupy processor address space.

Real-Time Execution

Real-time execution signifies continuous execution of your program without interference from the emulator. (Such interference occurs when the emulator temporarily breaks into the monitor so that it can access register contents or target system memory.)

Emulator features performed in real time include: running and analyzer tracing.

Emulator features not performed in real time include: display or modify of target system memory; load/dump of any memory, display or modification of registers, and single step.

Limitations, Restrictions

DMA Support

Direct memory access to H8/536 emulation memory is not permitted.

Sleep and Software Stand-by Mode

When the emulator breaks into the emulation monitor, H8/536 microprocessor sleep or software stand-by mode is released and comes to normal processor mode.

Watch Dog Timer in Background

Watch dog timer suspends count up while the emulator is running in background monitor.

RAM Enable Bit

The internal RAM of H8/536 processor can be enabled/disabled by RAME (RAM enable bit). However, once you map the internal RAM area to emulation RAM, the emulator accesses emulation RAM even if the internal RAM is disabled by RAME.

Notes

2

Getting Started

Introduction

This chapter will lead you through a basic, step by step tutorial designed to familiarize you with the use of the HP 64739 emulator with the Softkey Interface.

This chapter will:

- Tell you what must be done before you can use the emulator as shown in the tutorial examples.
- Describe the sample program used for this chapter's example.

This chapter will show you how to:

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

Before You Begin

Prerequisites

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

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

2. Installed 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 avoid questions later.

   You should read the *Softkey Interface Reference* manual to learn how to use the Softkey Interface in general. For the most part, this manual contains information specific to the H8/536 emulator.

A Look at the Sample Program

The sample program used in this chapter is listed in figure 2-1. The program emulates 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/hp64739/cmd_rds.src

Data Declarations

The "Table" section defines the messages used by the program to respond to various command inputs. These messages are labeled **Msg_A**,**Msg_B**, and **Msg_I**.

```
                .GLOBAL     Init,Msgs,Cmd_Input
                .GLOBAL     Msg_Dest

                .SECTION    Table,DATA
Msgs
Msg_A           .SDATA      "Command A entered "
Msg_B           .SDATA      "Entered B command "
Msg_I           .SDATA      "Invalid Command "
End_Msgs

                .SECTION    Prog,CODE
;***********************************************
;* Sets up the stack pointer.
;***********************************************
Init            MOV:G.W     #Stack,R7
;***********************************************
;* Clear previous command.
;***********************************************
Read_Cmd        MOV:G.B     #0,@Cmd_Input
;***********************************************
;* Read command input byte.  If no command has
;* been entered, continue to scan for input.
;***********************************************
Scan            MOV:G.B     @Cmd_Input,R0
                BEQ         Scan
;***********************************************
;* A command has been entered.  Check if it is
;* command A, command B, or invalid.
;***********************************************
Exe_Cmd         CMP:E.B     #H'41,R0
                BEQ         Cmd_A
                CMP:E.B     #H'42,R0
                BEQ         Cmd_B
                BRA         Cmd_I
;***********************************************
;* Command A is entered.  R1 = the number of
;* bytes in message A.  R4 = location of the
;* message.  Jump to the routine which writes
;* the messages.
;***********************************************
Cmd_A           MOV:I.W     #Msg_B-Msg_A-1,R1
                MOV:I.W     #Msg_A,R4
                BRA         Write_Msg
;***********************************************
;* Command B is entered.
;***********************************************
Cmd_B           MOV:I.W     #Msg_I-Msg_B-1,R1
                MOV:I.W     #Msg_B,R4
                BRA         Write_Msg
;***********************************************
;* An invalid command is entered.
;***********************************************
Cmd_I           MOV:I.W     #End_Msgs-Msg_I-1,R1
                MOV:I.W     #Msg_I,R4
;***********************************************
```

Figure 2-1. Sample Program Listing

```
;* Message is written to the destination.
;***********************************************
Write_Msg       MOV:I.W     #Msg_Dest,R5
Again           MOV:G.B     @R4+,R3
                MOV:G.B     R3,@R5+
                SCB/EQ      R1,Again
;***********************************************
;* The rest of the destination area is filled
;* with zeros.
;***********************************************
Fill_Dest       MOV:G.B     #0,@R5+
                CMP:I.W     #Msg_Dest+H'20,R5
                BNE         Fill_Dest
;***********************************************
;* Go back and scan for next command.
;***********************************************
                BRA         Read_Cmd

                .SECTION    Data,COMMON
;***********************************************
;* Command input byte.
;***********************************************
Cmd_Input       .RES.B      1
                .RES.B      1
;***********************************************
;* Destination of the command messages.
;***********************************************
Msg_Dest        .RES.W      H'3E
Stack           .RES.W      1       ; Stack area.
                .END        Init
```

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

Initialization

The program instruction at the **Init** label initializes the stack pointer.

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 see if a command is entered (a value other than 0 hex).

Processing Commands

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

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

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

If the command input byte is neither "A" nor "B", an invalid command has been entered, and execution is transferred to the instructions at **Cmd_I**.

The instructions at **Cmd_A**, **Cmd_B**, and **Cmd_I** each load register R1 with the length of the message to be displayed and register R4 with the starting location of the appropriate message. Then, execution transfers to **Write_Msg** which writes the appropriate message to the destination location, **Msg_Dest**.

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

The Destination Area

The "Data" section declares memory storage for the command input byte, the destination area, and the stack area.

This program emulates a primitive command interpreter.

Sample Program Assembly

The sample program is written for and assembled with the HP 64869 H8/500 Assembler/Linkage Editor. The sample program was assembled with the following command below(which assumes that **/usr/hp64000/bin** is defined in the PATH environment variable).

```
$ h8asm -debug cmd_rds.src <RETURN>
```

Linking the Sample Program

The sample program can be linked with following command and generates the absolute file. The contents of "cmd_rds.k" linkage editor subcommand file is shown in figure 2-2.

```
$ h8lnk -subcommand=cmd_rds.k <RETURN>
```

```
debug
input cmd_rds
start Prog(1000),Table(2000),Data(0FE00)
output cmd_rds
exit
```

Figure 2-2. Linkage Editor Subcommand File

Generate HP Absolute file

To generate HP Absolute file for the Softkey Interface, you need to use **"h8cnvhp"** absolute file format converter program. The h8cnvhp converter is provided with HP 64869 H8/500 Assembler/Linkage Editor. To generate HP Absolute file, enter following command:

```
$ h8cnvhp cmd_rds <RETURN>
```

You will see that cmd_rds.X, cmd_rds.L, and cmd_rds.A are generated. These are sufficient throughout this chapter.

Note

You need to specify "debug" command line option to both assembler and linker command to generate local symbol information. The "debug" option for the assembler and linker direct to include local symbol information to the object file.

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.

From the "pmon" User Interface

If **/usr/hp64000/bin** is specified in your PATH environment variable, you can enter the **pmon** User Interface with the following command.

$ **pmon** <RETURN>

If you have not already created a measurement system for the H8/536 emulator, you can do so with the following commands. First you must initialize the measurement system with the following command.

MEAS_SYS msinit <RETURN>

After the measurement system has been initialized, enter the configuration interface with the following command.

msconfig <RETURN>

To define a measurement system for the H8/536 emulator, enter:

make_sys emh8 <RETURN>

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

add <module_number> ***naming_it*** h8 <RETURN>

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

end <RETURN>

If the measurement system and emulation module are named "emh8" and "h8" as shown above, you can enter the emulation system with the following command:

emh8 default h8 <RETURN>

If this command is successful, you will see a display similar to figure 2-3. The status message shows that the default configuration file has been loaded. If the command is not successful, you will be given an error message and returned to the **pmon** User Interface. Error messages are described in the *Softkey Interface Reference* manual.

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

From the HP-UX Shell

If **/usr/hp64000/bin** is specified in your PATH environment variable, you can also 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).

```
                         HP64739-19001 A.03.00 01Apr91
                         H8/536 EMULATION SERIES 64700

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

  All Rights Reserved. Reproduction, adaptation, or translationwithout prior
  written  permission  is prohibited, except as allowed undercopyright laws.

                           RESTRICTED RIGHTS LEGEND

     Use , duplication , or disclosure  by the  Government is subject to
     restrictions as set forth in subparagraph (c) (1) (II) ofthe Rights
     in Technical Data and Computer Software clause at DFARS52.227-7013.
     HEWLETT-PACKARD Company , 3000 Hanover St. , Palo Alto, CA94304-1181


STATUS:   Loaded configuration file____________________________________...R....



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

Figure 2-3. Softkey Interface Display

If this command is successful, you will see a display similar to figure 2-3. The status message shows that the default configuration file has been loaded. If the command is not successful, you will be given an error message and returned to the HP-UX prompt. Error messages are described in the *Softkey Interface Reference* manual.

Using the Default Configuration

The default emulator configuration is used with the following examples.

The address range 0 hex through F5FF hex is mapped as emulation ROM, and F600 hex through FEFF hex as emulation RAM.

The emulator emulates the H8/536 processor (rather than H8/534) using the background monitor.

On-Line Help

There are two ways to access on-line help in the Softkey Interface. The first is by using 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 cause information on that topic to be displayed on your screen. For example, you can enter the following command to access "system command" help information.

`?` ***`system_commands`*** `<RETURN>`

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

```
---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%)
```

Pod Command Help

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

display pod_command <RETURN>
pod_command ’help m’ <RETURN>

The command enclosed in string delimiters (", ’, or ^) is any Terminal Interface command, and the output of that command 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 of the emulator configuration options (as the example command above shows).

```
Pod Commands
  Time                Command
10:00:00 help m

  m - display or modify processor memory space
    m <addr>               - display memory at address
    m -d<dtype> <addr>     - display memory at address with display option
    m <addr>..<addr>       - display memory in specified address range
    m -dm <addr>..<addr>   - display memory mnemonics in specified range
    m <addr>..             - display 128 byte block starting at address A
    m <addr>=<value>       - modify memory at address to <value>
    m -d<dtype> <addr>=<value>       - modify memory with display option
    m <addr>=<value>,<value>         - modify memory to data sequence
    m <addr>..<addr>=<value>,<value> - fill range with repeating sequence
  --- VALID <dtype> MODE OPTIONS ---
    b - display size is 1 byte(s)
    w - display size is 2 byte(s)
    m - display processor mnemonics

STATUS:   H8/536-Running in monitor____________________________________........
pod_command 'help m'


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

Loading Absolute Files

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

load `cmd_rds <RETURN>`

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

Displaying Symbols

When you load an absolute file into memory (unless you use the "nosymbols" option), symbol information is loaded. Both global symbols and symbols that are local to a source file can be displayed.

Global

To display global symbols, enter the following command.

display global_symbols <RETURN>

Listed are: address ranges associated with a symbol and the offset of the symbol within the minimum value of these global symbols.

```
Global symbols in cmd_rds
Static symbols
Symbol name                Address range     Contents   Segment         Offset
Cmd_Input                  0FE00                                             EE00
Init                       01000                                             0000
Msg_Dest                   0FE02                                             EE02
Msgs                       02000                                             1000

Filename symbols
Filename
cmd_rds.src









STATUS:   H8/536--Running in monitor___________________________________...R....
display global_symbols


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

Local 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.src: <RETURN>

Listed are: address ranges associated with a symbol and the offset of that symbol within the start address of the section that the symbol is associated with.

```
Symbols in cmd_rds.src:
Static symbols
Symbol name                 Address range     Contents   Segment         Offset
Again                       01032                                             0032
Cmd_A                       01019                                             0019
Cmd_B                       01021                                             0021
Cmd_I                       01029                                             0029
Cmd_Input                   0FE00                                             0000
Data                        0FE00                                             0000
Exe_Cmd                     0100F                                             000F
Fill_Dest                   01039                                             0039
Init                        01000                                             0000
Msg_A                       02000                                             0000
Msg_B                       02012                                             0012
Msg_Dest                    0FE02                                             0002
Msg_I                       02024                                             0024
Msgs                        01000                                             0000
Prog
STATUS:   H8/536--Running in monitor___________________________________...R....
display  local_symbols_in cmd_rds.src:


  load    store   stop-trc   copy           reset   specify cmb_exec     ---ETC--
```

Displaying Memory in Mnemonic Format

You can display, in mnemonic format, the absolute code in memory. For example to display the memory of the "cmd_rds" program,

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

Notice that you can use symbols when specifying expressions. The global symbol **Init** is used in the command above to specify the starting address of the memory to be displayed.

```
 Memory  :mnemonic :file = cmd_rds.src:
   address   data
-----------------------------------------------------------------    --------------
      01000  0CFE7E87    MOV:G.W #FE7E,R7
      01004  15FE000600  MOV:G.B #00,@FE00
      01009  15FE0080    MOV:G.B @FE00,R0
      0100D  27FA        BEQ 01009
      0100F  4041        CMP:E.B #41,R0
      01011  2706        BEQ 01019
      01013  4042        CMP:E.B #42,R0
      01015  270A        BEQ 01021
      01017  2010        BRA 01029
      01019  590011      MOV:I.W #0011,R1
      0101C  5C2000      MOV:I.W #2000,R4
      0101F  200E        BRA 0102F
      01021  590011      MOV:I.W #0011,R1
      01024  5C2012      MOV:I.W #2012,R4
      01027  2006        BRA 0102F
      01029  59000F      MOV:I.W #000F,R1

STATUS:   H8/536--Running in    monitor___________________________________...R....
display  memory Init mnemonic


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

Display Memory with Symbols

If you want to see symbol information with displaying memory in mnemonic format, the H8/536 emulator Softkey Interface provides "set symbols" command. To see symbol information, enter the following command.

set symbols on <RETURN>

As you can see, the memory display shows symbol information.

```
 Memory  :mnemonic :file = cmd_rds.src:
   address  label          data
--------------------------------------------------------------------------------
     01000     :Init      0CFE7E87    MOV:G.W #FE7E,R7
     01004  cmd:Read_Cmd  15FE000600  MOV:G.B #00,@FE00
     01009  cmd_rds:Scan  15FE0080    MOV:G.B @FE00,R0
     0100D                27FA        BEQ cmd_rds.src:Scan
     0100F  cmd_:Exe_Cmd  4041        CMP:E.B #41,R0
     01011                2706        BEQ cmd_rds.sr:Cmd_A
     01013                4042        CMP:E.B #42,R0
     01015                270A        BEQ cmd_rds.sr:Cmd_B
     01017                2010        BRA cmd_rds.sr:Cmd_I
     01019  cmd_rd:Cmd_A  590011      MOV:I.W #0011,R1
     0101C                5C2000      MOV:I.W #2000,R4
     0101F                200E        BRA cmd_rd:Write_Msg
     01021  cmd_rd:Cmd_B  590011      MOV:I.W #0011,R1
     01024                5C2012      MOV:I.W #2012,R4
     01027                2006        BRA cmd_rd:Write_Msg
     01029  cmd_rd:Cmd_I  59000F      MOV:I.W #000F,R1

STATUS:   H8/536--Running in monitor___________________________________...R....
set symbols on


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

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 specifies that the emulator start executing at a previously defined "start address". Transfer addresses are defined in assembly language source files with the .END assembler directive (i.e., pseudo instruction). For example, the sample program defines the address of the label **Init** as the transfer address. The following command will cause the emulator to execute from 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 "Running From Reset" section in the "In-Circuit Emulation" chapter).

Displaying Memory Repetitively

You can display memory locations repetitively so that the information on the screen is constantly updated. 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 simulates a primitive command interpreter. Commands are sent to the sample program through a byte sized memory location labeled **Cmd_Input**. You can use the modify memory feature to send a command to the sample program. For example, to enter the command "A" (41 hex), use the following command.

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

Or:

modify memory Cmd_Input ***strings 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
          0FE02-09    43   6F   6D   6D   61   6E   64   20       C o m     m  a n d
          0FE0A-11    41   20   65   6E   74   65   72   65       A   e     n  t e r e
          0FE12-19    64   20   00   00   00   00   00   00       d   .     .  . . . .
          0FE1A-21    00   00   00   00   00   00   00   00       . . .     .  . . . .
          0FE22-29    00   00   00   00   00   00   00   00       . . .     .  . . . .
          0FE2A-31    00   00   00   00   00   00   00   00       . . .     .  . . . .
          0FE32-39    00   00   00   00   00   00   00   00       . . .     .  . . . .
          0FE3A-41    00   00   00   00   00   00   00   00       . . .     .  . . . .
          0FE42-49    00   00   00   00   00   00   00   00       . . .     .  . . . .
          0FE4A-51    00   00   00   00   00   00   00   00       . . .     .  . . . .
          0FE52-59    00   00   00   00   00   00   00   00       . . .     .  . . . .
          0FE5A-61    00   00   00   00   00   00   00   00       . . .     .  . . . .
          0FE62-69    00   00   00   00   00   00   00   00       . . .     .  . . . .
          0FE6A-71    00   00   00   00   00   00   00   00       . . .     .  . . . .
          0FE72-79    00   00   00   00   00   00   00   00       . . .     .  . . . .
          0FE7A-81    00   00   00   00   00   00   00   00       . . .     .  . . . .

     STATUS:   H8/536--Running user program_________________________________...R....
     modify  memory Cmd_Input  bytes  to 41h


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

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 provided with one of H8/536 undefined opcode (1B hex) as breakpoint interrupt instruction. When you define or enable a software breakpoint, the emulator will replace the opcode at the software breakpoint address with the breakpoint interrupt instruction.

When software breakpoints are enabled and emulator detects the breakpoint interrupt instruction (1B hex), it generates a break to background request which as with the "processor break" command. Since the system controller knows the locations of defined software breakpoints, it can determine whether the breakpoint interrupt instruction (1B hex) is a software breakpoint or opcode in your target program.

If it is a software breakpoint, execution breaks to the monitor, and the breakpoint interrupt instruction is replaced by the original opcode. A subsequent run or step command will execute from this address.

If it is an opcode of your target program, execution still breaks to the monitor, and an "Undefined software breakpoint" status message is displayed.

When software breakpoints are disabled, the emulator replaces the breakpoint interrupt instruction with the original opcode.

Up to 32 software breakpoints may be defined.

Note

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

Note

Because software breakpoints are implemented by replacing opcodes with the undefined opcode (1B hex), you cannot define software breakpoints in target ROM. You can, however, use the Terminal Interface **cim** command to copy target ROM into emulation memory (see the *Terminal Interface: User's Reference* manual for information on the **cim** command).

Note

Software breakpoints should not be set, cleared, enabled, or disabled while the emulator is running user code. If any of these commands are entered while the emulator is running user code, and the emulator is executing code in the area where the breakpoint is being modified, program execution may be unreliable.

Enabling/Disabling Software Breakpoints

When you initially enter the Softkey Interface, software breakpoints are disabled. 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, the breakpoint interrupt instruction (1B hex) will be placed at the address specified. When the special code is executed, program execution will break into the monitor.

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_I <RETURN>

After the software breakpoint has been set, enter the following command to cause the emulator 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 has been 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 that the breakpoint is inactivated. When breakpoints are hit they become inactivated. To reactivate the breakpoint so that is "pending", you must reenter the "modify software_breakpoints set" command.

```
Software breakpoints  :enabled
    Address        label         status
      01029        cmd_rd:Cmd_I  inactivated










STATUS:   H8/536--Running in monitor      Software break: 001029_______...R....
display  software_breakpoints


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

Clearing a Software Breakpoint

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

modify software_breakpoints clear 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 class, or an individual register.

display registers <RETURN>

```
Registers

Next_PC 01029
 CP 00    TP 00    DP 00    EP 00    SR 0700 <     >   MDCR C7
 PC 1029  SP FE7E  FP 0000  BR 00
 R0 0075  R1 FFFF  R2 0000  R3 0020  R4 2012  R5 FE22  R6 0000  R7      FE7E










STATUS:   H8/536--Running in monitor______Software break: 001029________...R....
display registers


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

You can use "register class" and "register name" to display registers. Refer to "Register Names and Classes" section in chapter 5.

Stepping Through the Program

The step command allows you to step through program execution an instruction or a number of 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 a command appears on the command line, it may be entered by pressing <RETURN>.

```
Registers

Next_PC 0102C
 CP 00    TP 00    DP 00    EP 00    SR 0700 <     >   MDCR C7
 PC 102C  SP FE7E  FP 0000  BR 00
 R0 0075  R1 000F  R2 0000  R3 0020  R4 2012  R5 FE22  R6 0000  R7     FE7E

Step_PC 0102C  MOV:I.W #2024,R4
Next_PC 0102F
 CP 00    TP 00    DP 00    EP 00    SR 0700 <     >   MDCR C7
 PC 102F  SP FE7E  FP 0000  BR 00
 R0 0075  R1 000F  R2 0000  R3 0020  R4 2024  R5 FE22  R6 0000  R7     FE7E

Step_PC 0102F  MOV:I.W #FC02,R5
Next_PC 01032
 CP 00    TP 00    DP 00    EP 00    SR 0708 < n   >   MDCR C7
 PC 1032  SP FE7E  FP 0000  BR 00
 R0 0075  R1 000F  R2 0000  R3 0020  R4 2024  R5 FE02  R6 0000  R7     FE7E

STATUS:   H8/536--Stepping complete____________________________________...R....
step


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

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

run <RETURN>

Using the Analyzer

HP 64700 emulators contain an emulation analyzer. The emulation analyzer monitors 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 point at which the sample program reads the "B" (42 hex) command from the command input byte. To do this, you would trace after the analyzer finds a state in which a value of 42xxh is read from the **Cmd_Input** byte. The following command makes this trace specification.

trace after `Cmd_Input` ***data*** `42xxh` ***status read*** `<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".

Triggering the Analyzer by Data

You may want to trigger the emulation analyzer when a specific data appears on the data bus. You can accomplish this by specifying "**data**" in the "**trace**" command.

You always need to specify the "**data**" with a 16 bits value even when the data access is performed with byte sizes. This is because the emulation analyzer is designed to be able to catch the data on internal 16 bits-width data bus. The following table shows the way to specify the trigger condition by data.

```
 (DATA READ/WRITE)
=====================================================
                    |          | Available
 Location of data   | Accesses | <DATA> Specification
=====================================================
 Internal ROM,RAM   | Word     | HHLL *1
                    +----------+---------------------
                    | Byte     | DDxx *2
 -------------------+----------+---------------------
   Others                      | DDxx
=====================================================

 (INSTRUCTION FETCH)
=====================================================
                    |          | Available
 Location of data   | Address  | <DATA> Specification
=====================================================
 Internal ROM,RAM   | EVEN     | HHLL *1
                    +----------+---------------------
                    | ODD      | xxDD *2
--------------------+----------+---------------------
   Others                      | DDxx *2
=====================================================
*1 HHLL means 16 bits data
*2 DD means 8 bits data
```

For example, to trigger the analyzer when the processor accesses data 12 hex in external ROM, you may use "12xxh" as "**data**" specification.

H8/536 Analysis Status Qualifiers

The status qualifier "read" was used in the example trace command used before in this chapter. The following analysis status qualifiers may also be used with the H8/536 emulator.

```
Qualifier     Status Bits  (36..47)  Description

backgrnd      0xxx xxxx xxxxB        Background cycle
brelease      x111 xxxx xxxxB        Bus release cycle
byte          x110 xxxx xx1xB        Byte access
cpu           x110 xx1x xxxxB        CPU cycle
data          x110 xxxx x1xxB        Data access
dtc           x110 xx0x xxxxB        Data transfer controller cycle
exec          x101 xxxx xxxxB        Instruction execution cycle
fetch         x110 xx1x x001B        Program fetch cycle
foregrnd      1xxx xxxx xxxxB        Foreground cycle
grd           x110 0xx1 xxxxB        Guarded memory access
intack        x011 xxxx xxxxB        Interrupt acknowledge cycle
io            x110 xxx0 xxxxB        Internal I/O access
memory        x110 xxx1 xxxxB        Memory access
read          x110 xxxx xxx1B        Read cycle
word          x110 xxxx xx0xB        Word access
write         x110 xxxx xxx0B        Write cycle
wrrom         x110 x0x1 xxx0B        Write to ROM cycle
```

Displaying the Trace

The trace listings which follow are of program execution on the H8/536 emulator. To display the trace, enter:

display trace <RETURN>

```
Trace List                    Offset=0
Label:      Address       Data    Opcode or Status w/ Source Lines      time count
Base:       symbols       hex            mnemonic w/symbols              relative
after     :Cmd_Input        42FF    42    read  mem byte                  200     nS
+001   :cmd_rds.:+0000D     FFFF  INSTRUCTION--opcode unavailable          80.    nS
+002   :cmd_rds.:+00010     4127    4127  fetch mem                       120     nS
+003   cmd_rds.:Exe_Cmd     FFFF  CMP:E.B #41,R0                           80.    nS
+004   :cmd_rds.:+00012     0640    0640  fetch mem                       200     nS
+005   :cmd_rds.:+00011     FFFF  BEQ cmd_rds.sr:Cmd_A                    120     nS
+006   :cmd_rds.:+00014     4227    4227  fetch mem                        80.    nS
+007   :cmd_rds.:+00013     FFFF  CMP:E.B #42,R0                          120     nS
+008   :cmd_rds.:+00016     0A20    0A20  fetch mem                       200     nS
+009   :cmd_rds.:+00015     FFFF  BEQ cmd_rds.sr:Cmd_B                     80.    nS
+010   :cmd_rds.:+00018     1059    1059  fetch mem                       120     nS
+011   cmd_rds.sr:Cmd_B     0E59      59  fetch mem                       400     nS
+012   :cmd_rds.:+00022     0011    0011  fetch mem                       200     nS
+013   cmd_rds.sr:Cmd_B     FFFF  MOV:I.W #0011,R1                         80.    nS
+014   :cmd_rds.:+00024     5C20    5C20  fetch mem                       120     nS

STATUS:   H8/536--Running user program    Emulation trace    complete______...R....
display trace


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

Line 0 (labeled "after") in the trace list above shows the state which triggered the analyzer. The trigger state is always on line 0. The other states show the exit from the **Scan** loop and the **Exe_Cmd** and **Cmd_B** instructions. To list the next lines of the trace, press the <PGDN> or <NEXT> key.

The resulting display shows **Cmd_B** instructions, the branch to **Write_Msg** and the beginning of the instructions which move the "Entered B command " message to the destination locations.

To list the previous lines of the trace, press the <PGUP> or <PREV> key.

```
Trace List                  Offset=0
Label:      Address       Data    Opcode or Status w/ Source Lines      time count
Base:       symbols       hex            mnemonic w/symbols              relative
+015   :cmd_rds.:+00024     FFFF  MOV:I.W #2012,R4                        80.    nS
+016   :cmd_rds.:+00026     1220    1220  fetch mem                      120     nS
+017   :cmd_rds.:+00028     0659    0659  fetch mem                      200     nS
+018   :cmd_rds.:+00027     FFFF  BRA cmd_rd:Write_Msg                    80.    nS
+019   :cmd_rds.:+0002A     000F    000F  fetch mem                      120     nS
+020   cmd_rd:Write_Msg     245D      5D  fetch mem                      400     nS
+021   :cmd_rds.:+00030     FE02    FE02  fetch mem                      200     nS
+022   cmd_rd:Write_Msg     FFFF  MOV:I.W #FE02,R5                        80.    nS
+023   cmd_rds.sr:Again     C483    C483  fetch mem                      120     nS
+024   cmd_rds.sr:Again     FFFF  MOV:G.B @R4+,R3                         80.    nS
+025   :cmd_rds.:+00034     C593    C593  fetch mem                      120     nS
+026   :cmd_rds.:+00036     07B9    07B9  fetch mem                      400     nS
+027   cmd_rds.sr:Msg_B     45FF    45    read  mem byte                 200     nS
+028   :cmd_rds.:+00034     FFFF  MOV:G.B R3,@R5+                         80.    nS
+029   :cmd_rds.:+00038     F9C5    F9C5  fetch mem                      400     nS

STATUS:   H8/536--Running user program    Emulation trace    complete______...R....
display trace


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

Displaying Trace with No Symbol

The trace listing shown above has symbol information because of the "**set symbols on**" setting before in this chapter. To see the trace listing with no symbol information, enter the following command.

set symbols off

As you can see, the analysis trace display shows the trace list without symbol information.

```
Trace List                  Offset=0
Label:  Address   Data        Opcode or Status w/ Source Lines          time count
Base:     hex     hex                     mnemonic                       relative
after     0FE00     42FF    42    read  mem byte                         200     nS
+001      0100D     FFFF  INSTRUCTION--opcode unavailable                 80.    nS
+002      01010     4127    4127  fetch mem                              120     nS
+003      0100F     FFFF  CMP:E.B #41,R0                                  80.    nS
+004      01012     0640    0640  fetch mem                              200     nS
+005      01011     FFFF  BEQ 01019                                      120     nS
+006      01014     4227    4227  fetch mem                               80.    nS
+007      01013     FFFF  CMP:E.B #42,R0                                 120     nS
+008      01016     0A20    0A20  fetch mem                              200     nS
+009      01015     FFFF  BEQ 01021                                       80.    nS
+010      01018     1059    1059  fetch mem                              120     nS
+011      01021     0E59      59  fetch mem                              400     nS
+012      01022     0011    0011  fetch mem                              200     nS
+013      01021     FFFF  MOV:I.W #0011,R1                                80.    nS
+014      01024     5C20    5C20  fetch mem                              120     nS

STATUS:   H8/536--Running user program    Emulation trace    complete______...R....
set symbols off


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

Displaying Trace with Time Count Absolute

Enter the following command to display count information relative to the trigger state.

display trace count absolute <RETURN>

```
Trace List                  Offset=0
Label:  Address   Data        Opcode or Status w/ Source Lines          time count
Base:     hex     hex                     mnemonic                       absolute
after     0FE00     42FF    42    read  mem byte                       ------------
+001      0100D     FFFF  INSTRUCTION--opcode unavailable              +  80.    nS
+002      01010     4127    4127  fetch mem                            + 200     nS
+003      0100F     FFFF  CMP:E.B #41,R0                               + 280     nS
+004      01012     0640    0640  fetch mem                            + 480     nS
+005      01011     FFFF  BEQ 01019                                    + 600     nS
+006      01014     4227    4227  fetch mem                            + 680     nS
+007      01013     FFFF  CMP:E.B #42,R0                               + 800     nS
+008      01016     0A20    0A20  fetch mem                            +   1.0   uS
+009      01015     FFFF  BEQ 01021                                    +   1.1   uS
+010      01018     1059    1059  fetch mem                            +   1.2   uS
+011      01021     0E59      59  fetch mem                            +   1.6   uS
+012      01022     0011    0011  fetch mem                            +   1.8   uS
+013      01021     FFFF  MOV:I.W #0011,R1                             +   1.9   uS
+014      01024     5C20    5C20  fetch mem                            +   2.0   uS

STATUS:   H8/536--Running user program    Emulation trace    complete______...R....
display trace count absolute


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

Displaying Trace with Compress Mode

If you want to see more executed instructions on a display, the H8/536 emulator Softkey Interface provides **compress mode** for analysis display. To see trace display with compress mode, enter the following command:

display trace compress on <RETURN>

```
Trace List                  Offset=0
Label:  Address   Data        Opcode or Status w/ Source Lines          time count
Base:   hex       hex                     mnemonic                       absolute
after   0FE00      42FF     42    read  mem byte                       ------------
+001    0100D      FFFF   INSTRUCTION--opcode unavailable              +  80.    nS
+003    0100F      FFFF   CMP:E.B #41,R0                               + 280     nS
+005    01011      FFFF   BEQ 01019                                    + 600     nS
+007    01013      FFFF   CMP:E.B #42,R0                               + 800     nS
+009    01015      FFFF   BEQ 01021                                    +   1.1   uS
+013    01021      FFFF   MOV:I.W #0011,R1                             +   1.9   uS
+015    01024      FFFF   MOV:I.W #2012,R4                             +   2.1   uS
+018    01027      FFFF   BRA 0102F                                    +   2.5   uS
+022    0102F      FFFF   MOV:I.W #FE02,R5                             +   3.3   uS
+024    01032      FFFF   MOV:G.B @R4+,R3                              +   3.5   uS
+027    02012      45FF     45    read  mem byte                       +   4.20  uS
+028    01034      FFFF   MOV:G.B R3,@R5+                              +   4.28  uS
+030    0FE02      4545     45    write mem byte                       +   4.88  uS
+031    01036      FFFF   SCB/EQ R1,01032                              +   5.00  uS

STATUS:   H8/536--Running user program    Emulation trace    complete______...R....
display trace compress on


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

As you can see, the analysis trace display shows the analysis trace lists without fetch cycles. With this command you can examine program execution easily.

If you want to see all of cycles including fetch cycles, enter following command:

display trace compress off <RETURN>

The trace display shows you all of the cycles the emulation analyzer have captured.

Note

When the analysis trace is displayed with compress mode, the time count may not indicate correct time counts. This happens when time count is **relative**. Since the compress mode feature is implemented by eliminating fetch cycles when displaying analysis trace, relative time count shows incorrect value. If you are interested in the time count, display with time count **absolute**. Absolute value of time count always show correct value.

Changing the Trace Depth

The default states displayed in the trace list is 256 states. To reduce the number of states, use the "display trace depth" command.

display trace depth 512 <RETURN>

When you enter the following commands, you can see where the program returns to the **Read_Cmd** instruction at state 341.

display trace 341
set symbols on

```
Trace List                  Offset=0
Label:      Address       Data    Opcode or Status w/ Source Lines      time count
Base:       symbols       hex            mnemonic w/symbols          absolute
+334   :cmd_rds.:+0003C     4DFE    4DFE  fetch mem                 +  68.88  uS
+335   :cmd_rds.:+0003E     2226    2226  fetch mem                 +  69.40  uS
+336   :cmd_rds.:+00021     0000    00    write mem byte            +  69.60      uS
+337   :cmd_rds.:+0003C     FFFF  CMP:I.W #FE22,R5                  +  69.68  uS
+338   :cmd_rds.:+00040     F820    F820  fetch mem                 +  69.88  uS
+339   :cmd_rds.:+0003F     FFFF  BNE cmd_rd:Fill_Dest              +  70.00  uS
+340   :cmd_rds.:+00042     C120    C120  fetch mem                 +  70.08  uS
+341   :cmd_rds.:+00041     FFFF  BRA cmd_rds:Read_Cmd              +  70.20  uS
+342           01044        71C6    71C6  fetch mem                 +  70.40  uS
+343   cmd_rds:Read_Cmd     15FE    15FE  fetch mem                 +  70.80  uS
+344   cmd_rds:Read_Cmd     FFFF  MOV:G.B #00,@FE00                 +  70.88  uS
+345   :cmd_rds.:+00006     0006    0006  fetch mem                 +  71.00  uS
+346   :cmd_rds.:+00008     0015    0015  fetch mem                 +  71.20  uS
+347   :cmd_rds.:+0000A     FE00    FE00  fetch mem                 +  71.40  uS
+348      :Cmd_Input        0000    00    write mem byte            +  71.80      uS

STATUS:   H8/536--Running user program    Emulation trace complete______...R....
set symbols on


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

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 available when exiting the Softkey Interface: exiting and releasing the emulation system, exiting with the intent of reentering (continuing), exiting locked from multiple emulation windows, and exiting (locked) and selecting the measurement system display or another module.

End Release System

To exit the Softkey Interface, releasing the emulator so that other users may use the emulator, enter the following command.

end release_system <RETURN>

Ending to Continue Later

You may also exit the Softkey Interface without specifying any options; this causes the emulator to be locked. When the emulator is locked, other users are prevented from using it and 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 using the Softkey Interface from within window systems, the "end" command with no options causes an exit only in 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 have the option to 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 have entered the Softkey Interface via the **emul700** command.

3

In-Circuit Emulation

Many of the topics described in this chapter involve the commands which relate to using the emulator in-circuit, that is, connected to a target system.

This chapter will:

- Describe the issues concerning the installation of the emulator probe into target systems.
- Show you how to install the emulator probe.

We will cover the first topic in this chapter. For complete details on in-circuit emulation configuration, refer to the "Configuring the Emulator" chapter.

Prerequisites

Before performing the tasks described in this chapter, you should be familiar with how the emulator operates in general. Refer to the *HP 64700 Emulators: System Overview* manual and the "Getting Started" chapter of this manual.

Installing the Target System Probe

The emulator probe has a PLCC connector. The emulator is shipped with a pin guard over the target system probe. This guard is designed to prevent impact damage to the pins and should be left in place while you are not using the emulator.

Caution

DAMAGE TO THE EMULATOR CIRCUITRY MAY RESULT IF THESE PRECAUTIONS ARE NOT OBSERVED. The following precautions should be taken while using the H8/536 emulator.

Power Down Target System. Turn off power to the user target system and to the H8/536 emulator before inserting the user plug to avoid circuit damage resulting from voltage transients or mis-insertion of the user plug.

Verify User Plug Orientation. Make certain that Pin 1 of the target system microprocessor socket and Pin 1 of the user plug are properly aligned before inserting the user plug in the socket. Failure to do so may result in damage to the emulator circuitry.

Protect Against Static Discharge. The H8/536 emulator contains devices which are susceptible to damage by static discharge. Therefore, operators should take precautionary measures before handling the user plug to avoid emulator damage.

Protect Target System CMOS Components. If your target system includes any CMOS components, turn on the target system first, then turn on the H8/536 emulator; when powering down, turn off the emulator first, then turn off power to the target system.

Installing into a PLCC Type Socket

To connect the microprocessor connector to the target system, proceeded with the following instructions.

1. Remove the H8/536 microprocessor from the target system socket (PLCC socket). Note the location of pin 1 on the processor and on the target system socket.

2. Store the microprocessor in a protected environment (such as antistatic foam).

3. Install the target system probe into the target system microprocessor socket.

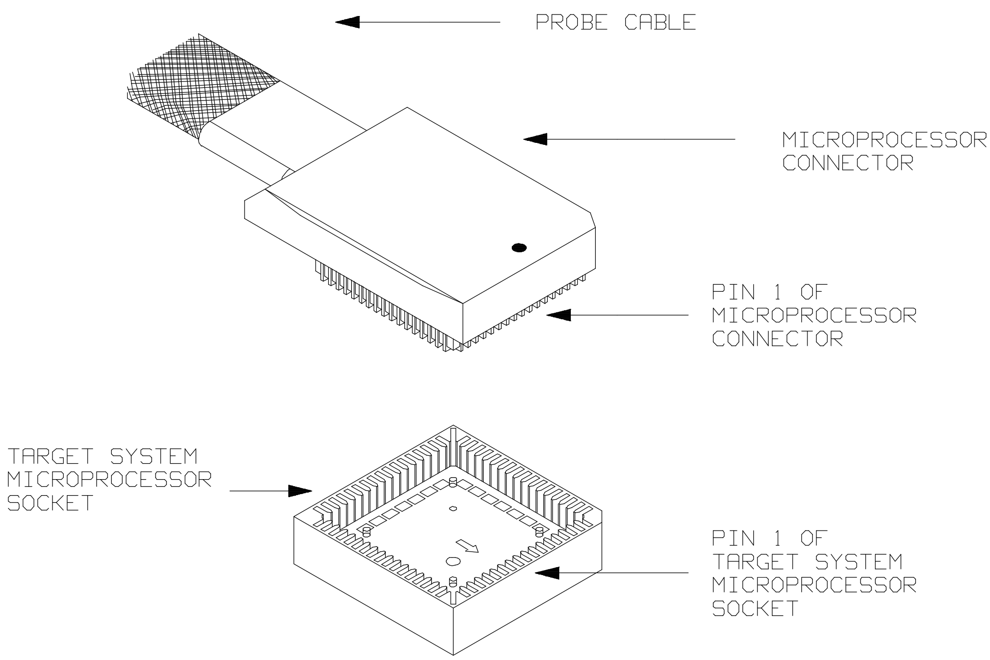

Figure 3-1. Installing into a PLCC type socket

Note

To make sure the contact between emulator probe and target system microprocessor socket, we recommend that you use

ITT CANNON "**LCS-84**" series 84 pin PLCC socket.

In-Circuit Configuration Options

The H8/536 emulator provides configuration options for the following in-circuit emulation issues.
Refer to the "Configuring the Emulator" for more information on these configuration options.

Using the Target System Clock Source

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

Selecting Visible/Hidden Background Cycles

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

Running the Emulator from Target Reset

You can specify that the emulator begins executing from target system reset. When the target system /RES line becomes active and then inactive, the emulator will start reset sequence (operation) as actual microprocessor.

At First, you must specify the emulator responds to /RES signal by the target system (see the "Enable /RES input from the target system?" configuration in Chapter 4 of this manual).

To specify a run from target system reset, select:

run from reset <RESET>

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

Notes

4

Configuring the Emulator

Introduction

The H8/536 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 in place of, or along 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 allow emulator execution to be diverted into the monitor when commands request access of target system resources (target system memory, register contents, etc.)

The emulator is a flexible 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 H8/536 emulator.

The configuration options are accessed 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 and grouped into the following classes.

General Emulator Configuration:

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

Memory Configuration:

- Selecting the background or foreground emulation monitor.
- Mapping memory.

Emulator Pod Configuration:

- Selecting the processor to emulate.
- Selecting the processor operation mode.
- Enabling emulator bus arbitration.
- Enabling NMI input from the target system.
- Enabling /RES input from the target system.
- Allowing the emulator to drive emulation reset to the target system.
- Allowing the emulator to drive background cycles to the target system.
- Selecting the reset value for the stack pointer.

Debug/Trace Configuration:

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

Simulated I/O Configuration: Simulated I/O is described in 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 described in this section involve 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 clock oscillator as the emulator clock source. The emulators' internal clock speed is 10 MHz (system clock).

external Selects the clock input to the emulator probe from the target system. You must use a clock input conforming to the specifications for the H8/536 microprocessor.

Note

Changing the clock source drives the emulator into the reset state. The emulator may later break into the monitor depending on how the following "Enter monitor after configuration?" question is answered.

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 upon completion of the emulator configuration.

How you answer this configuration question is important in some situations. For example, when the external clock has been selected and the target system is turned off, reset to monitor should not be selected; 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 reset to monitor is selected, 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?

If it is important that the emulator execute target system programs in real-time, you can restrict to real-time runs. In other words, when you execute target programs (with the "**run**" command), the emulator will execute in real-time.

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

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

Commands Not Allowed when Real-Time Mode is Enabled

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

- Register display/modification.
- Target system memory display/modification.
- Internal I/O registers display/modification.
- Load/store target system memory.

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

Breaking out of Real-Time Execution

The only commands which are allowed to break real-time execution are:

```
reset
run
break
step
```

Memory Configuration

The memory configuration questions allows you to select the monitor type 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 type configuration question allows you to choose between a foreground monitor (which is supplied with the emulation software but must be assembled, linked, converted, and loaded into emulation memory) or the background monitor (which resides in the emulator).

The *emulation monitor* is a program that is 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, say a command to display target system memory, the system controller writes a command code to the monitor communications area and breaks execution of the emulation processor from the user program into the monitor program. The monitor program then reads the command from the communications area and executes the H8/536 instructions which read the contents of the target system memory locations. After the monitor has completed its task, execution returns to the user program.

The *background monitor*, resident in the emulator, offers the greatest degree of transparency to your target system (that is, your target system should generally be unaffected by monitor execution). However, in some cases you may require an emulation monitor tailored to the requirements of your system. In this case, you will need to use a foreground monitor linked into your program modules. See the "Using the Foreground Monitor" appendix for more information on foreground monitors.

background Selects the use of the background monitor. A memory overlay is created and the background monitor is loaded into that area. When you select the background monitor and the current monitor type is "foreground", you are asked the following question.

Reset map (change of monitor type requires map reset)?

This question must be answered "yes" to change the monitor type.

foreground Specifies that a foreground monitor will be used. Foreground monitor programs are shipped with the Softkey Interface (refer to the "Using the Foreground Monitor" appendix). When you select a foreground monitor, you will be asked additional questions.

Reset map (change of monitor type requires map reset)?

This question must be answered "yes" or else the foreground monitor will not be selected.

Monitor address?

The default configuration specifies a monitor address of 8000 hex. The monitor base address must be located on a 2K byte boundary other than 0 hex; otherwise, configuration will fail.

Monitor filename?

This question allows you to specify the name of the foreground monitor program absolute file. Remember that the foreground monitor must already be assembled and linked starting at the 2K byte boundary specified for the previous "Monitor address?" question.

The monitor program will be loaded after you have answered all the configuration questions; therefore, you should not link the foreground monitor to the user program. If it is important that the symbol database contain both monitor and user program symbols, you can create a different absolute file in which the monitor and user program are linked. Then, you can load this file after configuration.

Mapping Memory

The H8/536 emulator contains 126 kilobytes of high-speed emulation memory (no wait states required) that can be mapped at a resolution of 256 bytes.

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

Blocks of memory can also 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).

The memory mapper allows you to define up to 16 different map terms.

Note

Target system accesses to emulation memory are not allowed. Target system devices that take control of the bus (for example, DMA controllers) cannot access emulation memory.

Note

The default emulator configuration maps location 0 hex through F5FF hex as emulation ROM, and location F600 hex through FEFF hex as emulation RAM. To use the internal ROM and RAM, memory space of these memories must be mapped as emulation memory.

When you answered "yes" to the "Reset map (change of monitor type requires map reset)?" question , you must map again for internal ROM and RAM.

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

Note

You should map all memory ranges used by your programs **before** loading programs into memory. This helps safeguard against loads which accidentally overwrite earlier loads if you follow a **map/load** procedure for each memory range.

Emulator Pod Configuration

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

Modify emulator pod configuration?

Processor type?

This configuration defines the processor to be emulated by the H8/536 emulator.

536	The emulator will emulate the H8/536 microprocessor.
534	The emulator will emulate the H8/534 microprocessor.

Processor operation mode?

This configuration defines operation mode in which the emulator works.

external	The emulator will work using the mode setting by the target system. The target system must supply appropriate input to MD0, MD1 and MD2. If you are using the emulator out of circuit when "external" is selected, the emulator will operate in mode 7.

When mode_1 through mode_7 is selected, the emulator will operate in selected mode regardless of the mode setting by the target system.

Selection	**Description**
mode_1	The emulator will operate in mode 1. (expanded minimum mode)
mode_2	The emulator will operate in mode 2. (expanded minimum mode with internal ROM)
mode_3	The emulator will operate in mode 3. (expanded maximum mode)

mode_4	The emulator will operate in mode 4. (expanded maximum mode with internal ROM)
mode_7	The emulator will operate in mode 7. (single chip mode)

Enable bus arbitration?

The bus arbitration configuration question defines how your emulator responds to bus request signals from the target system during foreground operation. The /BREQ signal from the target system is always ignored when the emulator is running the background monitor. This configuration item is only available for the H8/536 emulator.

yes When bus arbitration is enabled, the /BREQ (bus request) signal from the target system is responded to exactly as it would be if only the emulation processor was present without an emulator. In other words, if the emulation processor receives a /BREQ from the target system, it will respond by asserting /BACK and will set the various processor lines to tri-state. /BREQ is then released by the target; /BACK is negated by the processor, and the emulation processor restarts execution.

Note

You cannot perform DMA (direct memory access) transfers between your target system and emulation memory by using DMA controller on your target system; the H8/536 emulator does not support such a feature.

no When you disable bus arbitration, the emulator ignores the /BREQ signal from the target system. The emulation processor will never drive the /BACK line true; nor will it place the address, data and control signals into the tri-state mode.

Enabling and disabling bus master arbitration can be useful to you in isolating target system problems. For example, you may have a situation where the processor never seems to execute any code. You can disable bus arbitration to check and see if faulty arbitration circuitry in your target system is contributing to the problem.

Enable NMI input from the target system?

This configuration allows you to specify whether or not the emulator responds to NMI(non-maskable interrupt request) signal from the target system during foreground operation.

yes The emulator will respond to the NMI request from the target system.

no The emulator will not respond to the NMI request from the target system.

If you are using the background monitor, the emulator does not accept any interrupt during background execution. All edge-sensed interrupts (include NMI) are latched last one during in background, and such interrupts will occur when context is changed to foreground. All level-sensed interrupts and internal interrupts are ignored during in background operation.

Enable /RES input from the target system

This configuration allows you to specify whether or not the emulator responds to /RES and /STBY signals by the target system during foreground operation.

While running the background monitor, the emulator ignores /RES and /STBY signals except that the emulator’s status is "Awaiting target reset". (see the "Running the Emulation from Target Reset" section in the "In-Circuit Emulation" chapter).

yes The emulator will respond to /RES and /STBY input during foreground operation.

no The emulator will not respond to /RES and /STBY input from the target system.

Note

If you specify that the emulator will drive the /RES signal to the target system during emulation reset or by the overflow of Watch Dog Timer, the emulator should be configured to respond to the /RES input to the target system.

Drive emulation reset to the target system?

This configuration allows you to select whether or not the emulator will drive the /RES signal to the target system during emulation reset.

no Specifies that the emulator will not drive the /RES signal during emulation reset.

yes The emulator will drive an active level on the /RES signal to the target system during emulation reset.

This configuration option is meaningful only when the emulator is configured to respond to the /RES input to the target system. Refer to the "Enable /RES Input from Target?" configuration in this chapter.

Drive background cycles to the target system?

This configuration allows you specify whether or not the emulator will drive the target system bus on background cycles.

If you have selected to use a foreground monitor in "Memory Configuration" section in this chapter, emulator monitor cycles will appear at the target interface exactly as if they were bus cycles caused by any target system program.

no Background monitor cycles are not driven to the target system. When you select this option, the emulator will appear to the target system as if it is between bus cycles while it is operating in the background monitor.

yes Specifies that background cycles are driven to the target system. Emulation processor's address and control strobes (except /WR) are driven during background cycles. Background write cycles won't appear to the target system.

Reset value for stack pointer?

This question allows you to specify the value to which the stack pointer (SP) and the stack page register (TP) will be set on entrance to the emulation monitor initiated RESET state (the "Emulation reset" status).

The address specified in response to this question must be a 20-bit hexadecimal even address.

You **cannot** set this address at the following location.

- Odd address
- Internal I/O register address

When you are using the foreground monitor, this address should be defined in an emulation or target system RAM area which is not used by user program.

Note

We recommend that you use this method of configuring the stack pointer and the stack page register. Without a stack pointer and a stack page register, the emulator is unable to make the transition to the run state, step, or perform many other emulation functions. However, using this option **does not** preclude you from changing the stack pointer value or location within your program; it just sets the initial conditions to allow a run to begin.

Debug/Trace Configuration

The debug/trace configuration questions allows you to specify breaks on writes to ROM, and specify that the analyzer trace foreground/background execution, and bus release cycles. To access the trace/debug 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 will prevent the processor from actually writing to memory mapped as emulation ROM; however, they cannot prevent writes to target system RAM locations which are mapped as ROM, even though the write to ROM break is enabled.

yes Causes the emulator to 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 upon a write to ROM. The emulator will not modify the memory location if it is in emulation ROM.

Note

The **wrrom** trace command status options allow 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 wrrom** <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 option is specified by the default emulator configuration.

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.

Trace bus release cycles?

You can direct the emulator to send bus release cycle data to emulation analyzer or not to send it. This configuration item is only available for the H8/536 emulator.

yes When you enable tracing bus release cycles, bus release cycles will appear as one analysis trace line.

no Bus release cycles will not appear on analysis trace list (display).

Simulated I/O Configuration

The simulated I/O feature and configuration options are described in the *Simulated I/O reference* manual.

Interactive Measurement Configuration

The interactive measurement configuration questions are described in the chapter on coordinated measurements in the *Softkey Interface Reference* manual. Examples of coordinated measurements that can be performed between the emulator and the emulation analyzer are found in the "Using the Emulator" chapter.

External Analyzer Configuration

The external analyzer configuration options are described in the *Analyzer Softkey Interface User's Guide*.

Saving a Configuration

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

Configuration file name? <FILE>

The name of the last configuration file is shown, or 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; the filename specified 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 out of 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

Configuration files which have been previously saved may be loaded with the following 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

In the "Getting Started" chapter, you learned how to load code into the emulator, how to modify memory and view a register, and how to perform a simple analyzer measurement. In this chapter, we will discuss in more detail other features of the emulator.

This chapter discusses:

- Features available via "pod_command".
- Limitations and restrictions of the emulator.
- Register classes and names.
- Debugging C Programs
- Accessing target system devices using E clock synchronous instruction.

This chapter shows you how to:

- Store the contents of memory into absolute files.
- Make coordinated measurements.
- Use a command file.

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 of the most notable 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 the "pod_command". The Softkey Interface, and the configuration files in particular, assume that the configuration of the HP 64700 pod is NOT changed except by the Softkey Interface. Be aware that 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, commands which affect the communications channel should NOT be used at all. Other commands may confuse the protocol depending upon how they are used. 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.

Using a Command File

You can use a command file to perform many functions for you, without having to manually type each function. For example, you might want to create a command file that loads configuration, loads program into memory and displays memory.

To create such a command file, type "**log"** and press TAB key. You will see a command line "**log_commands**" appears in the command field. Next, select "**to**" in the softkey label, and enter the command file name "sample.cmd". This set up a file to record all commands you execute. The commands will be logged to the file sample.cmd in the current directory. You can use this file as a command file to execute these commands automatically.

Suppose that your configuration file and program are named "cmd_rds". To the load configuration:

load configuration cmd_rds <RETURN>

To load the program into memory:

load cmd_rds <RETURN>

To display memory 1000 hex through 1020 hex in mnemonic format:

display memory 1000h ***thru*** 1020h ***mnemonic***

Now, to disable logging, type "**log**" and press TAB key, select "**off**", and press **Enter**. The command file you created looks like this:

```
load configuration cmd_rds
load cmd_rds
display memory 1000h thru 1020h mnemonic
```

If you would like to modify the command file, you can use any text editor on your host computer.

To execute this command file, type "sample.cmd", and press **Enter**.

Debugging C Programs

Softkey Interface has following functions to debug C programs.

- Including C source lines in memory mnemonic display
- Including C source lines in trace listing
- Stepping C sources

The following section describes such features.

Displaying Memory with C Sources

You can display memory in mnemonic format with C source lines. For example, to display memory in mnemonic format from address **_main** with source lines, enter the following commands.

```
display memory _main mnemonic <RETURN>
set source on <RETURN>
```

You can display source lines highlighted with the following command.

```
set source on inverse_video on <RETURN>
```

To display only source lines, use the following command.

```
set source only <RETURN>
```

Specifying Address with Line Numbers

You can specify addresses with line numbers of C source program. For example, to set a breakpoint to line 20 of "main.c" program, enter the following command.

```
modify software_breakpoints set main.c: line
20 <RETURN>
```

Displaying Trace with C Sources

You can include C source information in trace listing. You can use the same command as the case of memory display. For example, to display trace listing with source lines highlighted, enter the following command.

```
display trace <RETURN>
set source on inverse_video on <RETURN>
```

Stepping C Sources

You can direct the emulator to execute a line or a number of lines at a time. For example, to step one line from address **_main**, enter the following command.

step source from `_main <RETURN>`

To step 1 line from the current line, enter the following command.

step source `<RETURN>`

You can specify the number of lines to be executed. To step 5 lines from the current line, enter the following command.

step 5 source `<RETURN>`

E clock synchronous instructions

You can access target system devices in synchronization with the E clock. To do this, use the following commands:

display io_port
modify io_port

The emulator will access the device using the MOVFPE/MOVTPE instruction.

Limitations, Restrictions

DMA Support

Direct memory access to H8/536 emulation memory is not permitted.

Sleep and Software Stand-by Mode

When the emulator breaks into the monitor (foreground/background), the H8/536 sleep or software stand-by mode is released and comes to normal processor mode.

Watch-Dog Timer

When the emulator breaks into background, the emulation processor's watch-dog timer suspends count up in background cycles.

RAM Enable Bit

The internal RAM of H8/536 processor can be enabled/disabled by RAME (RAM enable bit). However, once you map the internal RAM area to emulation RAM, the emulator accesses emulation RAM even if the internal RAM is disabled by RAME.

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 can also store emulation or target system memory to an absolute file with the following command.

store memory `1000h` ***thru*** `1042h` ***to*** `absfile` `<RETURN>`

The command above causes the contents of memory locations 1000 hex through 1042 hex to be stored in the absolute file "absfile.X". Notice that the ".X" extension is appended to the specified filename.

Coordinated Measurements

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

Register Names and Classes

The following register names and classes may be used with "display/modify registers" commands.

Summary

H8/536 register designators. All available register class names and register names are listed below.

BASIC Class

Register name	Description
PC	Program counter
CP	Code page register
SR	Status register
DP	Data page register
EP	Extended page register
TP	Stack page register
BR	Base register
R0	Register R0
R1	Register R1
R2	Register R2
R3	Register R3
R4	Register R4
R5	Register R5
R6	Register R6
R7	Register R6
R7	Register R7
FP	Frame pointer
SP	Stack pointer
MDCR	Mode control register

SYS Class

System control registers

Register name	Description
WCR	Wait control register
RAMCR	RAM control register
MDCR	Mode control register
SBYCR	Software stand-by control register

INTC Class

Interrupt control registers

IPRA	Interrupt priority register A
IPRAB	Interrupt priority register B
IPRC	Interrupt priority register C
IPRD	Interrupt priority register D
IPRE	Interrupt priority register E
IPRF	Interrupt priority register F

DTC Class

Data transfer controller registers

DTEA	DT enable register A
DTEB	DT enable register B
DTEC	DT enable register C
DTED	DT enable register D
DTEE	DT enable register E
DTEF	DT enable register F

PORT Class

I/O port registers

Register name	Description
P1DDR	Port 1 data direction register
P2DDR	Port 2 data direction register
P3DDR	Port 3 data direction register
P4DDR	Port 4 data direction register
P5DDR	Port 5 data direction register
P6DDR	Port 6 data direction register
P7DDR	Port 7 data direction register
P9DDR	Port 9 data direction register
P1DR	Port 1 data register
P2DR	Port 2 data register
P3DR	Port 3 data register
P4DR	Port 4 data register
P5DR	Port 5 data register
P6DR	Port 6 data register
P7DR	Port 7 data register
P8DR	Port 8 data register
P9DR	Port 9 data register
P1CR	Port 1 control register
P69CR	Port 69 control register

FRT1 Class

Free running timer 1 registers

FRTCR1	Timer control register
FRTCSR1	Timer control/status register
FRC1	Free running counter
OCRA1	Output compare register A
OCRB1	Output compare register B
ICR1	Input capture register

FRT2 Class

Free running timer 2 registers

Register name	Description
FRTCR2	Timer control register
FRTCSR2	Timer control/status register
FRC2	Free running counter
OCRA2	Output compare register A
OCRB2	Output compare register B
ICR2	Input capture register

FRT3 Class

Free running timer 3 registers

FRTCR3	Timer control register
FRTCSR3	Timer control/status register
FRC3	Free running counter
OCRA3	Output compare register A
OCRB3	Output compare register B
ICR3	Input capture register

TMR Class

Timer registers

TCR	Timer control register
TCSR	Timer control/status register
TCORA	Timer constant register A
TCORB	Timer constant register B
TCNT	Timer counter

PWM1 Class

PWM timer1 registers

PWMTCR1	Timer control register
DTR1	Duty register
PWMTCNT1	Timer counter

PWM2 Class

PWM timer2 registers

Register name	Description
PWMTCR2	Timer control register
DTR2	Duty register
PWMTCNT2	Timer counter

PWM3 Class

PWM timer3 registers

PWMTCR3	Timer control register
DTR3	Duty register
PWMTCNT3	Timer counter

WDT Class

Watchdog timer registers

WDTCSR	Timer control/status register
WDTCNT	Timer counter
RSTCSR	Reset control/status register

SCI1 Class

Serial communication interface 1 registers.

RDR1	Receive data register
TDR1	Transmit data register
SMR1	Serial mode register
SCR1	Serial control register
SSR1	Serial status register
BRR1	Bit rate register

SCI2 Class

Serial communication interface 2 registers.

Register name	Description
RDR2	Receive data register
TDR2	Transmit data register
SMR2	Serial mode register
SCR2	Serial control register
SSR2	Serial status register
BRR2	Bit rate register

ADC Class

A/D converter registers

Register name	Description
ADDRA	A/D data register A
ADDRB	A/D data register B
ADDRC	A/D data register D
ADDRD	A/D data register D
ADCSR	A/D control/status register
ADCR	A/D control register

A

Using the Foreground Monitor

Introduction

By using and modifying the optional foreground monitor, you can provide an emulation environment which is customized to the needs of a particular target system.

The foreground monitors are supplied with the emulation software and can be found in the following path:

/usr/hp64000/monitor/*

The H8/536 Softkey Interface is provided with four foreground monitor programs. You need to select appropriate monitor program as shown in the following table.

Processor	Processor Mode	Foreground Monitor
H8/536	Mode 1, 2, 7	fmon536min.src
H8/536	Mode 3, 4	fmon536max.src
H8/534	Mode 1, 2, 7	fmon534min.src
H8/534	Mode 3, 4	fmon534max.src

Comparison of Foreground and Background Monitors

An emulation monitor is required to service certain requests for information about the target system and the emulation processor. For example, when you request a register display, the emulation processor is forced into the monitor. The monitor code has the processor dump its registers into certain emulation memory locations, which can then be read by the emulator system controller without further interference.

Background Monitors

A *background* monitor is an emulation monitor which overlays the processor's memory space with a separate memory region. Entry into the monitor is normally accomplished by jamming the monitor addresses onto the processor's address bus.

Usually, a background monitor will be easier to work with in starting a new design. The monitor is immediately available upon powerup, and you don't have to worry about linking in the monitor code or allocating space for the monitor to use the emulator. No assumptions are made about the target system environment; therefore, you can test and debug hardware before any target system code has been written. All of the processor's address space is available for target system use, since the monitor memory is overlaid on processor memory, rather than subtracted from processor memory. Processor resources such as interrupts are not taken by the background monitor.

However, all background monitors sacrifice some level of support for the target system. For example, when the emulation processor enters the monitor code to display registers, it will not respond to target system interrupt requests. This may pose serious problems for complex applications that rely on the microprocessor for real-time, non-intrusive support. Also, the background monitor code resides in emulator firmware and can't be modified to handle special conditions.

Foreground Monitors

A *foreground* monitor may be required for more complex debugging and integration applications. A foreground monitor is a block of code that runs in the same memory space as your program. Foreground monitors allow the emulator to service real-time events, such as interrupts or watchdog timers, while executing in the monitor. For most multitasking, interrupt intensive applications, you will need to use a foreground monitor.

You can tailor the foreground monitor to meet your needs, such as servicing target system interrupts. However, the foreground monitor does use part of the processor's address space, which may cause problems in some target systems. You must also properly configure the emulator to use a foreground monitor (see the "Configuring the Emulator" chapter and the examples in this appendix).

You may link the foreground monitor with your code. However, if possible, linking the monitor separately is preferred. This allows the monitor to be downloaded before the rest of your program. Linking monitor programs separately is more work initially, but it should prove worthwhile overall, since the monitor can then be loaded efficiently during the configuration process at the beginning of a session.

An Example Using the Foreground Monitor

In the following example, we will illustrate how to use a foreground monitor with the sample program from the "Getting Started" chapter. By using the emulation analyzer, we will also show how the emulator switches from state to state using a foreground monitor.

For this example, we will be using the foreground monitor named "fmon536min.src". We will locate the monitor at 8000 hex; the sample program will be located at 1000 hex with the message table at 2000 hex and the command input, message destination, and stack locations at FE00 hex.

At first, you should copy the foreground monitor source file to your current directory and change file mode of the monitor source file.

```
$ cp /usr/hp64000/monitor/fmon536min.src .
<RETURN>
$ chmod 644 fmon536min.src <RETURN>
```

Assemble and Link the Monitor

You can assemble, link and convert the foreground monitor program with the following commands (which assume that **/usr/hp64000/bin** is defined in the PATH environment variable):

```
$ h8asm fmon536min.src  <RETURN>
$ h8lnk fmon536min <RETURN>
$ h8cnvhp -x fmon536min <RETURN>
```

If you haven't already assembled ,linked, and converted the sample program, do that now. Refer to the "Getting Started" chapter for instructions on assembling, linking, and converting the sample program.

Modify Location Declaration Statement (Minimum Modes)

To use the monitor, you must modify the .SECTION statement just after the first comment section of the monitor program listing. You should see the line below:

```
LOCATE_ADRS:    .EQU    H'8000          ;start monitor on 2k boundary
 .SECTION fm536min,CODE,LOCATE=LOCATE_ADRS
```

You can specify the monitor location by modifying this label LOCATE_ADRS. For example, if you want locate the monitor program at 6000 hex, make above line to as below:

```
LOCATE_ADRS:    .EQU    H'6000          ;start monitor on 2k boundary
 .SECTION fm536min,CODE,LOCATE=LOCATE_ADRS
```

Notice that the .SECTION statement is indented from the left margin; if it is not indented, the assembler will attempt to interpret the .SECTION as a label and will generate an error when processing the address portion of the statement. You can load the **fmon536min.src** monitor on a 2k byte boundary of 00800 hex through 0f800 hex.

In this example, we will locate the monitor at 8000 hex. Therefore, you don't have to modify the monitor program.

Modify Location Declaration Statement (Maximum Modes)

When you load the monitor "fmon536max.src" on a 2k byte boundary of 10000 hex through 0ff800 hex, you must change the following statement near the top of the monitor program. Because you cannot define the base address larger than 0FFFF hex with using ".SECTION" command in the monitor program.

```
LOCATE_ADRS             .EQU         H'8000       ;start monitor on 2k boundary
 .SECTION fm536max,CODE,LOCATE=LOCATE_ADRS
;LOCATE_ADRS            .EQU         H'0000
; .SECTION fm536max,CODE
```

You must change the statement as follows to add ";" at the first and second line and to delete ";" at the third and fourth line.

```
;LOCATE_ADRS             .EQU         H'8000       ;start monitor on 2k boundary
; .SECTION fm536max,CODE,LOCATE=LOCATE_ADRS
LOCATE_ADRS              .EQU         H'0000
 .SECTION fm536max,CODE
```

When you link the monitor program, you must define the address where the monitor will be loaded. For example, you may link the monitor program "fmon536max.src" with the following command to load the monitor at the base address 18000 hex.

```
$ h8lnk
:INPUT fmon536max
:START fm536max(01:8000)
:OUTPUT fmon536max
:EXIT
```

Notice that the "START fm536max(01:8000)" statement is used to locate the monitor at the base address 18000 hex.

When you load the monitor "fmon536max.src" on a 2k byte boundary of 00800 hex through 0f800 hex, you can take the same way to use the "fmon536min.src" ; refer to the "Modify Location Declaration Statement (Minimum Modes)" in this appendix.

Modifying the Emulator Configuration

The following assumes you are modifying the default emulator configuration (that is, the configuration present after initial entry into the emulator or entry after a previous exit using "end release_system"). Enter all the default answers except those shown below.

Modify memory configuration? yes

You must modify the memory configuration so that you can select the foreground monitor and map memory.

Monitor type? foreground

Specifies that you will be using a foreground monitor program.

Reset map (change of monitor type requires map reset)? yes

You must answer this question as shown to change the monitor type to foreground.

Monitor address? 8000h

Specifies that the monitor will reside in the 2K byte block from 8000 hex through 87FF hex.

Monitor file name? fmon536min

Enter the name of the foreground monitor absolute file. This file will be loaded at the end of configuration.

Mapping Memory for the Example

When you specify a foreground monitor and enter the monitor address, all existing memory mapper terms are deleted and a term for the monitor block will be added. Add the additional term to map memory for the sample program, and "end" out of the memory mapper.

```
0 thru 7fffh emulation rom <RETURN>
0fb00h thru 0ffffh emulation ram <RETURN>
end <RETURN>
```

See the "Mapping Memory" section of the "Configuring the Emulator" chapter for more information.

Configuration file name? fmcfg

If you wish to save the configuration specified above, answer this question as shown.

Load the Program Code

Now it's time to load the sample program. You can load the sample program with the following command:

load `cmd_rds <RETURN>`

Before running the sample program, you need to initialize the stack pointer by breaking the emulator out of reset:

reset `<RETURN>`
break `<RETURN>`

Now you can run the sample program with the following command:

run from `Init <RETURN>`

Single Step and Foreground Monitors

To use the "**step**" command to step through processor instructions with either of the monitors listed in this chapter, you **must** modify the processor's exception vector table. The entry that you must modify is the trace exception vector. The vector must point to the identifier TRACE_ENTRY in the foreground monitor. You can know the location of TRACE_ENTRY from the assemble listing generated by the assembler.

Address Error During Step Operation

In operation of H8/536 microprocessor, the Stack Pointer (SP) must always contain an even value. Once it becomes to an odd value, an address error will occur. In step operation of H8/536 emulator, if the SP is forced to be an odd value by user program, the emulator will fail to perform step instruction. The emulation processor will read the address error exception vector, and it will continue executing from the address pointed by the vector. If your program doesn't have proper routine to process the address error, the emulation monitor program may run away.

Caution

If the monitor program runs away, try to reset the emulator with **"reset"** command. When the emulator cannot restore control, all you can do is to initialize the emulator. In this case, you will lose all the data in emulation memory.

You can avoid the program run away by using an emulation monitor routine. To use the routine, the address error exception vector in your program must point to ADRSERR_ENTRY of the monitor program.

When the address error occurs, the emulator can break into the monitor by using the routine. However, when the emulator breaks into the monitor in this manner, register values are unreliable. Besides, the SP will contain an odd value.

To continue your measurement, you have to do the following:

- Reset the emulator.

Or:

- Modify registers to proper values by yourself.

When you are using the background monitor, you don't have to worry about this issue. The background monitor can handle it by itself.

Limitations of Foreground Monitors

Listed below are limitations or restrictions present when using a foreground monitor.

Synchronized Measurements

You cannot perform synchronized measurements over the CMB when using a foreground monitor. If you need to make such measurements, select the background monitor type when configuring the emulator.

Index

Notes